Homework 05

Due online through Canvas

Objective

In this homework you will practice using cross-validation and bootstrap in linear regression models.

Instructions

1. You should submit this assignment as a single R script file that produces all the necessary output as required by the questions below. **Important:** your entire code should run without any errors! Having an error that breaks code execution, even if it is a small typo, may result in zero points for your assignment.

2. Make sure to use the template R script file from Canvas as a starting point. Use comments inside your file to explain what you are doing with each part of each question (this will help review your work more efficiently). Make sure to comment out any code that is redundant, e.g. `View()` calls that you use to double check your work.

3. **Important:** if you decide to use any functions that come from external R packages (i.e. packages that are not included in R's default installation), you must adjust part 0 of the template script file to load and install necessary packages. Failing to do so will likely result in your code breaking during grading, which will yield 0 points for your assignment.

4. All questions are extensions of similar issues discussed in class and can be solved in the same way with a few tweaks.

5. Make sure there is no redundant/unused code in your files. Your scripts should only include your comments and code that must be executed to obtain answers. If you create any temporary objects (variables, datasets, etc), make sure to remove them at the end of your code.

Question 1 (70 points)

In this question you will test the limits of bootstrap by applying it to a linear regression model which violates all standard OLS assumption, thus making any standard OLS inference unreliable. As you will see, while bootstrap does allow one compute standard errors and confidence intervals where direct inference does not exist or is unreliable, it is not a magic "make-things-better" button — if your model suffers from lots of structural issues, bootstrap will not offer much improvement over standard OLS inference.

1.1. Start with running the code from section 1.1, which generates a sample from the following model:

$$y_i = 5 + 0.5x_{1i} - x_{2i} + x_{3i} + \epsilon_i$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right) \text{ with } \mu_X = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \text{ and } \sigma_X^2 = \begin{pmatrix} 1 & 0.6 & 0.6 \\ 0.6 & 1 & 0.2 \\ 0.6 & 0.2 & 1 \end{pmatrix}$$

- X variables are generated via `mvrnorm()` function as a multivariate normal distribution with identical means of 3 and variances of 1, with x_1 being generated with relatively high correlation of 0.6 with x_2 and x_3, thus introducing a problem of multicollinearity into OLS estimation.
- Random error ϵ_i is generated as a product of x_{1i} and u_i, where u_i is a bimodal non-normal distribution generated as a 50/50 mixture of two Beta distributions with parameters $\alpha_u = 1.5$ and $\beta_u = 4$ mirrowed around zero:

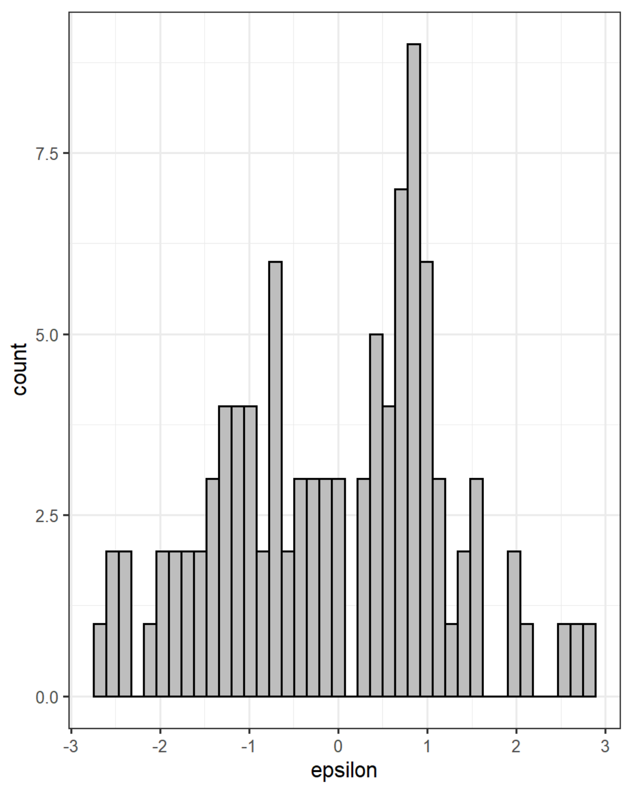
$$\epsilon_i = x_{1i} \cdot u_i$$

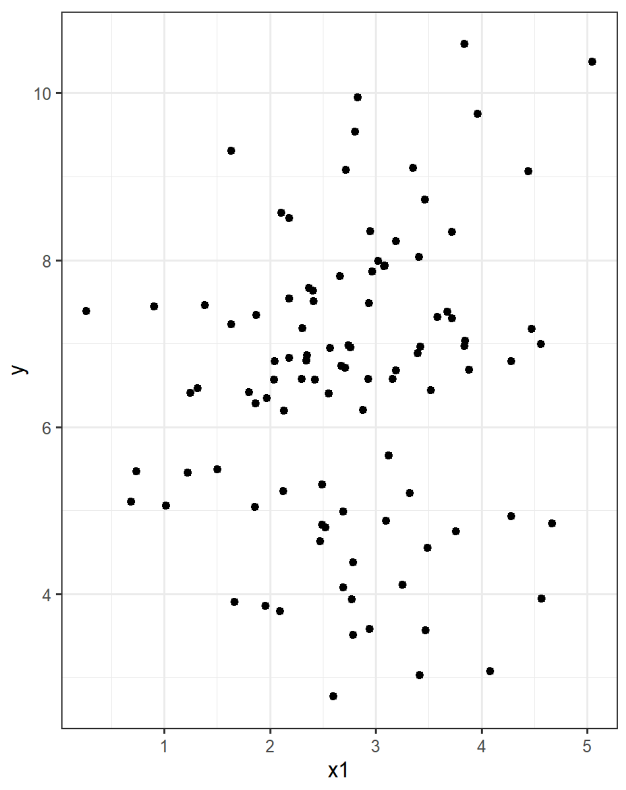
$$u_i \sim \begin{cases} 2 + \mathcal{B}(1.5, 4), & \text{Prob} = 0.5 \\ -2 - \mathcal{B}(1.5, 4), & \text{Prob} = 0.5 \end{cases}$$

  As such, error term in our model is not only non-normal, but it is also heteroscedastic, as the variance of ϵ_i conditional on x_{1i} is a function of x_{1i}:

$$\text{Var}\left(\epsilon_i | x_{1i}\right) = \text{Var}\left(x_{1i} \cdot u | x_{1i}\right) = x_{1i}^2 \cdot \sigma_u^2$$

Taken together, the next two pictures show a sample distribution of model's errors ϵ_i and a scatter plot of pairs $\{x_{1i}, y_i\}$. The distributions of errors has two peaks separated by a trough. Scatter plots shows a very weak linear relationship.





1.2. Estimate linear regression of y on x_1, x_2 and x_3, and save the results into object `q1.reg`. Have a look at your estimation results and note if $\widehat{\beta}_1$ is close to the true value of 0.5 and whether it is significant based on standard OLS inference.

1.3. Because standard small-sample OLS inference for standard errors and the distribution of $\widehat{\beta}_1$ is incorrect (non-normality of the error term + heteroscedasticity), we would like to see first what the true distribution of $\widehat{\beta}_1$ looks like. For this you will need to simulate the same sample 1000 times and estimate 1000 similar regressions.

- Start with creating a dataframe `q1.results` with 5 variables: `type`, `beta1.hat`, `sd`, `ci.l`, `ci.u`. We will use it to summarize and compare the results between standard OLS inference in one sample, in 1000 samples and in 1000 bootstrap samples.
- `q1.results` will contain 3 rows. So, when creating it make the variable `type` contain 3 observations with character values "OLS", "True" and "Boot", and the remaining variables being empty (`NA`).
- Populate the first row of `q1.results` with the values of the variables equal to, correspondingly, "OLS", slope estimate $\widehat{\beta}_1$ from `q1.reg`, OLS standard error for $\widehat{\beta}_1$ from `q1.reg`, lower and upper 95% confidence bounds for $\widehat{\beta}_1$ from `q1.reg`. A suggested way to do it is to use `coef(summary(q1.reg))` and `confint(q1.reg)` commands.
- After this step your `q1.results` should look as follows:

type	**beta1.hat**	**sd**	**ci.l**	**ci.u**
OLS	0.3472348	0.1800168	-0.01009581	0.7045654
True	*NA*	*NA*	*NA*	*NA*
Boot	*NA*	*NA*	*NA*	*NA*

Note: due to random number generation, there might be minor variations in the numbers above for your own system.

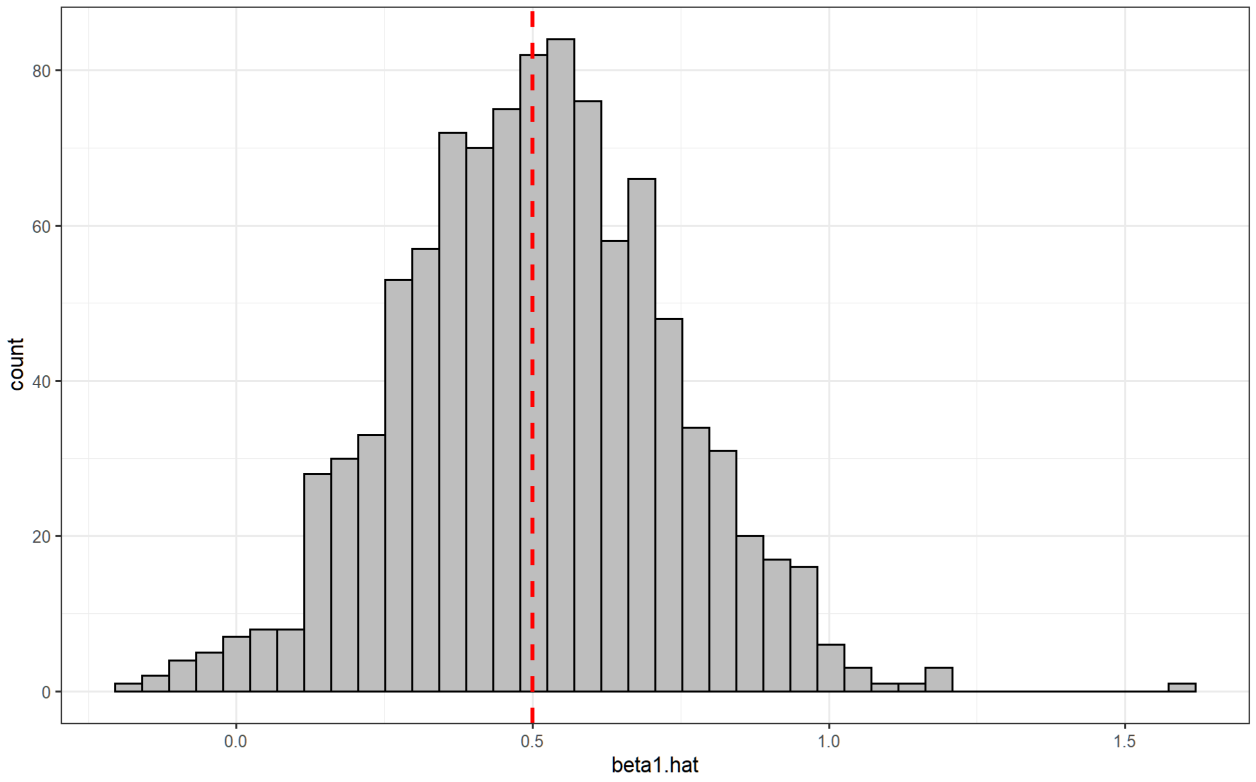
1.4. Next you will need to repeat the process of running our regression `n.sim = 1000` times for `n.sim = 1000` samples generated from the same population. Overall the code you need to use is very similar to the one I used in the lecture on cross-validation and bootstrap, so a simple copy-paste with a few changes should be fine.

- Start with creating a new dataframe `q1.data.sim` to store results from both true and repeated sampling. The dataframe should have `2*n.sim` rows and three variables: `beta1.hat`, `type` and `sim.id`. Variable `type` should take values "True" for first `n.sim` rows and "Boot" for the remaining `n.sim` rows. Variable `sim.id` should run integer numbers from 1 to `n.sim` twice.

1.5. Create a loop that goes through `sim.id` values from 1 to `n.sim` and for each simulation creates a new sample with the same parameters of all variables, estimates the same regression model and stores the value of $\widehat{\beta}_1$ into corresponding row of `q1.data.sim`.

- Start the loop with setting the random number generator seed to something like `100*i`, so that it changes with every loop run.
- Then copy-paste the data generation commands from section Q1.1. (without the `set.seed()` command), only changing the name of the dataframe to be created to `sim.data` instead of `q1.data`.
- Estimate linear regression of y on x_1, x_2 and x_3, and save the results into object `sim.reg`.
- Finally, take the estimate $\widehat{\beta}_1$ from your `sim.reg` and write its value into corresponding row of `q1.data.sim`.

1.6. If your loop runs correctly, you should see the following picture produced by `ggplot` (red dotted line shows true value of β_1):



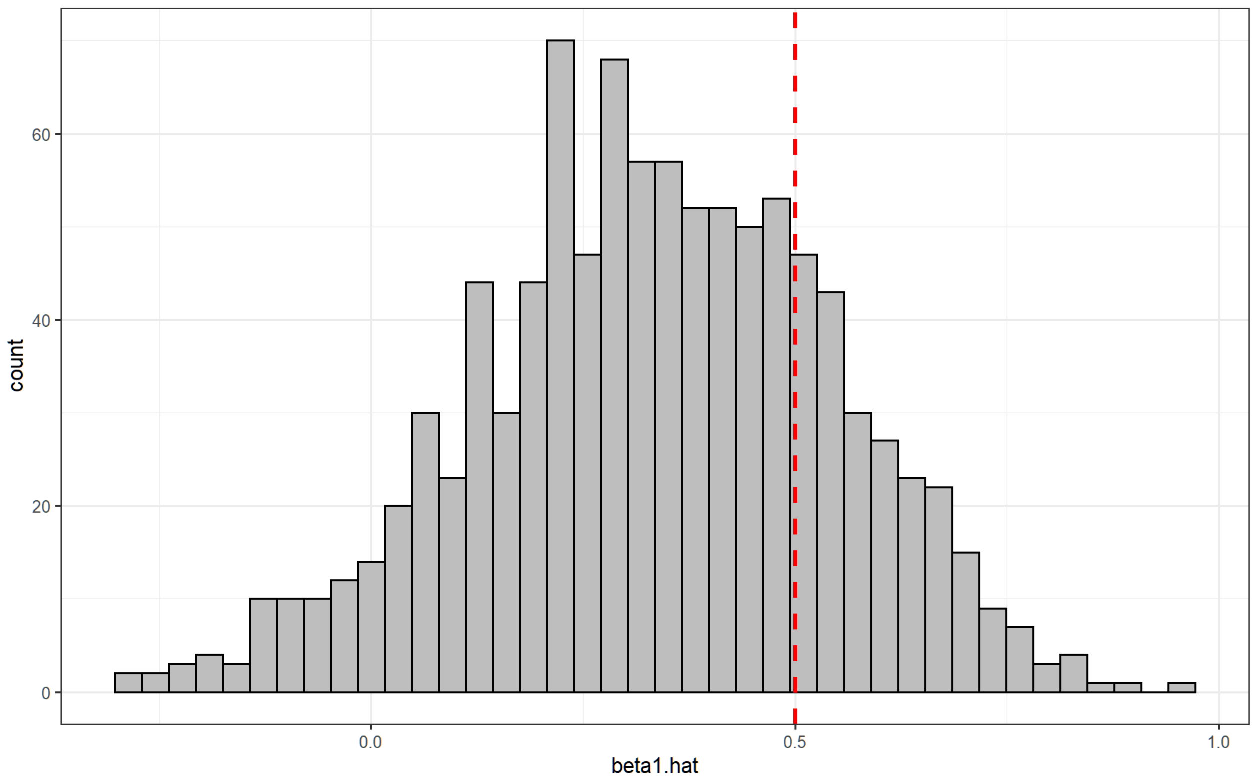
Even though we have two major violations of standard OLS assumtions, the estimator $\widehat{\beta}_1$ still is unbiased and more or less normal, so if we were to have a large sample size, we could probably still get correct inference using standard OLS errors. However, our sample is only 100 observations, and thus we cannot rely on asymptotic properties.

1.7. Use `mean()`, `sd()` and `quantile()` functions to save corresponding numerical statistics from your true samples in `q1.data.sim` into the second row of `q1.results`.

1.8. Now create a loop that will perform 1000 bootstrap samples and estimate 1000 bootstrap regression. See lecture for the code, it can be copied with very minor changes.

- Start the loop with setting the random number generator seed to something like `100*i`, so that it changes with every loop run.
- Create `index` object containing bootstrap row numbers from main data, i.e. row numbers samples with replacement from `q1.data`.
- Create your bootstrap sample as a `boot.data` object by subsetting `q1.data` with `index` as a subset vector.
- Estimate linear regression of y on x_1, x_2 and x_3 using bootstrap sample, and save the results into object `boot.reg`.
- Finally, take the estimate $\widehat{\beta}_1$ from your `boot.reg` and write its value into corresponding row of `q1.data.sim`.

1.9. If your loop runs correctly, you should see the following picture produced by `ggplot`:



Unlike the true distribution above and unlike the example in lectures, this time bootstrapped samples produce a biased estimate — BS distribution is not centered around true value of 0.5, but rather is biased downwards. This is because original OLS estimate from `q1.data` is also smaller than the true value, reflecting the fact that bootstrapped estimate is to original OLS estimate what OLS estimate is to true population value.

There exist quite a large collection of advanced bootstrap methods that allow one to construct more refines estimates and use bootstrap to decrease the possible bias of original OLS estimates. However, that is an area of advanced econometrics that is not very relevant to our course.

1.10. Use `mean()`, `sd()` and `quantile()` functions to save corresponding numerical statistics from your bootstrap samples in `q1.data.sim` into the third row of `q1.results`.

1.11. Compare all the results by viewing the content of `q1.results`. For example, in my simulations I got the following table:

type	**beta1.hat**	**sd**	**ci.l**	**ci.u**
OLS	0.3472348	0.1800168	-0.01009581	0.7045654
True	0.5044480	0.2271833	0.06065080	0.9517024
Boot	0.3329813	0.2108092	-0.10622954	0.7193257

The key takeaway here is the fact that bootstrap standard error of $\widehat{\beta}_1$ is much closer to the true standard error compared to the one from standard OLS estimate. That is because OLS relies on formulas that are no longer valid in our model, while bootstrap tries to quantify uncertainty purely based on the variation in the data.

Question 2 (30 points)

In this question you will use `Wage` dataset that is part of `ISLR` package, containing wage and other data for a group of 3000 male workers in the Mid-Atlantic region from the March 2011 Supplement to Current Population Survey data. The relevant variables in the dataset are as follows:

Variable	**Description**
`age`	Age of worker
`maritl`	Marital status
`race`	Race
`education`	Education level
`jobclass`	Type of job
`health`	Health level of worker
`health_ins`	Whether worker has health insurance
`wage`	Workers raw wage

Your goal is to use 10-fold CV to choose best polynomial form for `age` in a wage regression, and then use bootstrap to calculate standard error of average marginal effect of age.

2.1. Estimate a linear regression model with `wage` being the outcome variable and the other 7 variables from the table above being explanatory variables, and save the result into `q2.reg1` object.

- Since you will be adding polynomial of `age` below, it is recommend to create model's formula using the same approach as in the lecture code — by creating a character string `formula1` with model's formula and then applying `as.formula()` to it inside the `lm()` command.

2.2. Using `residualPlots()` command we can see that `age` definitely can benefit from add some polynomials. Create a dataframe `q2.cv.fit` that contains two variables: `poly` running a sequence of integers from 1 to 5 and `mse` with empty values. There should be a total of 5 rows and 2 columns in this dataframe.

2.3. Run a loop that will perform 10-fold cross-validation using `cv.glm()` function on a `q2.reg1` model with added polynomials of age from 1 to 5.

- The loop is very similar to the one I used in the lecture, so you can copy-paste it and make a few minor changes.

- On every loop run you need to create a new model's formula by taking original `formula1` and adding to it polynomials via the use of `poly()` function.
- Use `glm()` command to estimate your CV regression and save the results into `model.cv`.
- Do 10-fold CV on `model.cv` and save the resulting MSE into corresponding row of `q2.cv.fit`.

2.4. After you are done, compare the results in `q2.cv.fit` and choose the model with smallest MSE (in my case it was `poly == 3`, but in your case it might be something else). Create `formula2` object with adjusted formula that includes corresponding polynomials of `age` and estimate adjusted model `q2.reg2`.

2.5. In case of cubic polynomial for *age* the model had the following coefficients (other variables omitted for convenience):

$$\widehat{wage_i} = \ldots + 3.41 \cdot age_i - 0.0445 \cdot age_i^2 + 0.00013 \cdot age_i^3 + \ldots$$

This means that the marginal effect of age is no longer constant and depends on the specific values of age_i for worker i:

$$\text{ME}(age) = \frac{\Delta\widehat{wage_i}}{\Delta age_i} \approx 3.41 - 2 \cdot 0.0445 \cdot age_i + 3 \cdot 0.00013 \cdot age_i^2$$

If we want some aggregate measure, the simplest way would be to plug in a median value of age_i and call that a 'marginal effect of age for a worker with median age'. To do so, define a function `age.me(data, index)` that takes two arguments: `data` with the dataset and `index` with the sequence of row numbers to be selected from that dataset. The function then proceeds to calculate the marginal effect of age for a median worker based on a regression model estimated using `data[index,]`.

- The function `age.me()` serves the same purpose as the function `alpha.boot()` from the lecture — you will use it to run `boot()` command.
- Inside the function first create `boot.data` object as a subset of `data` with row numbers specified in `index`.
- Then run the same model as in `q2.reg2`, but using `boot.data` instead, and save the result into `boot.model`.
- Extract necessary coefficients from `boot.model` to be used in the formula for ME(*age*) above, i.e. polynomial coefficients on *age*, age^2 and so on.
- Calculate median value of *age* in `boot.data` and plug it into the formula for ME(*age*) alongside with estimated coefficients. Return the value from the formula as the function's output.

If your best model is a cubic polynomial, then the command `age.me(q2.data, 1:nrow(q2.data))` should produce the value of 0.3735526.

2.6. Finally, run the command `boot()` with data from `q2.data`, statistic defined by your `age.me()` function and 1000 bootstrap iterations. Save the result into object `q2.boot`, then look at the summary of that object.

- In my case bootstrap standard error for ME(age) of a worker with median age is around 0.092666, which means that t-statistic for significance is around 4, making it a very strong effect overall.