Final Assignment

Marc Kaufmann

11/12/2019

Grading

You will submit this assignment twice, and assess each other in between:

- First submission Wednesday November 27th
  - Worth 4.4% of total grade (40% of 11%)
- Assess by December 2nd
  - You submit *a single pdf page* as feedback
  - I will ignore everything past the first page, without exception
  - Worth 6.6% of total grade (60% of 11%)
- Second submission based on feedback December 9th
  - Worth 12% of total grade

The grading is such that you can not lose too much on the first submission and so that you can improve your submission for the final submission based on the feedback you receive from others. Therefore the feedback you give matters for your grade, since it should help the person receiving it.

It also means that you should attempt all parts on your first submission, so that you can receive helpful feedback on all parts.

Instructions

This time, upload your .Rmd file to Moodle, which will get run/knitted.

The Bootstrap

You will implement the bootstrap without using specialized libraries to do so (e.g. `boot`). That way, you have to (try to) understand what the bootstrap does and how it works.

We will use the bootstrap to compute the uncertainty of our estimates. For instance, if we find that the mean price of hotels is $100, what does that mean? How 'confident' are we in this? The mean can be $100 because all the hotels cost exactly $100 per night, or because one half costs $20 and the other half $180.

This is of course what the standard deviation tells us. But... how do we compute the standard deviation of the mean? If you run a regression model, the `summary()` function will provide you with an estimate of the uncertainty, usually the standard error. The problem is that the standard error depends on rather strong assumptions. When these assumptions are violated (that errors are Gaussian or uncorrelated or your model is wrong) then the standard errors are wrong – usually by overstating the certainty, or understating the uncertainty.[1]

Before diving in: What is it that we want the standard error to tell us? We want it to answer the following question: "If I gather new data of this kind, how far away am I going to be from my current mean estimate, most of the time?" The *most of the time* part often means *95% percent of the time* by custom – there is no objective reason that it should be 95% percent. The bootstrap makes it easy to change your definition of *most of the time*, but we won't go there.

[1] *Sidenote:* Ideally, I would do the following exercise with confidence intervals as well as standard errors, but I believe that may complicate a lot and interfere with understanding the bootstrap. So we'll stick with standard error.

If we had unlimited resources, how could we estimate this uncertainty? Well, just gather the data. Go out and repeat the experiment. For instance, we might find that the average price is $100 for hotels in Vienna. But we got that data for November 2017, say. So we can go out and gather the data on other nights, for other subsamples of the hotels. Suppose you repeat this for 100 days over 10 years. Now you have 100 datasets, and for each of them, you compute the average hotel price. How does this give you your measure of uncertainty?

Well you have dataset1, dataset2, . . . , dataset100, with mean1, mean2, mean3, . . . mean 100. So define

```
all_means <- c(mean1, mean2, ..., mean100)
```

and then the standard error (which is a fancy name for 'standard deviation of the estimator', to distinguish it from the standard deviation of the dataset) is simply `sd(all_means)`. That's it. Of course this measure isn't perfect: ideally you would want to know what happens when you do this an infinite number of times, but you get the idea.

Since in reality we cannot replicate every dataset hundreds or thousands (or even a couple) of times, we have to find other ways. Bradley Efron had the following brilliant idea: take the initial dataset1. Instead of going out and gathering *new* datasets, *simulate* a new dataset by sampling datapoints from the only dataset you have!

For instance, if our initial vienna dataset contains only 6 hotels (to make the illustration easy), we can generate a new dataset of 6 hotels by picking a hotel as follows:

1. Roll dice. Pick the hotel with the number given by the dice
2. Roll dice. Pick the hotel with the number given by the dice, *even* if you already picked that hotel
3. Do this 4 more times

Now you have dataset2. You can compute the mean for dataset2. That gives you mean2. Repeat this one hundred times. Or a thousand, since the main cost is simulation, not actual data gathering.

Once you have your B (for bootstrap) simulations and simulated means, you can compute the standard deviation of the means - and voila, you are done!

What if you want to do this for regression coefficients? Well, you generate a new dataset, then you run the same regression on the new dataset, which gives you a new value of the coefficient. You repeat B times. Now you compute the standard deviation of these B coefficients. Done.

Notice that you can do this for *every* statistic that you can compute: i.e. if a function takes a vector (or dataframe) and returns a single double, you can compute a value for every dataset and thus compute the standard deviation of all.

Sidenote: One problem with the above approach is that I tell you to resample from the full dataset. I believe that this is fine for simple statistics such as the mean - but it would not be appropriate for generating bootstrap intervals for p-values, or predictive accuracy, since for that exercise you need to have one training and one testing data set. However you can still bootstrap in that case by simulating training and testing data sets (but of course this can never help you with external validity, since you can't simulate data that you don't have).

Your mission, whether or not your are willing to accept it, is to write the code for the bootstrap as described above, by creating functions that allow you to compute the bootstrap for lots and lots of different functions (ideally for any function that takes a dataframe and returns a number).

1. Write the code that does what I describe above for the vienna dataset. That is, compute the mean price and compute the bootstrap standard deviation of this mean (aka 'the standard error'). Don't worry yet about writing functions that can be used for more general things.
2. Write a function `bootstrap_mean` that takes two arguments:

- `B`: the number of simulations/replications you want to run (how many datasets to generate)
- `v`: the column on Vienna prices (or more generally whatever column you want to compute the mean of), the one from which you sample the simulated datasets Use this and plot the standard error against B.

3. What is the non-bootstrap standard error? How can you get it out of R? (Hint: The mean function doesn't give it to you, but you can run a linear regression of a special kind that essentially only computes the average. You can solve this question differently as well.)
4. Compute the bootstrap standard error for the `median()` rather than `mean()`. Define a function `bootstrap_median` for this which takes again an input vector v for which you compute the mean, as well as a parameter B.
5. Compute the bootstrap standard error for:

- `mean()` and
- `median()` and
- the top quartile and
- the standard deviation of the price (yes, I want the standard deviation of the standard deviation... If this confuses you, go through the mean example and replace the computation of the `mean` by `the_thing_we_want` and realize that `the_thing_we_want` can be the standard deviation)
- `max()` One way to approach this is to define a new function for each. Another is to write a `bootstrap_func` function that takes an additional argument called `fun`, and then you call it `bootstrap_func(B, v, median)` to have the same effect as `bootstrap_median`. Implement this function `bootstrap_func`.

Example calls to this function: `bootstrap_func(1000, vienna_data$price, mean)` and `bootstrap_func(1000, vienna_data, some_function_that_takes_a_dataframe_as_argument)`. Notice that the second argument can be a vector or a dataframe, and this depends on what function you pass in. The mean requires a vector, the `some_function_that_takes_a_dataframe_as_argument` takes a dataframe. Computing linear model coefficients requires a dataframe for instance. 6. Use your new function to compute bootstrap estimators for the standard errors of some linear model coefficients on the vienna dataset. You have to define and name a function that returns the *coefficient* of the right linear model, and pass this function as one of the arguments to `bootstrap_func`. 7. Compare the bootstrap estimators to the ones that `summary(lm(...))` spits out. Show the output that gives these estimators.

You are done. For the second week you should make your functions as user-friendly as possible, using feedback from others on how to do this.

Note: The current assignment might make you think that the bootstrap always works, or is the right tool for all things. This is not the case, although it is beyond the scope of the assignment to show why. Sometimes the bootstrap can be badly off. Read Chapter 6 part 7 of Advanced Data Analysis from an Elementary Point of View by Cosma Shalizi for more. In addition, even when it is not misleading, because the bootstrap makes fewer assumptions, it is less precise - i.e. the bootstrap standard errors are larger than the ones you get by default. This is usually a good thing, since the usual assumpations are often violated. However, when you do believe that some assumption makes sense, you should use the knowledge it provides to get tighter bounds.