STAT-S 432: Homework 3

Due February 22, 2019 by 11:59 PM

Instructions: You must submit this homework by pushing a file named “hw3.Rmd” file to your team’s repo. Note that that is the **only** file you will be allowed to push. Commit early and often.

Data named uval.csv

“Gross domestic product” is a standard measure of the size of an economy; it’s the total value of all goods and services bought and solid in a country over the course of a year. It’s not a perfect measure of prosperity, but it is a very common one, and many important questions in economics turn on what leads GDP to grow faster or slower.

One common idea is that poorer economies, those with lower initial GDPs, should grower faster than richer ones. The reasoning behind this “catching up” is that poor economies can copy technologies and procedures from richer ones, but already-developed countries can only grow as technology advances. A second, separate idea is that countries can boost their growth rate by under-valuing their currency, making the goods and services they export cheaper. This week’s data set contains the following variables:

- Country, in a three-letter code (see http://en.wikipedia.org/wiki/ISO_ 3166-1_alpha-3).
- Year (in five-year increments).
- Per-capita GDP, in dollars per person per year (“real” or inflation-adjusted).
- Average percentage growth rate in GDP over the next five years.
- An index of currency under-valuation
  - The index is 0 if the currency is neither over- nor under- valued, positive if under-valued, negative if it is over-valued.Note that not all countries have data for all years. However, there are no missing values in the data table.

1. Problem 1 will use the same dataset as before. Now you are going to use kernel regression. You will be comparing the 2nd linear regression to the model you created in Homework 2 to a kernel regression that uses the same predictors. Use kernel regression, as implemented in the np package, to non-parametrically regress growth on log GDP, under-valuation, country, and year (treating year as a categorical variable). Set `tol` to about 10^{-3} and `ftol` to about 10^{-4} in the npreg command, and allow several minutes for it to run. (You’ll probably want to cache this part of your code.)

   - Give the coefficients of the kernel regression, or explain why you can’t.

   - Plot the predicted values of the kernel regression against the predicted values of the linear model. Describe the trend in the plot, if any. What does this plot tell you about the bias or lack of bias in the linear model

   - Plot the residuals of the kernel regression against its predicted values. Should these points be scattered around a flat line, if the model is right? Are they?

   - The npreg function reports a cross-validated estimate of the mean squared error for the model it fits. What is that? Does the kernel regression predict better or worse than the linear model with the same variables?

   - Plot the predictions for growth against each variable, holding the other variables fixed at their medians. You can obtain these plots using the plot command and the kernel regression object from npreg. Is there evidence that log(GDP) or under-valuation are strongly related to growth? Which variables do show a strong relationship with growth?

2. Problem 2 has you use the starter code provided. The code will generate a data set to be used for this problem and will also provide a true mean function `mu(x)`. The resulting data frame has an `x` column (the predictor) and a `y` column (the response).

- Plot y versus x. Overlay the true mean function `mu(x)` using the curve function. What do you notice for $x < 4\pi$ and $x > 4\pi$
- Using the `np` library in `R`, fit a separate kernel regression on each of the following datasets:
  - Only those datapoints with $x < 4\pi$
  - Only those data points with $x > 4\pi$
  - All the data points

  For each of these regressions, what is the optimal bandwidth (report as a table)? How does the optimal bandwidth for the overall dataset compare to the optimal bandwidth for each of the halves?
- For each of the three selected bandwidths make a plot showing the true mean $\mu(x)$, the data points, and the kernel regression predictions using the bandwidth. The result should be three plots, each tuned to one of the selected bandwidths. Give these plots clear titles to distinguish them. You may need to make the plots reasonably large to see details. To include bigger plots in your homework use the `fig.width` and `fig.height` code chunk settings of Rmarkdown.
- How do the three plots you created differ? In particular, how well do the regressions trained on the left and right halves do on their respective datasets? How well does the bandwidth from the overall dataset fit on each half? Be specific about any problems that occur. What lesson might this tell about functions of varying smoothness and kernel regression, if any?

```r
### Starter code for Homework 3, last problem ###

# Our true mean function: will be sin(x/2) on [0,4*pi] and sin(6*x) on [4*pi,8*pi]
# Input: a vector of real numbers (x)
# Output: the vector of function values at x
mu = function(x){
   # Initialize a vector of zeros
   y = numeric(length(x))
   # Figure out which points are to the left or right of 4*pi
   left_points = (x<=4*pi)
   right_points = (x>4*pi)
   # Assign the appropriate sine values
   y[left_points] = sin(x[left_points]/2)
   y[right_points] = sin(6*x[right_points])
   # Return y
   y
}

# A function to draw a sample from this curve
# Input: number of samples to draw (n)
# Output: data frame with columns named "x" and "y"
generate_sample = function(n){
   # Sample the x coordinates uniformly on [0,8*pi].
   # We sort the x here to make plotting easier later.
   x = sort(runif(n,0,6*pi))
   # Sample the y coordinates as Gaussians around mu(x)
   y = mu(x) + rnorm(n,0,.2) # standard deviation of the noise is hard coded
   # Bind this all together into a data frame
   data.frame(x=x,y=y)
}

# We set the seed so that your homeworks will match
set.seed(9781)
```

```
# Sample 300 points.  This is your data set!
data = generate_sample(300)
```