Hwk4_bs2996

Bangda Sun

October 10, 2016

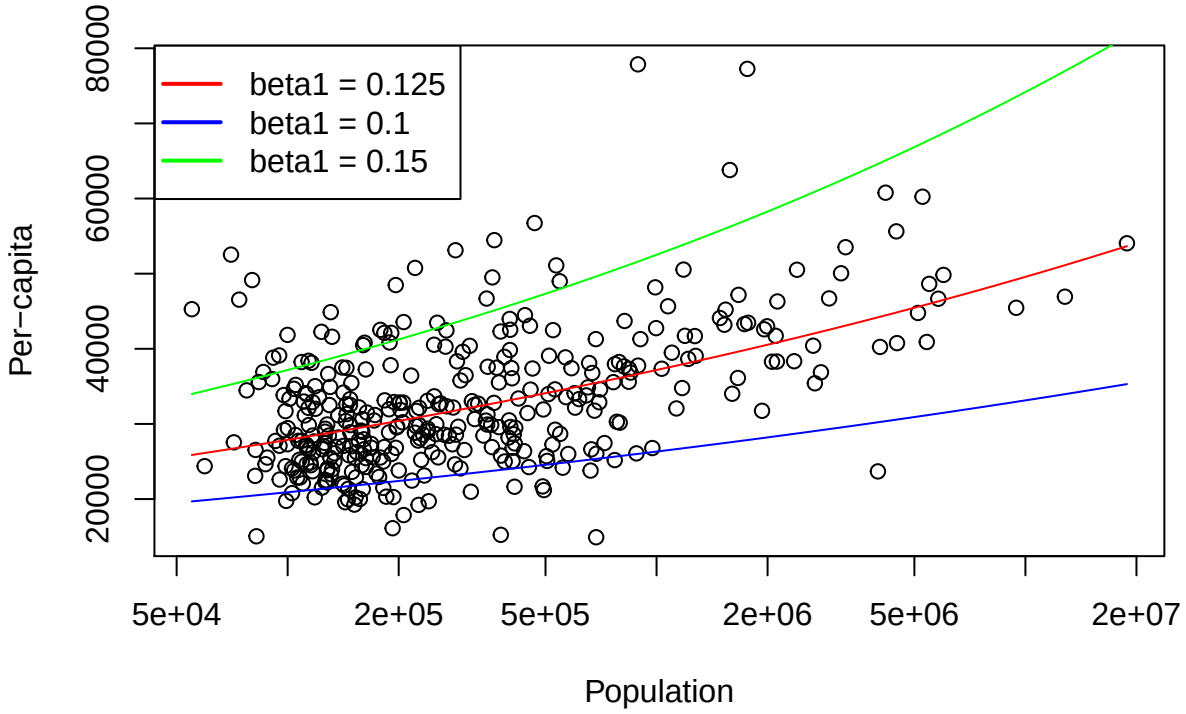
i. Plot the data as in lecture, with per-capita GMP on the y-axis and population on the x-axis. Using the `curve()` function to add the model (1) using the default values

provided in lecture. Add two more curves using $\beta_1 = 0{:}1$ and $\beta_1 = 0{:}15$ with the same value of β_0 from class. Make all three curves different colors using the `col` option.

```
set.seed(1)
setwd("C://Users//Bangda//Desktop//GR5206 Materials//Hwk4")
gmp <- read.table("gmp.txt", header = TRUE)
head(gmp)
```

```
##                          city        gmp pcgmp
## 1                 Abilene, TX 3.8870e+09 24490
## 2                   Akron, OH 2.2998e+10 32889
## 3                  Albany, GA 3.9550e+09 24269
## 4 Albany-Schenectady-Troy, NY 3.1321e+10 36836
## 5             Albuquerque, NM 3.0727e+10 37657
## 6              Alexandria, LA 3.8790e+09 25494
```

```
gmp$pop <- round(gmp$gmp/gmp$pcgmp)
plot(gmp$pop, gmp$pcgmp, log = "x", ylab = "Per-capita", xlab = "Population")
curve(6611*x^(1/8), add = TRUE, col = "red")
curve(6611*x^(0.1), add = TRUE, col = "blue")
curve(6611*x^(0.15), add = TRUE, col = "green")
legend(x = "topleft", legend = c("beta1 = 0.125", "beta1 = 0.1", "beta1 = 0.15"),
       col = c("red", "blue", "green"), lwd = 2, pch = c(NA, NA, NA))
```



ii. Write a function called `mse()` which calculates the mean squared error if the model on a given dataset. `mse()` should have three arguments: (1) a numeric vector of length two, with the first component corresponding to β_0 and the second to β_1, (2) a numeric vector containing the population values (X), and (3) a numeric vector containing the values of the per-capita GMP (Y). The output of your function should be a single numeric value (the mean squared error). The second and third arguments should have as default values the variables pop and pcgmp, respectively, from the gmp dataset. Your function may not use a loop (neither for nor which). Check that using the default data your function returns the following values:

```
> mse(c(6611, 0.15))
[1] 207057513
> mse(c(5000, 0.10))
[1] 298459914
```

```
mse <- function(beta, X = gmp$pop, Y = gmp$pcgmp){
  # The arguments of the function are
  # 1. beta0 and beta1 in a vector
  # 2. X: (default) population values
  # 3. Y: (default) per-capita CMP
  # return the mse
  stopifnot(length(beta) == 2)
  sse <- sum((Y - beta[1]*X^(beta[2]))^2)
  mse <- sse / length(Y)
  return(mse)
```

```
}
mse(c(6611, 0.15))
```

```
## [1] 207057513
```

```
mse(c(5000, 0.10))
```

```
## [1] 298459914
```

iii. R has several built-in functions for optimization which we'll talk about later on in the course. One of the simplest, which we use today, is `nlm()`, or non-linear minimization. `nlm()` takes two required arguments, a function to minimize and a starting value for that function. Run `nlm()` three times with your function `mse()` and three starting pairs for β_0 and β_1 as in

```
nlm(mse, c(beta0 = 6611, beta1 = 1/8))
```

What do the output quantities minmum and estimate represent? Check `?nlm` for help. What values does the call return for these? Note that you can access these elements using the dollar sign `$` as in the following:

```
nlm(mse, c(beta0 = 6611, beta1 = 1/8))$estimate
```

```
est1 <- nlm(mse, c(beta0 = 6611, beta1 = 1/8))
est2 <- nlm(mse, c(beta0 = 6611, beta1 = 0.10))
est3 <- nlm(mse, c(beta0 = 6611, beta1 = 0.15))
```

The minimum is the minimum value of mse, estimates are the value of beta0 and beta1 at the minimum point. These two quantities of three β_1 is respectly

```
list(est1.mini = est1$minimum, est1.ests = est1$estimate,
     est2.mini = est2$minimum, est2.ests = est2$estimate,
     est3.mini = est3$minimum, est3.ests = est3$estimate)
```

```
## $est1.mini
## [1] 61857060
##
## $est1.ests
## [1] 6611.0000000    0.1263177
##
## $est2.mini
## [1] 61857060
##
## $est2.ests
## [1] 6611.0000003    0.1263177
##
## $est3.mini
## [1] 61857060
##
## $est3.ests
## [1] 6610.9999997    0.1263182
```

iv. Using `nlm()` and the `mse()` function you wrote, write a function `plm()` which estimates the parameters β_0 and β_1 of the model by minimizing the mean squared error. The function `plm()` should take the following four arguments: (1) an initial guess for β_0, (2) an initial guess for β_1, (3) a vector containing the population values (X), and (4) a vector containing the per-capita GMP values (Y). All arguments except for the initial guesses should have suitable default values. It should return a list with the following three components: (1) the final guess for β_0, (2) the final guess for β_1, and (3) the final value of the MSE. Your function must call those you wrote in earlier questions (as opposed to simply repeating the code) and the appropriate arguments to `plm()` should be passed on to them. What parameter estimate do you get when starting from $\beta_0 = 6611$ and $\beta_1 = 0.15$? From $\beta_0 = 5000$ and $\beta_1 = 0.10$? If these are not the same, why do they differ? Which estimate has a lower MSE?

```
plm <- function(beta0, beta1, X = gmp$pop, Y = gmp$pcgmp){
  # This function has 4 arguments:
  # 1. initial value of beta0
  # 2. initial value of beta1
  # 3. X: (default) population values
  # 4. Y: (default) per-capita CMP
  # return the list including:
  # 1. final value of beta0
  # 2. final value of beta1
  # 3. final value of MSE
  stopifnot(length(beta0) == 1, length(beta1) == 1)

  # use nlm() to minimize the mse, the input of mse() is same as the plm()
  est        <- nlm(mse, c(beta0, beta1), X = X, Y = Y)

  # return the optimal value of beta0 and beta1 and the mse
  returnlist <- list(est.beta0 = est$estimate[1],
                     est.beta1 = est$estimate[2],
                     mse = est$minimum)
  return(returnlist)
}
# start from (6611, 0.15)
plm(6611, 0.15)
```

```
## $est.beta0
## [1] 6611
##
## $est.beta1
## [1] 0.1263182
##
## $mse
## [1] 61857060
```

```
# start from (5000, 0.1)
plm(5000, 0.10)
```

```
## $est.beta0
## [1] 5000
##
## $est.beta1
## [1] 0.1475913
```

```
##
## $mse
## [1] 62521484
```

Since their starting point are different, it might convergent to different point (or convergent to local minimum point rather than global minimum point). The former one has a lower MSE.

v. Let's practice the bootstrap in a simple example to convince ourselves, again, that it will work.

(a) Calculate the mean and standard deviation of the per-capita GMP values in your dataset using built-in function `mean()` and `sd()`. Using these values calculate the standard error of the mean.

```
pcgmp.mean <- mean(gmp$pcgmp); pcgmp.mean
```

```
## [1] 32922.53
```

```
pcgmp.sd <- sd(gmp$pcgmp); pcgmp.sd
```

```
## [1] 9219.663
```

```
pcgmp.se <- pcgmp.sd / sqrt(length(gmp$pcgmp)); pcgmp.se
```

```
## [1] 481.9195
```

(b) Write a function which takes in a vector `indices` of length n, where n is the number of per-capita GMP values in our dataset, and calculates the mean per-capita GMP for the cities indicated by the indices. For example if `indices = c(1, 5, 1, 3)`, then your function should calculate the mean `c(24490, 37657, 24490, 24269)`, the per-capita GMP for the first, fifth, first again, and third cities.

```
mean.cal <- function(indices){
  # this function create one bootstrap sample
  n <- dim(gmp)[1]
  stopifnot(length(indices) == n)
  mean.cal <- mean(gmp$pcgmp[indices])
  return(mean.cal)
}
```

(c) Using the function in (b) create a vector `bootstrap.means`, which has the mean per-capita GMP for one hundred bootstrap samples. Here you'll want to create a loop where each iteration performs a new bootstrap sample. In each iteration you use `sample()` to create an indices vector containing the indices of your bootstrap sample and then using indices calculate the mean using your function from (b).

```
bootstrap.mean <- rep(NA, 100)
for (i in 1:100){
  indices <- sample(1:dim(gmp)[1], replace = TRUE)
  bootstrap.mean[i] <- mean.cal(indices)
}
head(bootstrap.mean)
```

```
## [1] 32732.65 32687.77 32024.66 33258.84 32560.50 33875.90
```

(d) Calculate the standard deviation of the bootstrap.means to approximate the standard error from part (a). How well does this estimate match the value form part (a)?

```
bootstrap.se <- sd(bootstrap.mean); bootstrap.se
```

```
## [1] 435.2417
```

We can see this approximation is a little bit lower (decrease 9.69%) than the estimate value in part(a).

vi. Write a function `plm.bootstrap()`, the calculate a bootstrap estimates of the standard errors of β_0 and β_1. It should take the same arguments as `plm()` along with an argument B for the number of bootstrap samples to draw. Let the default be B = 100. `plm.bootstrap()` should return standard errors for both parameters. This function should call your `plm()` function repeatedly. What standard errors do you get for the two parameters using initial conditions $\beta_0 = 6611$ and $\beta_1 = 0.15$? Initial conditions $\beta_0 = 5000$ and $\beta_1 = 0.10$.

```
plm.bootstrap <- function(beta0, beta1, X = gmp$pop, Y = gmp$pcgmp, B = 100){
  # this function compute the bootstrap estimate of standard error of beta0 and beta1
  # former 4 arguments are same as plm()
  # las argument is the number of bootstrap samples with default 100

  bootstrap.sample <- matrix(rep(NA, B*2), ncol = 2)
  for (i in 1:B){
    indices <- sample(1:length(X), replace = TRUE)
    est <- plm(beta0, beta1, X = X[indices], Y = Y[indices])
    bootstrap.sample[i,1] <- est$est.beta0
    bootstrap.sample[i,2] <- est$est.beta1
  }
  beta0.se <- sd(bootstrap.sample[,1])
  beta1.se <- sd(bootstrap.sample[,2])
  est.se   <- list(bootstrap.se.beta0 = beta0.se, bootstrap.se.beta1 = beta1.se)
  return(est.se)
}
plm.bootstrap(6611, 0.15)
```

```
## $bootstrap.se.beta0
## [1] 1.322035e-08
##
## $bootstrap.se.beta1
## [1] 0.0010547
```

```
plm.bootstrap(5000, 0.10)
```

```
## $bootstrap.se.beta0
## [1] 1.5484e-08
##
## $bootstrap.se.beta1
## [1] 0.0009525965
```

vii. The file gmp-2013.txt contains measurements for 2013 (in contrast to measurements from 2006 in gmp.txt). Load it and use `plm()` and `plm.bootstrap()` to estimate the parameters for the model for 2013 and their standard errors. Have the parameters of the model changed significantly?

```
gmp2013 <- read.table("gmp-2013.txt", header = TRUE)
gmp2013$pop <- round(gmp2013$gmp/gmp2013$pcgmp)
plm(beta0 = 6611, beta1 = 0.15, X = gmp2013$pop, Y = gmp2013$pcgmp)
```

```
## $est.beta0
## [1] 6611
##
## $est.beta1
## [1] 0.1433688
##
## $mse
## [1] 135210524
```

```
plm.bootstrap(beta0 = 6611, beta1 = 0.15, X = gmp2013$pop, Y = gmp2013$pcgmp)
```

```
## $bootstrap.se.beta0
## [1] 1.3957e-08
##
## $bootstrap.se.beta1
## [1] 0.001039769
```

```
plm(beta0 = 5000, beta1 = 0.10, X = gmp2013$pop, Y = gmp2013$pcgmp)
```

```
## $est.beta0
## [1] 5000
##
## $est.beta1
## [1] 0.164427
##
## $mse
## [1] 139208731
```

```
plm.bootstrap(beta0 = 5000, beta1 = 0.10, X = gmp2013$pop, Y = gmp2013$pcgmp)
```

```
## $bootstrap.se.beta0
## [1] 6.107205e-08
##
## $bootstrap.se.beta1
## [1] 0.001160188
```

We can see the parameters (estimated value and their standard error) of the model have not changed significantly when using $\beta_0 = 6611, \beta_1 = 0.15$ as the initial guess; when using $\beta_0 = 5000, \beta_1 = 0.10$, the standard error of β_0 changed a lot.