Applying R Fundamentals in Applications

Brian Vegetabile

2016 Statistics Bootcamp
Department of Statistics
University of California, Irvine

September 14th, 2016

Overview

- Seeing the R fundamentals nice, but it is nice to see the fundamentals through the eyes of a few applications
- Today we'll outline a few applications revolving around simulations
  - Regression
  - Simulating the Central Limit Theorem
  - Resampling Methods
- One thing we'll highlight today is that when simulating it is necessary to compare the simulation to some truth

Regression in R - I

- In our linear algebra review we showed that if $\mathbf{X'X}$ was invertible, then

$$\hat{\beta} = (\mathbf{X'X})^{-1}(\mathbf{X'Y})$$

- Let's create a function that calculates this and then simulate some data to test our theory.

Regression in R - II

- First let's write our function, functions have the form

```
myFunction <- function(var1, var2, ...){
        some operations
        .
        .
        .
        return(object)
    }
```

- Therefore we need to name our function, and specify the input variables which it will expect to receive

Regression in R - III

- Let's call our function `linearRegression`, and based upon our theory it should expect some values `X` and `Y` and return our β coefficients

```
linearRegression <- function(X, Y){
        some operations
        .
        .
        .
        return(betas)
    }
```

Regression in R - IV

- Now to create our function, we'll need some matrix algebra functions in R
- Consider the matrices `A` and `B` and a vector `x`, then

Operator or Function	Description
`A * B`	Element wise multiplication
`A %*% B`	Matrix Multiplication
`t(A)`	Transpose of a Matrix
`diag(x)`	Creates a diagonal matrix with elements of `x` in the principal diagonal
`solve(A)`	Inverse of a square matrix `A`

- Let's write our function

Regression in R - V

- We should have all obtained the following

```
linearRegression <- function(X, Y){
    invXTX <- solve(t(X)\%*\%X)
    XTY <- t(X)%*%Y
    return(invXTX%*%XTY)
}
```

- Writing a function is one thing, but now we need to test our function.

Simulating Data for Linear Regression - I

- Let's talk about some functions to simulate data
- R has a lot of built in functions that are great for simulating data or creating random samples

Operator or Function	Description
`rnorm(n, mean = 0, sd = 1)`	Create a random sample of size n from a normal distribution
`rbinom(n, size, prob)`	Create a random sample of size n from a binomial(size, prob)
`runif(n, min = 0, max = 1)`	Create a random sample of size n from a uniform(min, max)
`sample(x, size, replace = FALSE, prob = NULL)`	`sample` takes a sample of the specified size from the elements of x using either with or without replacement.

Simulating Data for Linear Regression - II

- There are many other functions related to simulated data, there are distributions for poisson data, exponential data, gamma data, etc.
- Additionally there are functions for densities, distributions, and quantiles for each of the distributions.

Simulating Data for Linear Regression - III

- Recall the setting of simple linear regression, that is consider $Y_1, Y_2, \ldots, Y_n$ such that

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

  where $\epsilon_i \sim N(0, \sigma^2)$
- Therefore to test our linear regression function we'll need to specify β_0, β_1 and a set of x_i.
- Additionally we will want to add some random normal errors to each observation
  - We'll make use of those statistical distributions we just defined
- Let's start writing some code

Simulating Data for Linear Regression - IV

```
set.seed(2016)
# Setting the number of sim. points
n.samples <- 100

# Setting the beta coefficients
beta0 <- 5
beta1 <- 2

# Randomly sampling 100 x points
# uniformly on a set of values
x <- rnorm(n.samples, mean=2, sd=2)

# Generating Error Terms
error <- rnorm(n.samples)

# Creating our vector of observations
Y <- beta0 + beta1*x + error
```

Investigating Plotting I

- Before we test our function, let's make sure that our simulated data is what we expected
- This is a nice place to introduce a few plotting fundamentals
- Specifically,
    - `hist`, `density`, and `boxplot` - viewing and understanding univariate distributions
    - `plot` - scatterplots for understand how two variables are related.

Investigating Plotting II

```
# Start with the simplest plot
# Boxplot is a visual representation
# of a five number summary
boxplot(x)

# Clean up the plot a bit
boxplot(x, horizontal = T,
        col='blue',
        xlab = 'X',
        main = 'Distribution of X')
```

- The boxplot provides a visual representation of the five number summary which is provided by `summary`.
- Useful for comparing distributions next to each other

Investigating Plotting III

```
# Now we show histograms and their use
hist(x)
hist(y)

# What happens if we change the break points
hist(x, breaks = 5)
hist(x, breaks = 10)
hist(x, breaks = 20)
hist(x, breaks = 100)

# Let's make one nice plot
hist(x, breaks=10, freq = F, col='blue',
     xlab='X', ylab='Density',
     main='Distribution of X')
```

Investigating Plotting IV

- We see that the histogram gives us a lot of information about the univariate distribution
- There is another function which can also provide some information about the distribution called `density`.

Investigating Plotting V

```
# Let's take a look at the density function
x.density <- density(x)

# What happens if we change the bandwidth
plot(density(x))
plot(density(x, bw=0.25))
plot(density(x, bw=0.5))
plot(density(x, bw=1))

# Similar to the hist plot, density can
# be tuned by the user.
```

Investigating Plotting VI

- We see that boxplots, histograms, and density plots provide a nice visual summary of the data
- It fails to communicate how two variables are related to each other.
- This is where the scatter plot comes into play

Investigating Plotting VII

```
# Scatter plots
# Just plotting the X and Y data provides a
# visualization that we will be able to test
# our function!
plot(x,Y)

# That's not a great plot though,
# let us change some parameters
plot(x,Y,
     pch=19, col=rgb(0,0,0,0.25),
     xlab="X", ylab="Y")
```

Investigating Plotting VIII

- Plot takes many arguments beyond `x` and `y`.
- Two of my favorite arguments to change are `pch` and `col`
  - `pch=19` fills in the points so that they are solid, different numbers provide different shapes
  - `col` allows you to change the col of the points
- Related to `col` is the function `col`

```
rgb(red, green, blue, alpha,
    names = NULL, maxColorValue = 1)
```

  - This allows you to set the red, green, and blue levels of the color for plotting. The major advantage of this than specifying a 'named' color is setting the `alpha`level. This controls transparency

Investigating Plotting IX

- Finally we introduce one more function before continuing.
- The function `abline` can be used for plotting reference lines on a plot that has already been draw

```
abine(h=5) #draws a horizontal line at 5
abline(v=0) #draws a vertical line at 0
abline(a=2, b=5)  # draws a line with
                  # intercept a and slope b
```

Investigating Plotting X

The following bit of code allows us to see that the data is representative of true line

```
# abline
plot(x,Y,
     pch=19, col=rgb(0,0,0,0.25),
     xlab="X", ylab="Y")
abline(a = 5, b=2)
```

Checking Our Linear Regression Function I

- Now we are able to test our linear regression function
- We have created fake data which is able to test our function and verified that it is representative of the truth
- Now we need our data in the right format to test the function, recall

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{n1} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

or

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Checking Our Linear Regression Function II

- We see that need to create the matrix $\mathbf{X}$.
- The matrix $\mathbf{X}$ is often called the design matrix.
- To create it we can create a matrix where the first column is a vector of 1's and the second column is our simulated x data.
- Let's code that up real quick and test our code

Checking Our Linear Regression Function III

```
# Testing our function ----------------
# Creating our design matrix
X.Design <- cbind(rep(1, n.samples), x)

# Calculating the coefficients
betas <- linearRegression(X.Design, Y)
print(betas)

# Checking if it agrees with the truth
plot(x,Y,
     pch=19, col=rgb(0,0,0,0.25),
     xlab="X", ylab="Y")
abline(a = 5, b=2)
abline(a=betas[1], b=betas[2],
       col='red', lty=2, lwd=2)
```

Comparing with R's `lm` function I

- One last check we can do is to compare our result with the results that would be obtained from R's built in linear regression function
- Many of you have already played with this function, so I won't review it unless you have questions

Comparing with R's `lm` function II

```
> # Comparing with R’s lm function
> lm(Y ~ x)

Call:
lm(formula = Y ~ x)

Coefficients:
(Intercept)            x
      5.179        1.985

> print(paste(round(betas,3)))
[1] "5.179" "1.985"
```

R and the Central Limit Theorem I

- Let's use R to stress the Central Limit Theorem a bit
- Consider the following random variable

$$Y|X \sim N(\mu + x\delta, \sigma^2)$$
$$X \sim Bernoulli(p)$$

- This says that

$$Y|X = 1 \sim N(\mu + \delta, \sigma^2) \qquad Y|X = 0 \sim N(\mu, \sigma^2)$$

- Let's find a way to visualize this distribution through simulation, which is called a mixture of normal distributions

R and the Central Limit Theorem II

- Let's simulate a large number of values from this distribution and introduce a few functions
- Similar to our last simulation let's set up our parameters first

```
set.seed(62086)
# Let's visualize this distribution
## Parameters for Y
mu = 5
delta = 4
sigma = 1

## Parameters for X
p = 0.3
```

R and the Central Limit Theorem III

- Now we will want to simulate a very large number of observations, say $n = 100000$.
- To create our random variable Y, we first must simulate $X \sim Bernoulli(p)$
- Let's code that

```
# Large number of samples to visualize the distribution
n.samples <- 100000

# Simulating X and finding the number of 1's and 0's
X <- rbinom(n.samples, size = 1, prob = p)
nx <- sum(X)
ny <- n.samples - nx
```

R and the Central Limit Theorem IV

- Now we will simulate the values of Y conditioned on the values of X
- Let's introduce another great function in R called `ifelse()`

```
ifelse(test, yes, no)
```

- `test` - an object which can be coerced to logical mode
- `yes` - return values for true elements of test
- `no` - return values for false elements of test

R and the Central Limit Theorem V

- Based on the values of X we can draw values for Y from the conditional distributions.

```
Y <- ifelse(as.logical(X),
            rnorm(nx, mean = mu+delta, sd = sigma),
            rnorm(ny, mean = mu, sd = sigma))
```

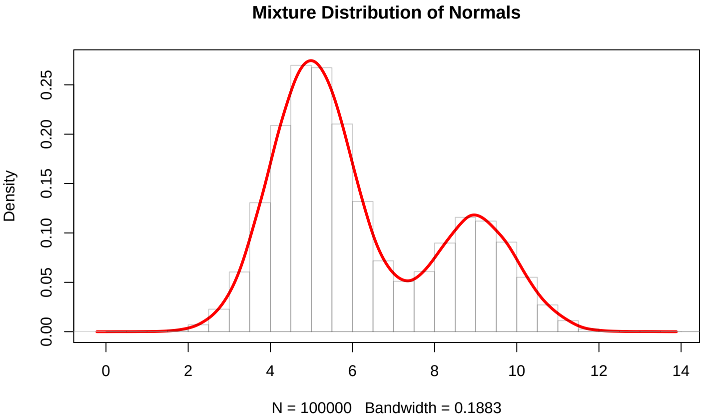
- Notice that we could have populated a matrix `Y` using a `for` loop, but I wanted to highlight this very handy function.

R and the Central Limit Theorem VI

- Let's visual what this distribution looks like

```
# Let's visualize our candidate distribution
plot(density(Y),
     col='red', lwd=3,
     main='Mixture Distribution of Normals')
hist(Y, breaks=seq(0, 15, 0.5),
     freq=F, border = rgb(0,0,0,0.2), add=T)
```

R and the Central Limit Theorem VII



Reviewing the Central Limit Theorem I

- Now we need to come up with a simulation to test the central limit theorem

Theorem (Central Limit Theorem)

Let $X_1, X_2, \ldots$ be a sequence of independent and identically distributed random variables with mean μ and finite variance σ^2. Then

$$\sqrt{n}(\bar{X}_n - \mu) \xrightarrow{\mathcal{L}} N(0, \sigma^2)$$

Reviewing the Central Limit Theorem II

- Therefore we need μ and σ^2 for this weird distribution.
- From Wikipedia, can use the page for mixture distributions, along with a little algebra to show that

$$E(Y) = \mu^* = \mu + p\delta$$
$$Var(Y) = \sigma^{2*} = \sigma^2 + \delta^2 p(1-p)$$

- Therefore by the Central Limit Theorem we have that

$$\sqrt{n}(\bar{Y}_n - \mu^*) \xrightarrow{\mathcal{L}} N(0, \sigma^{2*})$$

- Now we have theoretical results to compare with

Sampling from the distribution I

- Again this will highlight **sampling distributions**, particularly the distribution of the sample mean.
- We can reuse our code from earlier to investigate the true distribution, but now put it in the framework of a simulation

Sampling from the distribution II

```
# Creating a simulationg to test the CLT
n.sims <- 10000
simulated.means <- rep(NA, n.sims)
sample.size <- 20
for(sim in 1:n.sims){
  X <- rbinom(sample.size, size = 1, prob = p)
  nx <- sum(X)
  ny <- sample.size - nx

  Y <- ifelse(as.logical(X),
              rnorm(nx, mean = mu+delta, sd = sigma),
              rnorm(ny, mean = mu, sd = sigma))
  Ybar <- mean(Y)
  simulated.means[sim] <- Ybar
}
```

Sampling from the distribution III

- Let's investigate how the simulation results compare with the theoretical results from the Central Limit Theorem
- We'll introduce another function called `dnorm` which calculates the density of a normal distribution

Sampling from the distribution IV


```
x.vals <- seq(min(simulated.means),
              max(simulated.means), 0.01)
sampmean.dens <- dnorm(x.vals,
                       mean = true.mean,
                       sd = sqrt(true.var))
plot(x.vals, sampmean.dens,
     type='l', xlab='',
     main='Distribution of the Sample Means')
lines(density(simulated.means), col='red', lwd=3)
```

Introducing the Bootstrap

- Finally we bring our attention to the bootstrap for estimating sample variances.
- For Casella & Berger
  - In statistics we learn about characteristics of a population by taking a sample
  - As the sample represents the population, characteristics of the sample should give us insight into characteristics of the population
  - The bootstrap helps us learn about these characteristics by taking *resamples* (samples from the sample)
  - Developed by Efron in the 1970s and it is worth reading his monograph on the topic

Approach

- Consider a random sample $X_1, \ldots, X_n$ and some estimate $\hat{\theta}(x_1, \ldots, x_n) = \hat{\theta}$.
- To create a bootstrap estimate of the variance of $\hat{\theta}$, create a bootstrap sample of size B
  i) From the realized sample $\mathbf{x} = (x_1, \ldots, x_n)$ sample n values *with replacement*, denoted $\mathbf{x}^*$.
  ii) Calculate the statistics $\hat{\theta}_i(\mathbf{x}^*)$ and store
- Calculate the bootstrapped variance

$$Var_B(\hat{\theta}) = \frac{1}{B-1} \sum_{i=1}^{B} (\hat{\theta}_i^* - \bar{\hat{\theta}}^*)^2$$

Bootstrapped Variances I

- Let's go back to our favorite example of the exponential distribution.
- By the central limit theorem we know that

$$\sqrt{n}(\bar{X} - \lambda) \xrightarrow{L} N(0, \lambda^2)$$

- Additionally we had shown that the sample mean $\bar{X} \sim Gamma(n, \lambda/n)$
- Let's create a bootstrap variance estimate of the sample mean's variance and compare that with the central limit theorem and the true result.

Bootstrapped Variances II

- To start, we know that $E(\bar{X}) = \lambda$ additionally, we can show from results of gamma distributions that $Var(\bar{X}) = \lambda^2/n$.
- Now let's find some bootstrap estimates.
- We'll need an original sample from an exponential distribution with λ
- We'll also need to choose the number of bootstrap samples to take

Bootstrapped Variances III


```
sample.size <- 50
lambda <- 0.2  # Note R uses a different parameter
               # parameterization. The mean is 5
B <- 5000
original.sample <- rexp(sample.size,
                        rate = lambda)
```

Bootstrapped Variances IV

- Now let's create a matrix to store each bootstrap estimate.
- The matrix will need to B rows long and some column width.
- Let's make the width 3 and perform a few other estimates along the way to see the power of the bootstrap
  - mean
  - median
  - variance

  we'll also set up the for loop to perform our estimates.

Bootstrapped Variances V


```
bootstrap.ests <- matrix(NA, ncol = 3, nrow=B)
for(b in 1:B){
  bootstrap.sample <- sample(original.sample,
                             size = sample.size,
                             replace = T)
 ...
}
```

Bootstrapped Variances VI

- Now we fill in the estimates that we would like to perform, storing the values in the appropriate locations.
- This is where understanding matrix subsetting comes in handy

Bootstrapped Variances VII

```
sample.size <- 50
lambda <- 0.2  # Note R uses a different parameter
# parameterization. The mean is 5
B <- 5000
original.sample <- rexp(sample.size,
                        rate = lambda)
bootstrap.ests <- matrix(NA, ncol = 3, nrow=B)
for(b in 1:B){
  bootstrap.sample <- sample(original.sample,
                             size = sample.size,
                             replace = T)
  bootstrap.ests[b, 1] <- mean(bootstrap.sample)
  bootstrap.ests[b, 2] <- median(bootstrap.sample)
  bootstrap.ests[b, 3] <- var(bootstrap.sample)
}
```

Bootstrapped Variances VIII

- Now based off these estimates we must compare with the truth
- We introduce another function which is incredibly useful in simulations and that is the `apply` function
- Usage

  ```
  apply(X, MARGIN, FUN, ...)
  ```

- Parameters
  - `X` - an array including a matrix
  - `MARGIN` - For a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns.
  - `FUN` - The function to be applied to that margin. Often, mean median, variance, etc.

Bootstrapped Variances IX

- Let's use this function to summarize some of our bootstrap estimates.
- Note that you could use `colMeans()` if you'd like, but `apply` is more general.

```
bootstrap.means <- apply(bootstrap.ests, 2, mean)
bootstrap.vars <- apply(bootstrap.ests, 2, var)
```

Bootstrapped Variances X

- Comparing to the truth, we see that the bootstrapped variance of the sample mean is very consistent with the mean obtained by the central limit theorem
- Additionally both are consistent with the true values for this
- What is very useful is that we can get a bootstrapped estimate of the variance of the sample median! Or the sample variance!
- We don't need to rely solely on the central limit theorem and the delta method
- If you're interested in this method I suggest reading more on the bootstrap by Efron