ENV 710 – Applied Data Analysis
Fall 2014
Lab 6: Introduction to Resampling

Getting started

This lab explores resampling techniques. In particular, we will use bootstrapping to estimate confidence intervals (CIs). We start with the simple case of bootstrapping for a single variable and two independent variables. Then we examine the slightly more complex case of two related variables. The trick to randomization techniques is figuring out how to resample the data. The basic procedure remains generally the same for the different techniques: (1) randomize the data, (2) calculate the appropriate statistic (CI, *t*, R^2, etc.), (3) repeat the process a bunch of times (usually 1000 or more), (4) interpret the randomized distribution of statistics.

After this lab, you should be able to:

- Run for-loops and resample data
- Calculate bootstrap CIs for a single variable and two independent variables
- Calculate bootstrap CIs for two related variables.

At the end of the lab, there are a few questions to answer. Please type your answers to each problem, including any requested graphs, in a Word document. *Submit your answers and your **R**-code to the class Sakai site under the folder Assignments before 5 pm on either Mon., Oct. 6 (Section 07) or Wed., Oct. 8 (Sections 01 and 02).*

More functions in R

In this lab, we introduce a few new **R** commands.

`boot()` - function from the 'boot' package that generates bootstrap replicates of a statistic applied to data; this function obviates the need of a for-loop
`sample()` - takes a sample of specified size from a vector; the user can specify sampling with or without replacement
`segments()` – draws line segments between pair of points, given by x0, y0, x1, y2
`lm()` - fits linear models, either regression of analysis of variance

To use `boot()`, you will need to install the 'boot' package in your **R** library.

```
set.seed(1001)
 x <- rpois(25, lambda = 6)

  samp1 <- sample(x, 25, replace = T)
   samp2 <- sample(x, 25, replace = T)
    data <- data.frame(cbind(samp1, samp2))

 ratio <- function(data, i){
```

```
                sum(data$samp1[i])/sum(data$samp2[i])}

 (b <- boot(data, ratio, R = 999))
  b$t
   mean(b$t)
```

A few notes about the above code. The `ratio()` function calculates the statistic, the ratio of *samp1/samp2*. The `boot()` function will calculate this statistic for 999 resamples of the original data vectors. The function must include an index, in this case *i*, to allow boot to select a sample. In this code, the results of `boot()` are stored in the variable `b`. The results of `b` show the original ratio (i.e. `sum(samp1)/sum(samp2)`), the bias of the bootstrapped ratio, and the standard error of the bootstrapped distribution. `b$t` is a matrix that includes all the bootstrapped samples of the statistic so that you can use them to calculate other statistics – mean of the bootstrapped statistics, standard error, and confidence intervals. In this case, `mean(b$t)` calculates the mean of the 999 bootstrapped ratios. Note that `mean(b$t)-sum(samp1)/sum(samp2)` returns the bias.

Estimating a population mean

Before we get into the details, let's review sampling as an introduction to bootstrapping. Often when we have a sample from a population, we are interested in estimating the mean of the population itself. The mean of the sample gives us an estimate of the mean of the entire population. For example, we might use a sample of NSOE graduate student IQ's to learn about the mean IQ for the entire school. Our best guess at the mean IQ for the school is the mean IQ in our sample.

The sample mean is a point estimate of the population mean. It gives us a single value as an estimate for the true population mean. But, ideally, we would also like to know how close this estimate is to the actual value.

[1] Take a random sample of size 35 from a normal distribution with mean 20 and standard deviation 4. (Please use `set.seed(1001)`.) What's the true mean of the population? What's the estimate from your sample? Does your estimate exactly match the population mean?

```
set.seed(1001)
mysample <- rnorm(35, mean = 20, sd = 4)
mean(mysample)
```

To figure out how close our estimate is to the true value, we can consider the sampling distribution of the sample mean. The sampling distribution of the sample mean tells us how the sample mean will vary from sample to sample for repeated samples from the population. Statistical theory tells us that the sampling distribution of the sample mean has standard deviation given by:

$$\sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}},$$

where σ is the standard deviation of the population. Most of the time (about 95% of the time), the sample mean will be within approximately 2 of these standard deviations from the true population mean.

[2] What is the standard deviation of the sample mean from the above distribution? Is your sample mean from #1 within 2 of these standard deviations from the true mean?

Let's investigate the sampling distribution of the sample mean further by taking 1000 samples from this normal distribution. We will make a histogram of the means of these 1000 samples and verify that about 95% of the time the sample mean will be within $\pm 2\sigma_{\bar{X}}$ of the true sample mean.

```
set.seed(1001)
sample.means <- c()
n <- 25

for (i in 1:1000){sample.means[i] <- mean(rnorm(n, 20, 4))}

hist(sample.means, las = 1, col = "darkblue", main="")
segments(sort(sample.means)[25], 0, sort(sample.means)[25],
         250, col = "red", lwd = 2)
segments(sort(sample.means)[975], 0, sort(sample.means)[975],
         250, col = "red", lwd = 2)

sum(sample.means >= 20 - 2*4/sqrt(n) &
      sample.means <= 20 + 2*4/sqrt(n))/1000
```

[3]: Modify the above code to graph a histogram that shows the number of sample means $\pm 1\sigma_{\bar{X}}$ of the true sample mean. How many of the means are within this interval?

Generally we don't know the true population mean and standard deviation, which is why we estimate them from the sample. So rather than calculating $\pm 2\sigma_{\bar{X}}$, instead we estimate σ with *s*, the sample standard deviation. This gives us the standard error of X, defined as:

$$SE_{\bar{X}} = \frac{s}{\sqrt{n}}$$

We can form a 95% confidence interval for this example (*n* = 25) by taking $\bar{X} \pm t_{1-(\alpha/2)} SE_{\bar{X}}$. We are 95% confident that the true population mean lies in this interval. We can find the critical *t*-value using:

```
qt(1-0.05/2, df = n-1)
```

If the sample size were larger we would use a quantile from the normal distribution. Alternatively, using the t-distribution would produce very similar results, as a t-distribution with many degrees-of-freedom approximates a normal distribution.

[4] Using `mysample` for #1, calculate a 95% confidence interval for the true population mean. Does this interval include the true population mean of 20?

The procedure outlined above for forming a confidence interval for the population mean using the sample mean is only valid if the population is normally distributed or if the sample size is large enough (n ≥ 30 is a rule of thumb). We may run into cases that violate these assumptions, or we may want to form confidence intervals for quantities besides the mean. The bootstrap is a method that can be used to form confidence intervals in a more general setting.

The bootstrap CI

The bootstrap estimates standard error by resampling the data in our original sample. The idea is to treat the sample as a substitute for the population. Instead of repeatedly drawing samples of size *N* from the population, we will repeatedly draw new samples of size *N* from our original sample. To avoid ending up with exactly the same sample every time, we resample *with replacement*. This means that after a value is drawn, we replace it before drawing again. This way we can draw the same value out more than once! Let's try it.

Let's take a sample of 10 with replacement from your original sample. This is called a bootstrap sample. Are there any duplicates in your bootstrap sample?

```
set.seed(1001)
boot.sample <- sample(mysample, 10, replace = T)
```

To do a full bootstrap, we take a large number of bootstrap samples (at least 1000) from the original sample. For each bootstrap sample, we calculate the mean of the bootstrap samples. The variation in the means of the bootstrap samples gives us an estimate of the variability in the sample mean. That is, we can *estimate the standard error of the sample mean using the standard deviation of the bootstrapped sample means*. We can then use this to construct a confidence interval for the true population mean.

Let's try this out by taking 1000 bootstrap samples from your original sample and calculating the mean of each of these bootstrap samples. Then we will make a histogram of these bootstrapped means.

```
set.seed(1001)
boot.mean <- c()
n <- 20

for(i in 1:1000){

      boot.sample <- sample(mysample, n, replace = T)
      boot.mean[i] <- mean(boot.sample)

}

hist(boot.mean, las = 1, col = "darkblue", main = "")
```

We can then calculate the standard error for the bootstrap distribution and construct a 95% confidence interval. How does this compare to the interval you constructed in #4?

```
se <- sd(boot.mean)
```

```
 tcrit <- qt(0.975, df=n-1)
 ci <- c(mean(boot.mean)-tcrit*se, mean(boot.mean)+tcrit*se)
```

Hopefully you have been convinced that bootstrap sampling is a good substitute for the true sampling distribution.

The percentile CI (also see the Addendum) is calculated as the $(\alpha/2, 1-\alpha/2)$ percentiles of the bootstrapped data. Here, α is 0.05, so we need to take the 2.5% and 97.5% percentiles (of 1000 bootstrap samples) or the 25th and 975th values.

```
c(quantile(boot.mean, 0.025), quantile(boot.mean, 0.975))
```

The bootstrap with two related variables

Now we will consider the situation where we have data on two variables, but the variables are measured on the same individual. This is the type of data that arises in a linear regression situation. We measure two variables on a given individual, so that these measurements are naturally paired together. It doesn't make sense to bootstrap the two variables separately, because they are linked, and so must remain linked when bootstrapped. Instead, we use bivariate bootstrap sampling of the (x, y) pairs of data.

For example, if our original data contains the observations (1,3), (2,6), (4,3), and (6, 2), we re-sample this original sample in pairs. The important thing is that the x and y coordinates must remain linked while being re-sampled.

We are going to work with the Galapagos data from Gotelli and Ellison, Chapter 8. The data represents the species-area relationship. Note that the species and area are linked – we cannot separate one from the other – and therefore we need to bootstrap them as pairs. In principle, the species-area relationship is a general model that predicts the number of species on an island based on its area. Therefore, the Galapagos data represents a sample of all the islands in the world and the number of species that would be found on them. We want to find the variation around our sample statistics. In this case, we want to know the possible range of values for the intercept and slope. A confidence interval contains plausible values for our statistics.

Load the data.

```
dat<-read.csv("Galapagos.csv", header=T)
attach(dat)
```

Plot the raw data. We know that the relationship between species and area is best described by a power function. Therefore, it is not surprising that the data are not linear and would be poorly fit by a linear regression.

```
plot(Area, Nspecies, ylim=c(0, 400), las=1, pch=19, bty="l",
     xlab="Island area (km)", ylab="Number of species")

fit <- lm(Nspecies ~ Area)
 abline(fit, col = "darkblue", lwd = 1.5)
```

Now we plot the log10-transformed data and add the best-fitting regression line.

```
plot(log10(Nspecies) ~ log10(Area), las=1, pch=19, bty="l")
 Lfit<-lm(log10(Nspecies) ~ log10(Area))
  abline(Lfit, col="purple", lwd=1.5)
```

You can get the regression coefficients (slope and intercept) from the linear regression model by:

```
Lfit$coefficients
```

In our example, `(Intercept)` is the y-intercept of the regression line and `log10(Area)` is the slope.

Now we take 1000 bootstraps to estimate the 95% CI's on both the log10 slope and log10 intercept parameters of the line. To do this, we sample from the data, randomly choosing the lines of the data frame to keep (with replacement). Then we transform the data and conduct the linear regression, and store the coefficients. From the distribution of coefficients, we can then calculate the 95% percentiles that represent our CI's for each parameter (in this case, on the log-scale).

```
runs <- 1000
len <- length(Nspecies)
slope <- c()
intercept <- c()

for(i in 1:runs){

   samp <- dat[sample(nrow(dat), size=len, replace=T), ]
    fit <- lm(log10(samp$Nspecies) ~ log10(samp$Area))
     intercept[i] <- fit$coefficients[1]
      slope[i] <- fit$coefficients[2]
}

    mean.int <- mean(intercept)
    mean.slope <- mean(slope)
```

Now we make histograms of the bootstrapped parameter distributions for intercept and slope and visualize the 95% CIs, using percentile CIs.

```
par(mfrow=c(1,2))

 hist(intercept, las=1)
  abline(v=sort(intercept)[25], col="red", lwd=1.5)
   abline(v=sort(intercept)[975], col="red", lwd=1.5)
    ci.int<-c(quantile(intercept, 0.025),
              quantile(intercept, 0.975))

 hist(slope, las=1)
 abline(v=sort(slope)[25], col="red", lwd=1.5)
  abline(v=sort(slope)[975], col="red", lwd=1.5)
   ci.slope<-c(quantile(slope, 0.025),
               quantile(slope, 0.975))
```

Run the above script to find 95% bootstrap interval for each of the coefficients and to create histograms of the bootstrapped distributions.

[5]: What are the mean estimates for the intercept and slope of the regression on the original scale of the data? Report and interpret 95% bootstrap intervals for the two coefficients. In regression, we often want to show that the coefficients are not equal to zero. Based on the 95% bootstrap intervals, are the values of the coefficients different from zero?

Problem #1: Responses from above

Provide answers to the questions [1-5] in the above text of the lab.

Problem #2: A real example – insect relative abundance

Suppose we were interested in estimating the relative abundance of insect species in the Cape Fear River. We collect data on the abundances of 12 different species:

```
insects <- c(0.14, 15.49, 29.04, 6.36, 1.83, 5.40, 31.89,
             3.92, 0.54, 2.01, 12.67, 48.75)
hist(insects, las = 1, ylab = "No. species",
     xlab = "Abundance", main="", col="lightgrey")
```

Estimate the mean abundance of insect species using the bootstrap procedure, given that small sample size and lack of a normal distribution prohibits us from using parametric methods. Please do all the following:

1. Bootstrap the data and calculate the mean of the bootstrapped means. How does this compare to the sample mean?
2. Estimate the standard error of the bootstrapped mean. How does this compare to the standard error calculated by the typical parametric formula?
3. Calculate the 90% confidence interval for the original sample. Then calculate the 90% confidence interval for the bootstrapped mean of the relative abundance of insect species, using the (a) normal parametric formula, (b) as a percentile CI.

Addendum

There are multiple ways to calculate bootstrap CIs. For example, in the `boot` package, you can select between normal, percentile, basic, studentized, and BCa CIs. The below code shows a simple example for using `boot.ci()` to get any of these CI's. Then there is code for calculating each one of these separately on the insect data.

To use the `boot()` function to generate bootstrap replicates of a statistic requires that you create a function that calculates the statistic. Importantly, within the function you must include an index. Here I show an example for calculating bootstrap CIs on the sample mean. The `boot.ci()` command returns five types of CIs.

```
mn <- function(d, i){
       avg <- mean(d[i])
        return(avg)
}

set.seed(1001)
results <- boot(data=insects, mn, R = 1000)
boot.ci(results, type=c("norm"))
boot.ci(results, type=c("basic"))
boot.ci(results, type=c("perc"))
boot.ci(results, type=c("bca"))
```