STAT 113: Lab 4

Sampling Distributions and Confidence Intervals

Last Revised September 29, 2017

Overview and Reference

The goal of this lab is to explore a simulation-based method of computing standard errors of a statistic like a sample mean, which we use to create confidence intervals about a parameter, like a population mean.

What to Turn In

You should turn in your solutions to the four Homework problems at the end of the packet as an RMarkdown document. Upload your `.Rmd` source and your `.pdf` output to Blackboard (you may either Knit directly to `.pdf`, or Knit to `.html` and then "print" that to a `.pdf` when you're finished).

If you need to refer to the Markdown workflow, you can look at the lab handout from last week or the week before. The Markdown workflow outline from last week's lab is reproduced below for your convenience.

R Markdown Workflow

Once you have done this a few times you will develop your own habits, but these are good habits to develop the first few times you use Markdown.

1. From the RStudio File menu, select New File > R Markdown.... Now you have several options. From the list on the lefthand side, select From Template, and then on the right, pick mosaic fancy to get a template that includes a bunch of examples showing you how to include various kinds of content. As you create your own documents you will want to delete much of the extra stuff in the middle (starting with the line `## Using RMarkdown` and ending before `* File creation date`), but reading this template is a good way to learn about how Markdown works.
2. The first thing you should do after creating a new Markdown document is to save the file. Do File > Save As... and give your document a name.
3. Before editing anything, press the Knit HTML button in the document toolbar (or type Ctrl-Shift-k / Cmd-Shift-k). If you are on the server or if you happen to have LaTeX installed on your computer, you could choose Knit PDF instead. If on the server you may get a message about a blocked pop-up. Allow it to show, and you will get a new window with a mix of code, text, and plots. If you prefer, you can change the setting that puts the output in a new window and have it show up in the lower right pane of RStudio instead. You will find this option under Tools > Global Options... > R Markdown in a dropdown menu that says "Show output preview in..."
4. The first thing you should edit is the Title and Author fields at the top of the document. Give the document a title such as "STAT 113: Lab 3", and put your name inside the quotes on the Author line. Re-Knit to see your change reflected in the document.
5. When you make a change, you can test that your code works as expected by first sending that chunk's code to the console by clicking the little green triangle (looks like a "play" button) in the upper right corner of the chunk. This will execute all the code in that chunk in the console. If this chunk depends on earlier ones, you may first need to click the button to the left that send all code in previous chunks to the console. If it works as expected, "Knit" again to update your output.

Notice that there is also a reference guide with lots more info on customizing formats, etc., under Help Markdown Quick Reference that will pop up in the Help tab.

Note You should *not* include code in your Markdown document that you would not want to be run over and over again, or that produces output that you would not put in a writeup. This includes `install.package()` lines, lines that access documentation (e.g., `?EmployedACS`), or generate dialogue boxes.

Review of Some Important Definitions

Some of the terms in the overview paragraph were only recently defined in class. As a reminder, here are their definitions.

Definitions A **population** consists of all the cases of potential interest. Populations have **parameters** (like a mean, median, standard deviation, etc.) that summarize some property of the population

Definitions A **sample** consists of a subset of cases from the population; ideally a representative (perhaps random) subset. Summary values like the mean, median, standard deviation, etc. are called **statistics** when they are calculated on samples. We typically use statistics as estimates of their corresponding parameter.

Definition To understand how good our estimates are, we want to investigate the **sampling distribution** of our statistic, which consists of values of that statistic each calculated using a different random sample drawn from the population.

Definition The **standard error** of a statistic/parameter pair (e.g., the mean) is the standard deviation calculated on the statistics in the sampling distribution.

1 Computing the Standard Error by Repeated Sampling

We consider real estate data from the city of Ames, Iowa. The details of every real estate transaction in Ames is recorded by the City Assessor's office. Our particular focus for this lab will be all residential home sales in Ames between 2006 and 2010. This collection represents our population of interest. In this lab we would like to learn about these home sales by taking smaller samples from the full population. Let's load the `mosaic` package and the data.

```
library("mosaic")
Ames <- read.file("http://colindawson.net/data/ames.csv")
```

In this lab we have access to the entire population, but this is rarely the case in real life. Gathering information on an entire population is often extremely costly or impossible. Because of this, we often take a sample of the population and use that to understand the properties of the population.

Suppose we were interested in estimating the mean living area in Ames. We might survey a random sample of, say, 50 homes in the city and collect various information about each home. The full dataset contains 2930 homes, so we will have data on less than 3% of the population, but it will turn out that we can make some decent estimates about the population, provided our sample is random.

We will focus on the variables `Price`, which records the sale price in dollars of each home in the dataset, and `Area`, which contains the total above-ground living area in square feet.

Let's take a sample of size 50 from the population and compute the mean Area in the sample. The `sample()` command takes a simple random sample of a specified size from a data frame. The result is a data frame consisting of the sampled cases. We will store the result in a named variable called `Sample50` (we could pick any name we want).

An R tip when doing random simulations In this lab we will be doing some stochastic (random) simulations, like sampling many times from a population. This will involve commands that take random samples from larger datasets. Since the sample is random, it will change every time you re-run the command. This can be annoying if you are trying to describe the results in text. To avoid this issue, you can include the following line of code at the start of your script or Markdown document to initialize the random number generator the same way every time you re-run or Knit your document (as long as the code between this line and the random line stays the same):

```
## You can pick any number here; I've used my T number
## This needs to go before any line that samples.
set.seed(00029747)
```

```
Sample50 <- sample(Ames, size = 50)
```

Exercise 1 Describe the distribution of `Area`s in your sample. What would you say is the "typical" size within your sample? Also state precisely what you interpreted "typical" to mean.

Exercise 2 Would you expect another student's distribution to be identical to yours? Would you expect it to be similar? Why or why not?

You may have chosen to use the mean or the median as a "typical" value in the distribution of areas. So that we're all on the same page, let's focus on the mean.

```
sample.mean <- mean(~Area, data = Sample50)
```

Depending on which 50 homes your sample happens to contain, your estimate could be a bit above or a bit below the true population mean. In general, though, the sample mean turns out to be a pretty good estimate of the average living area, and we were able to get it by sampling less than 3% of the population.

In this lab, because we have access to the population, we can build up the sampling distribution for the sample mean directly by repeating the above steps many times.

Computing a sample mean is easy: just get the sample, and compute the mean, as you did above.

Note that we could take the sample and compute the mean in one step, as we have done in previous labs with `filter()`:

```
## These are like methods 1 and 3 described for in Lab 2 for
## nesting and chaining commands:
mean(~Area, data = sample(Ames, size = 50)) # Method (1)
sample(Ames, size = 50) %>% mean(~Area, data = .) # Method (3)

## Since each time we run the sample() command we get
## a new sample, the resulting mean will change each time too (unless
## we were to reset the random number generator each time using set.seed())
## In contrast, when we took a sample and stored it in Sample50,
## we "froze" the sample to use repeatedly.  When we want to simulate
## repeated sampling, we do not want to use the same sample every time.
```

With the speed of modern computers, it is easy to simulate sampling many, many times from a population. For example, we can simulate drawing 5000 samples each of size 50 from the population and looking at how the sample mean varies across these 5000 samples:

The following code will take 5000 samples each of size 50 from the population, and compute the mean `Area` for each sample, storing the resulting means in a new data frame.

```
Sampling.dist <- do(5000) * sample(Ames, size = 50) %>% mean(~Area, data = .)
```

R and `mosaic` note: The `do()` function in `mosaic` allows you to repeat some code a specified number of times, and store the result of each iteration in a variable. Note that each time we iterate, we get a different random sample. The resulting object, `Sampling.dist`, is a data frame where each case represents a sample of size 50, and the variable recorded is the mean of the `Area` variable for that sample. Use `head()` to examine the first few means (and take note of the variable name: depending on the way you combine your commands (e.g., nesting vs. chaining), the name might differ; but in this case and for the most recent version it should be called `result`).

Let's plot the resulting distribution of means with a dot plot

```
## pay attention to upper case vs. lowercase
dotPlot(~result, data = Sampling.dist, nint = 100)
```

Markdown tip to reduce Knitting time When doing computationally intensive simulation like resampling, it takes a little while to run the sampling code. To avoid having to do this every single time you Knit your document, you can add the chunk option `cache = TRUE` to the code chunk that does the resampling (this works like `include = FALSE` and the other chunk options we have seen, and goes in the same place between the curly braces at the top of the code chunk). This will store the results of the chunk in a "cache" that can simply be read in the next time you Knit. Note though that if any code in the chunk changes, it will be re-run. So it is a good idea to put code that does resampling in a chunk by itself and cache only that chunk, so you are less likely to need to modify it.

We can compute the standard error of the mean by finding the standard deviation of the simulated sample means.

```
## the first line calculates and stores the standard error
true.standard.error <- sd(~result, data = Sampling.dist)
## the second line displays the value
true.standard.error
```

2 Bootstrap Resampling

In reality we do not have access to the population, so we cannot get the true standard error by repeatedly sampling: somehow we have to do everything we want to do using a single sample.

We could rely on probability theory to get an analytic estimate of the standard error from the sample. We will see later that in the case of a sample mean, the standard error is related to the population standard deviation as follows:

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

We can estimate σ_X, the population standard deviation, with s_X, the sample standard deviation, and use this to get an estimated standard error.

However, this approach is specific to the mean, and only works if the population distribution is approximately Normal or the sample is large.

A more general approach is to use the sample as an estimate of the entire population distribution (think of the sample histogram as an estimate of the population histogram) and then simulate repeated sampling from this estimated population. Now not only are we using the sample standard deviation, but we are also using information about the shape of the distribution (skew, "kurtosis", which describes what proportion of the data lies in the "tails") to inform our estimate of the standard error.

To simulate repeatedly sampling from a population with the same histogram shape as the sample but many more cases, we want to act as though there are many copies of each sample case. We can achieve this effect by drawing samples *with replacement* from the sample itself. That is, each time we draw a case for our simulated sample, we keep that case in the pool so that it can be drawn again. This is as if there are many many copies of each case so that we never "use up" a value by drawing it.

To do sampling with replacement in R we can use the `resample()` function. Let's simulate drawing 5000 samples of size 50 with replacement, using our original sample of size 50 as the never-depleting data pool, and computing means for each "resample". This is called **bootstrap resampling** (based on the metaphor of "pulling ourselves up by our bootstraps", since we are seemingly paradoxically creating more data out of nowhere by resampling).

```
### Use cache = TRUE for this chunk
Bootstrap.means <- do(5000) *
    resample(Sample50, size = 50) %>% mean(~Area, data = .)
```

Note that we set the size of the bootstrap samples to be the same as the size of our original sample. This is so that we can examine how far away, say, the sample mean and the population mean are when the sample is *of size 50*. If the sample were larger or smaller, we might have a different degree of confidence in our estimate.

3 Confidence Intervals

If we knew the true sampling distribution of the sample statistic (e.g., the sample mean), then the standard error would be the standard deviation of that distribution. Recall the fact stated in class that in a symmetric, bell-shaped distribution, about 95% cases are within 2 standard errors of the mean.

Therefore, since when the samples are random the mean of a sampling distribution of sample means is the population mean, a sample mean and the population mean it is estimating are 2SEs apart or less about 95% of the time.

Definition The 95% **Margin of Error** associated with an estimate is the distance m such that the estimate (the statistic) is within m of what it is estimating (the parameter) for 95% of all possible random samples. When the sampling distribution is symmetric and bell-shaped, m is about $2SE$.

When we only have one sample, we can estimate the standard error using the standard deviation of the distribution of bootstrap sample statistics (means, in this case).

```
estimated.standard.error <- sd(~result, data = Bootstrap.means)
```

Since the sample mean is within 2 SE of the population mean 95% of the time, we estimate with "95% confidence" that the population mean is within 2 SE of the sample mean. The interval this defines to estimate the population mean is called a **95% Confidence Interval** (abbreviated "CI").

```
CI.lower <- sample.mean - 2 * estimated.standard.error
CI.upper <- sample.mean + 2 * estimated.standard.error
c(CI.lower, CI.upper)  # the c() function combines a list of values
```

Exercise 4 Write a sentence about the confidence interval you get, describing how to interpret it. What does it mean that the interval has a 95% confidence level?

Exercise 5 Determine whether, in this case, the true population mean does in fact fall in your interval. Everyone in the class will have a different intervals, since you each generated different samples. Draw your interval on the whiteboard so everyone can see how they vary. How many captured the true population mean? If you were to repeat this exercise many many times, what fraction of the time do you expect your interval to "miss" the true value? (Or, equivalently if many many students did the exercise, what proportion of them would "miss"?)

Notice that we are either underestimating or overestimating the true standard error by some amount, and we don't know which or by how much, so our margin of error could be too large or too small. But that's okay; we only require that our confidence interval capture the population parameter 95% of the time. As long as things average out, our interval-generating procedure will be valid.

Let's simulate drawing many samples and generating a confidence interval for each one. We could use bootstrap intervals, but this would take a long time, so we'll fall back on the analytic procedure of estimating the standard error as

$$\hat{SE}_{\bar{X}} = \frac{s_X}{\sqrt{n}}$$

```
## use cache = TRUE for this chunk if using Markdown
Sample.stats <- do(100) *
    sample(Ames, size = 50) %>% favstats(~Area, data = .)
```

Look at the data frame created — there should be one column for each of the sample statistics computed by `favstats()`, and one row for each of the 100 samples drawn.

Let's add two new statistics: the lower and upper boundaries of a 95% confidence interval.

```
Sample.stats <-
    mutate(Sample.stats,
           CI.lower = mean - 2 * sd / sqrt(n),
           CI.upper = mean + 2 * sd / sqrt(n))
```

R Note: The `mutate()` function is like `filter()` in that it operates on a data frame and returns a new one. Whereas `filter()` selects cases according to some condition, `mutate()` adds new columns, perhaps defined in terms of others. Notice that this time we are overwriting the original data frame by using the same name again.

Type the following to execute an R script from my website that defines a useful plotting function, `plot_ci()`, that will show each in a set of confidence intervals along with the true population mean, highlighting those that miss.

```
### The source() function executes everything in a given R script file
source("http://colindawson.net/misc/plot_ci.R")
```

Call the function on your collection of sample statistics, specifying the true population mean in the `mu=` argument. Note that this function assumes that you have defined the new variables `CI.lower` and `CI.upper`, by those exact names, in your collection of sample statistics.

```
plot_ci(Sample.stats, mu = mean(~Area, data = Ames))
```

Exercise 6 How many of the 100 intervals in your simulation missed? Is this what you expected?

Homework

1. Create two new sampling distributions as above, one using samples of size 30 and one using samples of size 120. Plot the distribution of sample means for each one and compute the mean of means and the standard error. What do the three distributions have in common? How do they differ? Try to explain why this pattern makes sense.
2. Plot the means from the bootstrap sample. Where are they centered? Is this different than where the true sampling distribution is centered? How does the variability compare?
3. Construct a bootstrap 95% confidence interval for the mean sale price in Ames using a sample of size 50. Construct two more intervals based on samples of size 25 and size 100, respectively. How do the intervals differ? Does the result make sense? Explain.
4. Construct a 95% confidence interval for the correlation between `Area` and `Price` the same way (i.e., do everything the same way except where you computed means before, compute correlations instead). Do you notice anything different about the bootstrap distribution of correlations, compared to the distribution of means? (Hint: Plot it!)