Confidence Intervals for Price of Hostess Twinkies

Goal

The goal of this project is to learn how to compute an empirical 95% confidence interval for the sample mean price for Hostess Twinkies using an awesome technique called *bootstrapping*. Bootstrapping allows us to assess the accuracy of a sample mean we've computed, giving us data about how sample means vary. As part of this lab, you will also learn to read in a file full of numbers. In this case, we are going to read in the price of Hostess Twinkies, a tasty snack recently returned from the dead, from around the US.

Please do your work in filename *userid*-conf/`bootstrap.py`.

Discussion

Given a list of prices for Twinkies, we can easily compute the mean (average). More specifically, we'd be computing the *sample mean*, which is not usually exactly the same as the *population mean*. Getting the population mean is either impractical (pollsters cannot poll every single voter) or impossible and so we must be satisfied with the sample mean. Unfortunately, the sample mean varies from sample to sample so it's more informative to use an interval statistic rather than a point statistic. The interval describes the uncertainty of the sample mean as an estimate of the true population mean. A sample mean confidence interval of 95% tells us the range in which most of the sample means should fall. Therefore, a confidence interval gives us an estimated range that is likely to include the population mean.

Given just one set of numbers, such as our Twinkie prices, (just one mean) how do we get an interval? Using the central limit theorem, which tells us how sample means, m, bounce around the true population mean, μ: sample mean m is drawn from $N(\mu, \sigma^2/n)$ for sample size n. Because we don't know the population statistics, we use the sample mean, $\overline{X}$, and sample variance, s^2, as population estimates. Then, we look at the range of 95% of the mass under the sample mean distribution, $N(\mu, \sigma^2/n)$. The range is $1.96s$ to the left and right of the sample mean m:
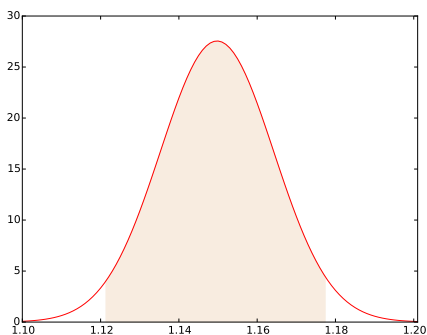
But, is that the best we can do? Here's the problem: For each sample we collect, we will get a differ-

ent sample mean and variance which means that our confidence interval is also a random variable that bounces around. The distribution would appear to shift around if we were plotting the sample mean distributions as the samples were collected. The edges of our interval would be look fuzzy. Dang!

We've seen how increasing the number of samples (not sample size) increases precision or resolution when visualizing distributions for the central limit theorem. We can't increase the sample size to get a narrower interval, but we *can* improve the precision of our interval estimate by acquiring more samples from whatever the Twinkies distribution happens to be.

How do we get lots of samples from an underlying distribution that we cannot identify? Through the magic of *bootstrapping* where we repeatedly *resample* from our single sample *with replacement*. To get a new sample, we randomly select n values from our known data set of size n, leaving each value available to be selected again. That gives us one trial and we compute our test statistic, the mean, on each sample X. For M such trials, we get M sample means, $\bar{X}$. To get the 95% interval for the mean, we sort the means and strip away the bottom 2.5% and the top 2.5% of the means. For example, if M is 200, we drop the smallest 5 (200×0.025) and largest 5 means from the list of 200 means. Then, the lowest and highest value in that truncated list represent the boundaries of the confidence interval. Cool, right?

> The histogram of the sample means from all trials would approximate a normal distribution, but the amazing thing is bootstrapping works even when the central limit theorem does not apply! I.e., when we don't know what the point statistic distribution is. That is the power of simulation.

To verify that we are doing the right thing, we will draw the theoretical normal distribution expected by the central limit theorem and then shade in the 95% theoretical confidence interval, which we know is 1.96 standard deviations on either side of the mean: $\mu \pm 1.96\sigma$. NOTE: Because we have to use the sample estimates for μ and σ, we will be more sure about our bootstrapped estimates than this visualization. The visualization will be just for sanity checking.

Steps

1. First, we have to get the data from a file called `prices.txt` from
https://github.com/parrt/msan501/tree/master/data:

```
prices = []
f = open("prices.txt")
for line in f:
    v = float(line.strip())
    prices.append(v)
```

When debugging or during development, you can print those numbers out to verify they look okay.

2. Now, we need a function that lets us sample *with replacement* from that raw data set. In other words, we need a function that gets n values at random from a data parameter (a list of numbers). It should allow repeated grabbing of the same value (that's what we call "with replacement").

```python
def sample(data):
    """

    Return a random sample of data values with replacement.
    The returned array has same length as data.
    """
```

The idea is to get an array of random numbers from $U(0, n-1)$ for n=len(data). You can use Python's built-in random number generator (import random) or your own from our previous lecture but make sure to add it to your repository. These random values are a set of indices into the data array so just loop through this index array grabbing values from data according to the index. For example if you have indexes = [3,9] for a 2-element data array, then return a new array [data[3], data[9]]. My solution for this function is 2 lines.

3. Now define TRIALS=40 and perform that many samplings of prices. For each sample, create the sample mean and add it to a list variable called X_.

4. Sort that list and get the values at indices TRIALS*0.025 and int(TRIALS*0.975)-1 in X_.

5. Print these left and right values as that will tell you the bounds of your 95% confidence interval. You might get something like (there will be a lot of variation) 1.13057101449 and 1.1880057971.

6. Add matplotlib boilerplate code to create a figure and to plot diamonds on the graph at those locations. I use ax variable below; here's how I get it:

```python
...
fig = plt.figure()
ax = fig.add_subplot(111)
...
```

7. Now plot the normal curve using your amazing understanding of the central limit theorem. Use the following range and also set the overall graph range:

```python
import scipy.stats as stats # you might have to install scipy with pip

x = np.arange(1.05, 1.25, 0.001)
plt.axis([1.10, 1.201, 0, 30])
x = np.arange(1.05, 1.25, 0.001)
y = stats.norm.pdf(x,mean,stddev)  # WHAT ARE MEAN AND STDDEV?
plt.plot(x,y, color='red')
```
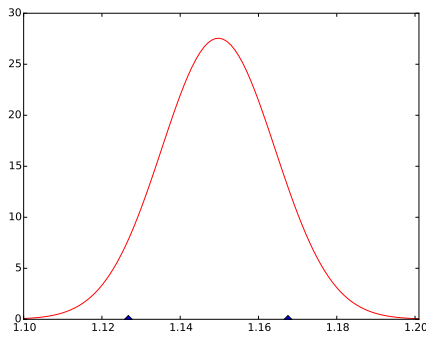
BTW, if you want, you can use your own function called normpdf() that implements the symbolic definition of the normal density function:

$$normpdf(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$$

```python
def normpdf(x, mu, sigma):
    return ...
```

**8.** Run it and you should get the following graph:

Ok, that's great but we have no idea if this is correct or not. Now, let's go nuts and show lots of stuff on the graph.

9. First, let's shade in the theoretical 95% confidence interval.

```
mean = ...
stddev = ...
# redraw normal but only shade in 95% CI
clt_left  = ...
clt_right = ...

ci_x = np.arange(clt_left, clt_right, 0.001)
ci_y = normpdf(ci_x,mean,stddev) # or use stats.norm.pdf()
# shade under (ci_x,ci_y) curve
plt.fill_between(ci_x,ci_y,color="#F8ECE0")
```

Run it again to see how it shades in the graph.
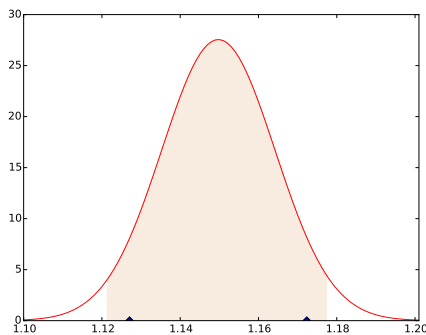
10. Now let's annotate with lots of information. Please read through and figure out what all of that stuff does to draw the nice arrows and so on.

```
plt.text(.02,.95, '$TRIALS = %d$' % TRIALS, transform = ax.transAxes)
plt.text(.02,.9,  '$mean(prices)$ = %f' % np.mean(prices), transform = ax.transAxes)
plt.text(.02,.85, '$mean(\\overline{X})$ = %f' % np.mean(X_), transform = ax.transAxes)
plt.text(.02,.80, '$stddev(\\overline{X})$ = %f' %
    np.std(X_,ddof=1), transform = ax.transAxes)
```

```python
plt.text(.02,.75, '95%% CI = ($%1.2f,\\ %1.2f$)' % (left, right), transform = ax.transAxes)

plt.text(1.135, 11.5, "Expected", fontsize=16)
plt.text(1.135, 10, "95% CI $\\mu \\pm 1.96\\sigma$", fontsize=16)
plt.title("95% Confidence Intervals: $\\mu \\pm 1.96\\sigma$", fontsize=16)

ax.annotate("Empirical 95% CI",
            xy=(clt_left, 0),
            xycoords="data",
            xytext=(1.13,4), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
            fontsize=16)
```

11. Run it and you should get the following graph:
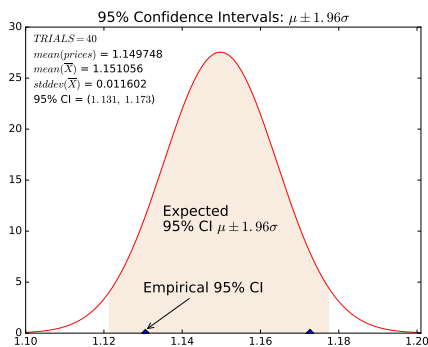
12. We need to make the number of trials a parameter and let's do the annotations as well.
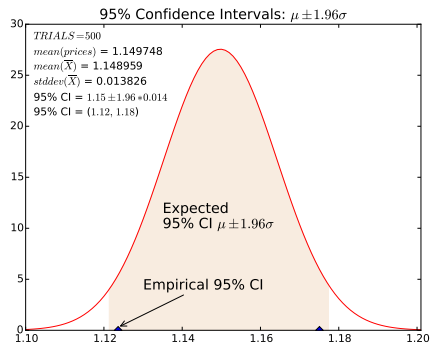
```python
import sys

fancy=False
TRIALS = 40
if len(sys.argv)>1:
    TRIALS = int(sys.argv[1])
    if len(sys.argv)>2 and sys.argv[2]=='-fancy':
        fancy=True
```

Then wrap your annotation code in `if fancy` and add code to save your graph (in all cases):

```python
plt.savefig('bootstrap-'+str(TRIALS)+('-basic' if not fancy else '')+'.pdf', format="pdf")
```

13. We don't have to increase the number of trials very much before the confidence interval tightens up nicely. 500 trials should look something like:

```
$ python bootstrap.py 500 -fancy
```

95% Confidence Intervals: $\mu \pm 1.96\sigma$

$TRIALS = 500$
$mean(prices) = 1.149748$
$mean(\overline{X}) = 1.148959$
$stddev(\overline{X}) = 0.013826$
95% CI $= 1.15 \pm 1.96 * 0.014$
95% CI $= (1.12, 1.18)$

Expected
95% CI $\mu \pm 1.96\sigma$

Empirical 95% CI

If you run it multiple times, the edges of the interval should not bounce around very much where as they do with only 40 trials.

Deliverables. *userid*-conf/bootstrap.py and a PDF called *userid*-conf/bootstrap-500.pdf, the result of $TRIALS = 500$.