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.

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 pull 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, how do we get an interval? I.e., how do we get more than one sample mean from one sample? We can't; we have to get more samples. How do we get lots of samples from an underlying distribution that we cannot identify? By repeated *resampling* from our single sample *with replacement*. This is called *bootstrapping*. To get a new sample, the idea is to randomly select N values from our known data set of size N. That gives us one trial. We then do more trials, computing our test statistic, the mean, on each sample.

The histogram of the sample means from all trials approximates the sample mean distribution, which we know from the central limit theorem, is $Normal(\mu, \sigma^2/N)$ for population mean μ and sample size N (not the number of trials). We don't know the underlying distribution because we've just got a bunch of prices from a file, but we don't care. The central limit theorem works on just about any underlying distribution. We don't need the distribution shape, but we do need the mean, μ, and variance, σ^2. For those, we can use the sample mean, $\overline{X}$ and sample variance, s^2, as population estimates.

To bootstrap, all we have to do is repeat the following: conjure up a new sample, X, then compute and track the mean $\overline{X}$. For M such trials, we get M sample means. 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 100, we drop the smallest 3 (or 2) and largest 3 (or 2) means from the list of 100 means. Then, the lowest and highest value in that truncated list represent the boundaries of the confidence interval. Cool, right?

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: $\mu \pm 1.96\sigma$.

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

Steps

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

```python
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`). These then 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 has two lines in it.

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 from indices TRIALS*0.025..TRIALS*0.975 in X_ and call it `inside`.

5. Print the first and last value of the `inside` array as that will tell you what the bounds of your 95% confidence interval are

```python
print inside[0], inside[-1]
```

5% of 40 is 2 which means grab 1 element on either side. You might get something like (there will be a lot of variation):

```
1.1270115942 1.17842898551
```

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')
```

8. Run it and you should get the following graph:
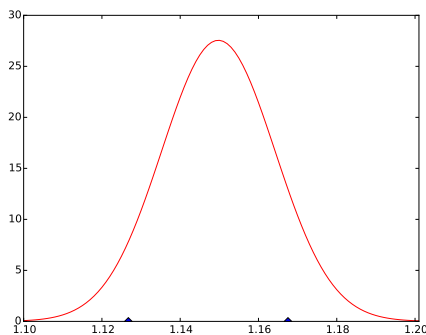
Ok, that's great but we have no idea if this is correct or not. Now, let's go nuts and show lots of stuff on the graph.

9. First, let's shade in the theoretical 95% confidence interval using a function called `normpdf()` that you implement from the symbolic definition of the normal density function:
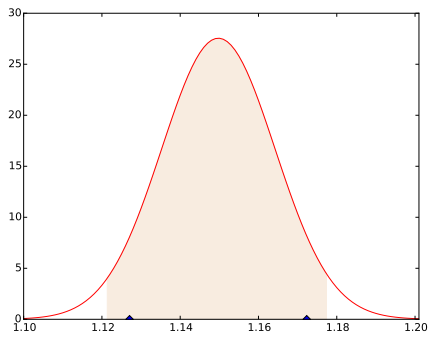
$$normpdf(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$$

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

mean = ...
stddev = ...
# redraw normal but only shade in 95% CI
left  = ...
right = ...

ci_x = np.arange(left, right, 0.001)
ci_y = normpdf(ci_x,mean,stddev)
# 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.

```python
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)
plt.text(.02,.75, '95%% CI = $%1.2f \\pm 1.96*%1.3f$' %
    (np.mean(X_),np.std(X_,ddof=1)), transform = ax.transAxes)
plt.text(.02,.70, '95%% CI = ($%1.2f,\\ %1.2f$)' %
                (np.mean(X_)-1.96*np.std(X_),
                 np.mean(X_)+1.96*np.std(X_)),
        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=(inside[0], .3),
            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:
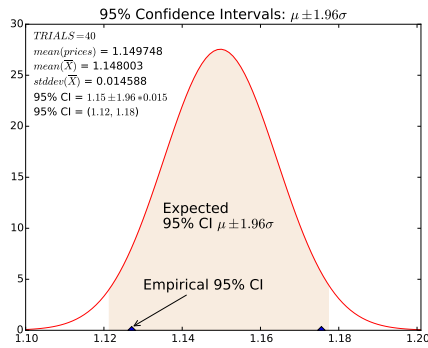
12. We need to make the number of trials a parameter and let's do the annotations as well.

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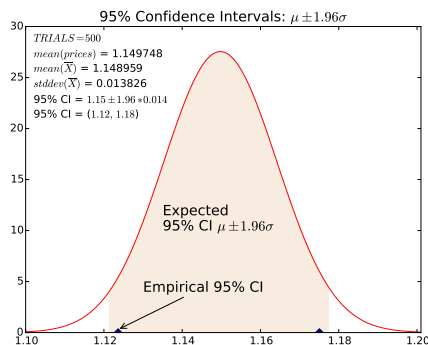
Then wrap your annotation code in `if fancy` and add code to save your graph (in all cases):

```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. Try 500:

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

You'll get:

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