Homework 4: Standard Errors of the Cat Heart

36-350, Fall 2013

Due at 11:59 pm on Thursday, 26 September 2013

INSTRUCTIONS: Read the whole assignment carefully before beginning. You can use code from solutions to earlier homeworks or labs, with attribution.

Direct objective: Practice with top-down design and code testing. *Indirect objectives:* Fitting statistical models; using simulations to test statistical procedures.

In the last lab, we fit a gamma distribution to the weight of cat's hearts. We did this by adjusting the parameters so that the theoretical values of the mean and variance matched the observed, sample mean and variance. Since the mean and variance are the first two **moments** of the distribution, this is an example of the **method of moments** for estimation.

The method of moments gives a point estimate $\hat{\theta}$ of the parameters θ. To use a point estimate, we need to know how **precise** it is, i.e., how different it would be if we repeated the experiment with new data from the same population. We often measure imprecision by the **standard error**, which is the standard deviation of the point estimates $\hat{\theta}$. (You saw the standard error of the mean in your introductory statistics classes, but we are *not* computing the standard error of the mean here.)

If we actually did the experiment many times, getting many values of $\hat{\theta}$, we could take their standard deviation as the standard error. With only one data set, we need to do something else. There is usually no simple formula for standard errors of most estimates, the way there is for the standard error of the mean. Instead, we will see how to approximate the standard error of for our estimate of the gamma distribution computationally.

We can draw random values from a gamma distribution using the `rgamma()` function, which works like `rexp()` from lab 1.

```
rgamma(n=35,shape=0.57,scale=15)
```

would generate a vector of 35 random values, drawn from the gamma distribution with "shape" parameter $a = 0.57$ and "scale" $s = 15$. (Alternately, one can give the argument `rate`, which is $1/s$.) By applying the estimator to random samples drawn from the distribution, we can see how much the estimates will change purely due to noise.

1. (5 total) *Start with the point estimate*

   (a) (4) Write a function, `gamma.est`, which takes as input a vector of data values, and returns a vector containing the two estimated parameters of the gamma distribution, with components named `shape` and `scale` as appropriate. You can re-use your code from lab, or use the code from the lab solutions. If you use the solutions, say so, and modify the return value appropriately.

   (b) (1) Verify that your function implements the appropriate formulas by showing that it matches the results in the lab solutions for all cats together, for the female cats, and for the male cats.

2. (15 total) *Draws from the gamma distribution*

   (a) (5) Generate a vector containing ten thousand random values from the gamma distribution with $a = 19$ and $s = 0.56$. What are the theoretical values of the mean and of the variance? What are their sample values?

   (b) (5) Plot the histogram of the random values, and add the curve of the theoretical probability density function.

   (c) (5) Apply your `gamma.est` function to your random sample. Report the estimated parameters and how far they are from the true values.

3. (25) *Top-level function* Write a function, `gamma.est.se`, to calculate the standard error of your estimates of the gamma parameters, on simulated data drawn from the gamma distribution. It should take the following arguments: true shape parameter `shape` (or a), true scale parameter `scale` (or s), size of each sample `n`, and number of repetitions at that sample size `B`. It should return two standard errors, one for the shape parameter a and one for the scale parameter s. (These can be either in a vector or in a list, but should be named clearly.) It should call a function `gamma.est.sim` which takes the same arguments as `gamma.est.se`, and returns an array with two rows and `B` columns, one row holding shape estimates and the other row scale estimates. Your `gamma.est.se` function should *not*, itself, estimate any parameters or generate any random values.

4. (15) *Testing with a stub* To check that `gamma.est.se` works properly, we write a **stub** or **dummy** version of `gamma.est.sim`, which takes the correct arguments and returns an array of the proper size, but whose entries are fixed so that it's easy for us to calculate what `gamma.est.se` ought to do.

   (a) (5) Write `gamma.est.sim` so that the entries in the first row of the returned array alternate between `shape` and `shape+1`, and those in the second row alternate between `scale` and `scale+n`. It should match the following, except for the row names, which are optional.

```
> gamma.est.sim(2,1,10,10)
       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
shapes    2    3    2    3    2    3    2    3    2     3
scales    1   11    1   11    1   11    1   11    1    11
> gamma.est.sim(2,8,5,7)
       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
shapes    2    3    2    3    2    3    2
scales    8   13    8   13    8   13    8
```

(b) (2) Calculate the standard deviations of each *row* in the two arrays above.

(c) (8) Run your `gamma.est.se`, with this version of `gamma.est.sim`. Do its standard errors match the standard deviations you just calculated? Should they?

5. (25 total) *Replacing the stub*

   (a) (20) Write the actual `gamma.est.sim`. Each of the `B` columns in its output should be the result of applying `gamma.est` to a vector of `n` random numbers generated by a different call to `rgamma`, all with the same `shape` and `scale` parameters. For full credit, use `replicate` rather than looping. *Hint*: Look at lecture 3, towards the end.

   (b) (5) Run `gamma.est.se`, calling your new `gamma.est.sim`, with `shape=2`, `scale=1`, `n=10` and `B=1e5`. Check that the standard error for `shape` is approximately 1.6 and that for `scale` approximately 0.54. Explain why this problem cannot give exact control values.

6. (5, extra credit) *Jackknife vs. simulation* Explain the difference between this way of calculating standard errors and the "jackknife" method from lab 4. How closely do the standard errors you got here match those you got in lab? Do the differences matter? Explain why you'd prefer one technique over the other.