Stat243: Problem Set 5, Due Wednesday Oct. 22

October 15, 2014

Comments:

- This covers Unit 7.
- It's due at the start of class on 10/22.
- Note that you can mix and match R and Python chunks in your knitr/Rmd documents. Simply use `engine='python'` for the Python ones.
- As usual, simply providing the raw code is not enough; make sure to describe how you approached the problem, the steps you took, and output illustrating what your code produces.
- Please note my comments in the syllabus about when to ask for help and about working together.
- As discussed in the syllabus, please turn in (1) a copy on paper, as this makes it easier for us to handle AND (2) an electronic copy through Git following Jarrod's instructions.

Questions

1. You're working with a collaborator on a statistical model. They have the following independent binomial data, $y_i \sim \text{Binom}(n, p_i)$, for which they are estimating the likelihood as

$$\begin{aligned}\prod_{i=1}^{m} P(y_i; n, p_i) &= \prod_{i=1}^{m} \text{Binom}(y_i; n, p_i)) \\ \log \frac{p_i}{1-p_i} &= X_i^\top \beta\end{aligned}$$

   using *prod()* and *dbinom()* in R. They tell you that they are getting zero for the likelihood regardless of the value of β, using the data in the file *ps5prob1.Rda* (this file provides one possible β value as well as y and X). Verify this and then explain to them what the problem is and how to do the calculation.

2. Here we'll consider the effects of adding together numbers of very different sizes. Let's consider adding the number 1 to 10000 copies of the number 1×10^{-16}. Mathematically the answer is obviously $1 + 1 \times 10^{-12} = 1.000000000001$ by multiplication, but we want to use this as an example of the accuracy of summation with numbers of very different magnitudes, so consider the sum $1 + 1 \times 10^{-16} + \cdots + 1 \times 10^{-16}$.

   (a) How many decimal places of accuracy are the most we can expect of our result (i.e., assuming we don't carry out the calculations in a dumb way). In other words, if we store 1.000000000001 on a computer, how many decimal places of accuracy do we have? This is not a trick question.

(b) In R, create the vector $x = c(1, 1 \times 10^{-16}, \ldots, 1 \times 10^{-16})$. Does the use of *sum()* give the right answer up to the accuracy expected from part (a)?

(c) Do the same as in (b) for Python. For Python the *Decimal()* function from the decimal package is useful for printing additional digits. Also, this code will help you get started in creating the needed vector:

```
import numpy as np

vec = np.array([1e-16]*(10001))
```

(d) Use a *for* loop to do the summation, $((x_1 + x_2) + x_3) + \ldots$. Does this give the right answer? Now use a *for* loop to do the summation with the 1 as the last value in the vector instead of the first value. Right answer? If either of these don't give the right answer, how many decimal places of accuracy does the answer have? Do the same in Python.

(e) What do the results suggest about how R's *sum()* function works? Is it simply summing the numbers from left to right?

(f) Extra credit: Figure out what R's *sum()* function is doing, either by finding documentation or by looking at the actual code used.

This should suggest that the *sum()* function works in a smart way, but that if you calculate the sum manually, you need to be careful in some situations.