Let us not forget the basics

Ben Klemens

28 July 2013

This post concludes a series I've been doing on generic model objects and their payoff. The big payoff is in what I think of as *narrative modeling*, in which we think about what actually transpired in the real world to cause the data we see to be the way we see it, and then write down a model following that narrative.

I'm not the first to think of this, and hierarchical, Bayesian, and agent-based modeling all follow that sort of form. But it's a nontraditional way of doing statistics, and the tools aren't written around it. We need to be able to use RNG-based models and creatively compose models from simpler sub-models, so we had to start with developing a model object and then work our way forward from there.

But most of the statistical world is built around simple pseudonarratives where there is a single (possibly multivariate) distribution expressible in closed form, or a function of the form

$$f^0(Y_i) = \beta_0 + \beta_1 f^1(x_i^1) + \beta_2 f^2(x_i^2) + \cdots + \beta_n f^n(x_i^n) + \epsilon,$$

where each $f^j(\cdot)$, $j = 0, \ldots, n$, is a deterministic function of one variable (on the outside, two), and ϵ has a distribution expressible in closed form. This generalized linear model (GLM) is a realistic narrative only in certain cases—certainly not in the immense range of situations in which it is used.

So, I don't like it. I'm happy to see that it's largely losing fashion among research statisticians, who seem (from my perspective) decreasingly obsessed with stretching that form into more situations and more interested in trying alternative functional forms where randomness isn't just an ϵ tacked on at the end of a deterministic narrative.

But the reality is that:

- To most people, the GLM encompasses all of statistics. This is what they learned in the one or two stats classes they took, and they have better things to do than to find out that there's more in the statistical world.
- The GLM actually works pretty well in a lot of situations. It is an inaccurate tool, but if we just want to know if $\log(A)$ goes up or down in proportion to B, and whether we should care more about the effect of B on A or the effect of C on A, it will tell us. If you call me at 9AM and tell me you need to know how A, B, and C relate by 11AM, I'm going to run linear models, not develop a realistic narrative. It would be wonderful if every first approximation using a GLM was followed by a model that the model author actually believes to be possibly true,

but here in reality there is often not time, and see the above bullet point about what people have learned.

Changing gears The last several entries were about narrative modeling, and the demo code used Apophenia, the library of stats functions I wrote to make narrative modeling feasible and friendly. In certain parlance, the last several entries have been Rocket Science [except Ballistics is one of those cases where GLMs sometimes work very well]. I tried to keep the stress on narrative modeling, not Apophenia.

The next several entries will be about doing much more common tasks—even fitting linear models—using Apophenia. I'll cover both the mechanics of what's going on, and the design decisions behind it all.

These are things that other languages and platforms also do, but with a different platform and some different underlying design decisions.

Given that you can do these things using other platforms, the question *Why use C for these things?* comes up. One answer is embodied in the last dozen entries, which required real computational firepower and a system that's a little more structured and a little less do-what-I-mean than many existing systems. C is also better for embedding: if you have a program that is not about statistics, but has a data analysis component, and you need to do a quick correlation or calculate a line of best fit, C may be the right language to do that in, especially given that *everything* has a C API.

But we could just as easily rephrase the question: *Why not use C for these things?* As will be shown, most common data-oriented tasks take about as many lines of C as Python, R, or whatever code. I have received more than enough communications from detractors who told me that it takes ten times as many lines of C code to do something as it takes in R, which just tells us that the detractor really has no idea how to write in C. If somebody tells you something like this, please point them to this series.

Next time, I'll read in a data set, generate a simple 2×2 crosstab, and run a simple test about the table. Depending on how you choose to count, it'll be about five lines of code.

[PS: I'll try to post every four days. Posting every three days was a bit of a challenge, especially given that over some three-day periods during which normal life transpired, I was developing a full agent-based model and applying relatively novel-for-ABMs analysis techniques or transformations to the model. I felt vindicated that I really had developed a platform where such heavy lifting could be done rapidly, but on the other hand, there's more to life than blogging. . . .]