A general model object

Ben Klemens

4 May 2009

Last time, I talked about the balkanization of modeling: basic statistical models, agent-based models, many types of physical models, are all academically disjoint, by which I mean that few people simultaneously use more than one of these paradigms.

This time, I want to bring it to a more practical level: if your model is anything beyond a never-to-be-estimated system of equations, it will be expressed in computer code. But our software has as much diversity in conceptions of a model as our academic departments do. If I want to do a regression, I can use R and its pleasant generalized linear model (GLM) interface; if I want to do agent-based modeling, I can use any of a number of object-friendly frameworks—but not R, which is terrible for such modeling. [We are not having the discussion about computationally-intensive modeling in R now, but trust me when I say it's not the right tool for the job.] If you're doing an EE-type simulation with polar coordinates or complex variables, then you're more likely to be doing it in Matlab, and have probably never heard of R.

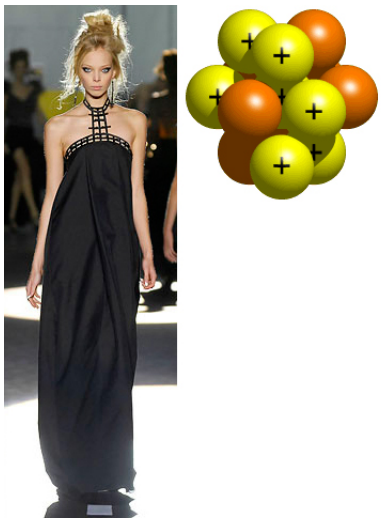

Figure 1: Different fields have diverse and hard-to-reconcile conceptions of a model.

The balkanization creates problems for those who want to do something outside of

the group norms. If I want to use an agent-based model as a prior and then do Bayesian updating using observed data, none of the above will accommodate me. If I want to try hierarchical modeling tools to simplify and approximate a complex circuit diagram, I'll be doing a whole lot of inter-package negotiation. Generally, if I want to talk across different disciplines (or even sub-disciplines), I have to have multiple conceptions of a model in my head at once. Many humans have no problem with this, but few software packages are capable of such a feat.

A standard model form Is there a general definition of a model that we can operationalize into code, which would allow us to interchange and combine models in one place?

I've lost a lot of sleep over that question. A definition that's too general is basically vacuous—we can just say that a model somehow maps from one set of functions to another, and that'll cover anything you can think of, but won't say anything. If we restrict models to GLMs, we can get an immense amount of work done, but at the cost of balkanizing away all the fields where modeling doesn't mean regressions. The engineering challenge is in finding a decent balance between generality and practicality.

Here's what I came up with in the end. I present it to you not only because I think it's an important question, but to challenge you to think of what you would or would not include in an implementation of a general model. My own definition, which I've operationalized, reduces down to this:

> A model intermediates between data, parameters, and likelihoods.

There are four significant words in the definition, which require four more definitions:

Parameters: These you are familiar with. The mean and variance of a Gaussian distribution, the coefficients of a regression, or a simulation's tweaks about frequencies, population size, &c.

Data: Data is generally the other input: the draw from the Normal distribution whose odds you're finding or the observations for your regression. When doing estimation, we think in terms of the data being an input and the parameters being an output, but you'll see below that there's more symmetry than that; we can turn the tables and fix the parameters to produce artificial data.

Likelihoods: The odds of the given parameters and data co-occurring. For a Normal distribution, this is what you get from looking up the data point on the appropriate table. For a regression, the likelihood is calculated under the assumption that errors have a Normal distribution.

By having a likelihood, does the definition force us to stick to probabilistic models? No, because there is always some means of evaluating the quality of the model, and that can be read as a likelihood. Typically, this is a distance to some sort of ideal, or some objective to be maximized. If you don't believe me that a distance function can generate a subjective likelihood, then just take this as metaphorical (until I have a chance to post an entry on why this is the case, and why we shouldn't distinguish between likelihood and probability).

Intermediation: We could go in three meaningful directions among the above.

Data $\Rightarrow$ Parameters. This is the standard estimation problem. You have some data, and find the most likely mean and variance of the Normal distribution, or the regression coefficients that produce the line of best fit. We invariably use the likelihood function to do this.

Data + Parameters $\Rightarrow$ Likelihoods. That is, the odds of having the given inputs. These are the tables in the back of statistics textbooks for different distributions, with parameters along the columns and data values along the rows.

Parameters $\Rightarrow$ Data. Given a set of fixed parameters, we can find the most likely data, or the expected data, or make random draws to produce an artificial data set consistent with the model and parameters.

These methods are generally linked, and you can often solve one from the other. For example, if you give me a likelihood function ($D + P \Rightarrow L$), I can find you the optimal parameters via maximum likelihood estimation (MLE, $D \Rightarrow P$). This is a boon for software design, because when a model doesn't have a quick method for estimating parameters, I can fill in a default method; when it does, like the case of ordinary regression, I can use that instead of the general but computationally-intensive MLE.

You can see that there's a lot more that we want our models to do than just direct estimation of parameters from data. You may have in mind other things that one would do with a model that aren't covered; I drew the line here based on the above problem of writing a definition that is both inclusive and operationalizes into something useful.

But I should note testing, which is certainly one of the more common things to do with a model. I separate testing from the model proper, because of everything I've already said about the importance of differentiating the descriptive and inferential sides, and entry #007 on building a pipeline with separate estimation and testing steps. From such a perspective, testing is not a part of the model, but something you can do with it.

Every test has the same form: establish a distribution; locate the data and a reference point (usually zero); establish the odds that the data and reference point differ given the distribution. I toyed with the idea of establishing a generalized testing object, but saw little benefit, being that the test is already typically a single line of code: looking up the CDF of the appropriate distribution at the given point. If you don't have a CDF on hand, you can of course generate it from the model via random draws.

[Which brings us to one more thing we often want to do with a model: generate a CDF. I implement this as a histogram, which is just another view of the same model.]

Be creative What's the benefit of this standardization of models? First, we can write methods to work with any model. Part of the estimation-testing pipeline that we often see in many stats packages is a tendency to put the code for certain methods inside the code for models with which the methods are closely associated, which means that the method isn't available when some other model wants to use it. With methods that take black-box models as inputs, we have a better chance of applying methods from discipline A to models from discipline B.

We can test models against each other. This could mean a Poisson versus an Exponential, or as above, it could mean a theoretical distribution versus a histogram fitted

to the model. Or how about an agent-based model with or without some extra moving part? Most importantly, we can compare a common baseline model, like a simple linear regression, against a relatively complex simulation.

My goal in all of this was to use a simulation to generate a probability distribution. In the format here, that makes sense, and is easy: just specify a likelihood function based on a distance to an optimum, use it to estimate the parameters of the model, then use those optimal parameters to find likelihoods for other purposes. Because the model is a black box with a limited set of interfaces (like the likelihood function and the parameter estimation routine), we don't have to care about the methodological innards of the model, and can use it as we would a Poisson distribution.

For many purposes this black-boxing is exactly the right way to deal with a plethora of models. Excluding some models from some uses because of their methods of computation is just blunt bigotry, which has no place in our enlightened era. There are still some practical considerations about what works best in a given situation, but you probably thought about those things during the pencil-and-paper stage of the project. It is not the place of the software to place restrictions on your creativity.

[Not one to just chat about the theory, this is a bird's-eye view of the `apop_model` object actually implemented in the Apophenia library. There are still technicalities to be surmounted, which I discuss in the coder-oriented design notes[1].]

[1] `http://apophenia.sourceforge.net/apop_notes.html`