Building Zelig Modules

Matt Owen

July 2, 2010

1 Outline of a Zelig Module

The Zelig Software Package is intended to give statisticians (or anyone interested in developing statistical models) a flexible and intuitive means to create statistical models under a unified language. That is, Zelig standardizes the language used to interpret and simulate statistical models. To this end, Zelig incorporates several features used to streamline and standardize how to write such a model.

In order to create a Zelig module several simple functions must be written:

do.zelig.*model name* A simple function that acts as an interface between the Zelig module and the already developed model. Specifically, this do.zelig.*model name* organizes how Zelig calls the pre-existing model that Zelig is wrapping.

qi.*model name* This function is the most important component of a Zelig module. It specifies and computes the "quantities of interest" of the model. That is, qi.*model name* uses the fitted model computed by the "zelig" function to simulate quantities of interest.

param.*model name* This function is used by qi.*model name* to aid in producing the quantities of interest. While it is not strictly necessary to use this function, it often helps improve the clarity of the code.

In addition to those basic functions, the experienced Zelig developer may choose to override the existing methods for displaying data. In order to do this, one simply needs to create or edit one of the following functions:

summary.*model name* This returns the summary data on an R object. This essentially amounts to a list describing important components of one's Zelig module.

print.*model name* This function prints quantities of interest to the screen.

For complicated models, the developer may want to use programming "hooks" or a complete re-write of sim.*model name* to ensure that everything is computed correctly. More details on this in the API specification. This, in particular, crops up whenever the developer must make function calls to code compiled from C or FORTRAN.

2 The Zelig Model-Building API

2.1 build.zelig2

build.zelig2 is a function that registers a module with the actual Zelig software package. It offers a quick way to produce Zelig models without having to re-package the entire Zelig software suite. In essence, it boils down the do.zelig.*model name* to bare minimum. For almost all produced models it is a completely sufficient replacement for a do.zelig.*model name* function.

2.2 description of the "build.zelig2" function call

Note: ..1 and ..2 are pseudonyms for the first and second parameters passed into the "build.zelig2" function. This essentially means that the first and second parameters cannot be set via keyword (and should not be unless the user knows the exact consequences of this action).

..1: The first argument is a character-string specifying the model name.

..2: The second argument is character-string or function specifying the external model that Zelig will call.

...: A set of arguments of either:

- A key-value pair representing a parameter being explicitly set for the model.
- A character-string representing the name of a parameter being set by the end-user.

2.3 Basic Rules for build.zelig2 Calls

1. The first parameter is a character-string representing the name of the Zelig model that is being created. If the first parameter is "ls", then it is invoked by setting the model parameter to "ls" in the zelig call.

2. The second parameter is a function (or a character-string holding the name of the function) that Zelig will used to create the fitted model. If the second parameter is set to 'lm' (or "lm"), then Zelig will use the lm function (which performs a linear fit to the data) to fit the data.

3. All parameters other than the first and second follow a simple guideline:
   - If it is key-value pair, the exact value specified will be passed directly to the model-fitting funciton.
   - If it is a character-string without a key, then Zelig will search the parameters passed by the end user for one that matches the specified character-string.

2.4 Example Call

build.zelig2("ls", lm, model=F, "formula", "data")

2.5 Explanation of Example Call

"ls" This is the first parameter, and as a result, specifies the name of the zelig model that we are creating. In the above example, this ensure that a Zelig model with the name "ls" is created. It does nothing more than this.

lm This is the second parameter, and as a result, the outside functoin to call. That is, the second parameter is later used by Zelig to actually fit the statistical model given the parameters we pass it. In the above example, lm (the function call to fit a linear model) specifies this.

model=F This is a parameter other than the first or second, and as a result, follows the guidelines outlined in section 1.2. In the above example, this will set the "model" parameter to F (equivalent to 'FALSE') anytime a "ls" model is created by Zelig.

"formula" This is a parameter other than the first or second, and, as a result, follows the guidlines in section 1.2. Because there is no equal sign to the left of the string, this parameter specifies that Zelig should expect the end-user to specify a value for formula. That is, Zelig will rely on the end-user to specify a parameter titled "formula".

"data" Similarly to the case with "formula", this parameter tells Zelig that it should expect a parameter specified by the end-user, titled "data".

3 do.zelig.*model name*

The do.zelig.*model name* function serves the sole purpose of specifying and setting the parameters of the model-fitting function. That is, this function explains to Zelig what parameters should be passed to the actual function that computes the model. The following are equivalent ways to create a valid do.zelig.*model name* function

Method 1

This is the first - and tersest - way to implement a least-squares mode in Zelig. It is nearly completely equivalent to the build.zelig2 method.

```
do.zelig.ls <- function(model, formula, ..., data, weights=NULL)
  list(lm, model = F, "formula", "weights", "data")
```

Method 2

This method offers a more explicit way to create the model. When in doubt, it is best to create the model in this fashion.

```
do.zelig.ls <- function(model, formula, ..., data, weights=NULL)
  list(lm,
       model = F,
       formula = formula,
       weights = weights,
       data = data
       )
```

Method 3

This final method makes use of the zelig.call class to create the model. It offers no true benefit over any of the other methods, except for the truly savvy Zelig developer.

```
do.zelig.ls <- function(model, formula, ..., data, weights=NULL)
  zelig.call(lm,
             default = list(model = F),
             forward = c("formula", "weights", "data")
             )
```

Unfortunately, not all methods can be implemented so straight-forwardly. In particular, some models require the formula to be written using very particular syntax. If this is the case, the developer must add additional code that puts the formula in the correct format. This procedure varies on a module-to-module basis.

4 setx: setting explanatory variables

5 The 'by' keyword: running simulations on subsets of data

The 'by' keyword is a particularly powerful parameter that may be passed to the 'zelig' function; it allows the user to subset the data-frame by the specified factors. That is, the example function call:

$$zelig(vote\ race + educate, model = "logit", data = turnout, by = “race'')$$

will divide the *turnout* into individual data-frames into their respective values for “race”. In this case, those values are “white” and “others”.

6 The 'sim' function: simulating quantities of interest

Every Zelig model has a *sim* function. This, when called, directs the Zelig module on how to compute quantities of interest. In most cases, the *sim* function does not need to be overloaded.

6.1 Workflow of the 'sim' function

Before a 'sim' object is returned from this function call, several steps take place:

sim This is the entry-point of the *sim* function. It is responsible for all error-catching and function calls used to compute *quantities of interest.* It is passed *zelig* object and, optionally, one or two *setx* objects.

param This (optionally created) function typically samples the probability distribution implied by the fitted model. For example, in the case of a logit model, the param function returns a set of samples drawn from Multinomial-Normal distribution, based on the coefficients of fitted model and their covariance matrix. This step is similar to the idea of predicting values from the model.

qi This (required) function computes the quantities of interest. This step can differ dramatically between models, so it is recommended the user takes care in writing this function. No default *qi* function is packaged with Zelig.

summarize This (required) function summarizes the results from *qi*'s simulation step. This function does not typically need to be written, and by default computes the mean, standard deviation, median, and 2.5% and 97.5% quantiles.

6.2 param

The *param* function, as stated above, is a helper function that generates samples taken from the distribution that defines the fitted model. This function is entirely optional, and is typically invoked by the *qi* function to produce the random elements needed to simulate a quantity of interest.

While this function is not necessary, it is often useful to write in order to clarify how one thinks about the specific model.

6.3 qi

The *qi* function is perhaps the most important function in any Zelig module; it manages what a *quantity of interest* is, as well as, how it is computed. This step varies greatly from model to model, however there are several important steps that should be followed:

1. *Call the param function*: This is entirely optional but sometimes important for the clarity of your algorithm. This step typically consists of taking random draws from the fitted model's underlying probability distribution.
2. *Compute the Quantity of Interest*: Depending on your model, there are several ways to compute necessary quantities of interest. Typical methods for computing quantities of interest include:
   (a) Using the 'predict' method of your given linear model.
   (b) Using the sample provided by 'param' to generate simulations of the *Quantities of Interest.*
   (c) Using a Maximum-likelihood estimate on the fitted model.
3. *Create a list of titles for your Quantities of Interest*:
4. *Generate the Quantity of Interest Object*: Finally, with the computed Quantities of Interest, you must

6.4 Creating a "qi" Object

There are several methods for casting the results of your *qi* function to

1. Returning a list (or call to *make.qi*) of key-value pairs:
   - make.qi("Title #1" = qi_1, "Title #2" = qi_2, "Title #3" = qi_3)
   - list("Title #1" = qi_1, "Title #2" = qi_2, "Title #3" = qi_3)

2. - as.qi(list(
     c("Title #1", "Title #2", "Title #3"),
     list(qi1, qi2, qi3)
     ))
   - list(list(
     c("Title #1", "Title #2", "Title #3"),
     list(qi1, qi2, qi3)
     ))

For the truly savvy Zelig developer, the latter option comes with several features that may be useful, however, it lacks the elegance of the first method. When in doubt, the first method offers a mix of expressiveness and clarity that will typically outweigh the slightly extended functionality of the latter.