Fitting Multiple Statistical Models Simultaneously

It is often important, especially when computing data based on counterfactuals, to have a side-by-side comparison of the different fitted models and their *quantities of interest*. Typically, the statistician interested in comparing multiple models would:

- Prepare Several Different Datasets, by:
  - Splitting a Dataset into Several Categorized by Some Particular Quality
  - Preparing Counterfactual Data Based on a Pre-existing Dataset
  - Collecting Several Different Data-sets Analyzing Similar Data
- Compute a Fitted Model for Each Dataset
- Simulate Quantities of Interest for Each Fitted Model
- Organize the Results in a Logical Fashion

This process can be quite arduous, and the programming syntax associated with these sorts of processes is often unintuitive and cumbersome. Resulting from the necessity to simplify this process, the Zelig software package offers an intuitive and simple syntax to produce these results.

1 The *mi* class: An Easy Method for Multiply Imputed Data

Zelig offers a simple way to fit models for multiply imputed data. By submitting an *mi* object to the data parameter of Zelig, the statistician can prepare analysis of multiply imputed data quickly and easily.

1.1 Parameters and Return-values of the "mi" Function

@...: A series of *data.frame* objects. Note: if a non-*data.frame* object is submitted, the *mi* class will raise an error.

Return Value: An *mi* object that the *zelig* function will use to run analysis on the datasets.

1.2 Examples of Multiply Imputed Data

```
library(Zelig)

data(turnout)

# split the turnout data into 2 sets
turnout1 <- turnout[1:1000,]
turnout2 <- turnout[1001:2000, ]

# create the mi object
split.turnout <- mi(turnout1, turnout2)

# fit the models
z.out <- zelig(vote ~ race + educate, model="logit", data=split.turnout)

# set a counterfactual on both data-sets
x.out <- setx(z.out, age=65)

# simulate quantities of interest for both models
s.out <- sim(z.out,  x = x.out)

# output results
summary(s.out)
```

Important Notes

- When *data* is submitted to the zelig function via the *mi* class, Zelig returns an object of both class *MI* and *zelig*. This is in contrast to usually simply being of the class *zelig*.
- ...

1.3 Developing Models to Interact with *mi* Objects

For the most part, the Zelig developer should not need any additional code to have their model work with the *mi* class. However, an API exists for the managing and printing of results from this class.

2 The *by* Keyword: Subsetting Datasets by Factors

For some models, a the significance of a factor variable - a variable whose range is a finite, specific number of values - is best understood by subsetting

a dataset by each of its potential values. For example, when trying to analyze voting patterns, it may make sense to fit your model separately by race, religion, or birth-state. Typically, this would require creating a new data-set for each of the possible combinations of race, religion, and birth-state. Zelig, via the emphby keyword, employs a straightforward method to create such models.

2.1 How-to Use the *by* Keyword

- *by* must be a column name of the dataset (or datasets) being passed to the model
- The formula must not contain any of the values passed into *by*
- *by* should be a column describing factors - not numerical data. Otherwise, results may be too numerous or nonsensical
-

2.2 Code Example

```
library(Zelig)

data(turnout)

z.out <- zelig(
        vote ~ age + income + educate,
        model = "logit",
        data  = turnout,
        by    = "race"
        )

x.out <- setx(z.out)

s.out <- sim(z.out, x.out)

summary(s.out)
```

2.3 Explanation of the Above Code

The above code individually fits, analyzes, and simulates a model for voter turnout based on age, income, and race. By passing the *by* keyword the value "race", our model knows to split the turnout data object into separate components: one for each different in the data column labeled "race". The beauty in this script is the simplicity in which it does this process. That is,

it differs in no way from a typical call to *zelig* with the exception that the *by* keyword is specified.

3 Using the *mi* Class and the *by* Keyword Together

No special programming is necessary to use both the *mi* class along with the *by* keyword! Just simply pass an *mi* object to data, and pass the names of columns to subset the data by to the *by* parameter, and *zelig* will handle the rest. In fact, the printout that Zelig will produce will automatically categorize the data according to which dataset was submitted and which factor was subsetted. For complex operations an API is available for the savvy developer.