Zelig: Everyone's Statistical Software[1]

Kosuke Imai[2] Gary King[3] Olivia Lau[4]

August 4, 2011

[1]The current version of this software is available at `http://gking.harvard.edu/zelig`, free of charge and open-source (under the terms of the GNU GPL, v2)

[2]Assistant Professor, Department of Politics, Princeton University (Corwin Hall, Department of Politics, Princeton University, Princeton NJ 08544; `http://imai.princeton.edu/`, `kimai@princeton.edu`)

[3]Albert J. Weatherhead III University Professor, Harvard University (Institute for Quantitative Social Sciences, 1737 Cambridge Street, Harvard University, Cambridge MA 02138; `http://gking.harvard.edu`, `king@harvard.edu`, (617) 495-2027).

[4]Ph.D. Candidate, Department of Government, Harvard University (1737 Cambridge Street, Cambridge MA 02138; `http://www.people.fas.harvard.edu/~olau`, `olau@fas.harvard.edu`).

Contents

Part I

Introduction

Chapter 1

Introduction to Zelig

1.1 What Zelig and R Do

Zelig[1] is an easy-to-use program that can estimate and help interpret the results of an enormous and growing range of statistical models. It literally *is* "everyone's statistical software" because Zelig's unified framework incorporates everyone else's (R) code. We also hope it will *become* "everyone's statistical software" for applications, and we have designed Zelig so that anyone can use it or add their models to it.

When you are using Zelig, you are also using R, a powerful statistical software language. You do not need to learn R separately, however, since this manual introduces you to R through Zelig, which simplifies R and reduces the amount of programming knowledge you need to get started. Because so many individuals contribute different packages to R (each with their own syntax and documentation), estimating a statistical model can be a frustrating experience. Users need to know which package contains the model, find the modeling command within the package, and refer to the manual page for the model-specific arguments. In contrast, Zelig users can skip these start-up costs and move directly to data analyses. Using Zelig's unified command syntax, you gain the convenience of a packaged program, without losing any of the power of R's underlying statistical procedures.

In addition to generalizing R packages and making existing methods easier to use, Zelig includes infrastructure that can improve all existing methods and R programs. Even if you know R, using Zelig greatly simplifies your work. It mimics the popular Clarify program for Stata (and thus the suggestions of King, Tomz, and Wittenberg, 2000) by translating the raw output of existing statistical procedures into quantities that are of direct interest to researchers. Instead of trying to interpret coefficients parameterized for modeling convenience, Zelig makes it easy to compute quantities of real interest: probabilities, predicted values, expected values, first differences, and risk ratios, along with confidence intervals, standard errors, or full posterior (or sampling) densities for all quantities. Zelig extends Clarify by seamlessly integrating an option for bootstrapping into the simulation of quantities of interest. It also integrates a full suite of nonparametric matching methods as a preprocessing step to improve the performance of any parametric model for causal inference (see MatchIt). For missing data, Zelig accepts multiply imputed datasets created by Amelia (see King, Honaker, Joseph, and Scheve, 2001) and other programs, allowing users to analyze them as if they were a single, fully observed dataset. Zelig outputs replication data sets so that you (and if you wish, anyone else) will always be able to replicate the results of your analyses (see King, 1995). Several powerful Zelig commands also make running multiple analyses and recoding variables simple.

Using R in combination with Zelig has several advantages over commercial statistical software. R and Zelig are part of the open source movement, which is roughly based on the principles of science. That is, anyone who adds functionality to open source software or wishes to redistribute it (legally) must provide the

[1] Zelig is named after a Woody Allen movie about a man who had the strange ability to become the physical reflection of anyone he met — Scottish, African-American, Indian, Chinese, thin, obese, medical doctor, Hassidic rabbi, anything — and thus to fit well in any situation.

software accompanied by its source free of charge.[2] If you find a bug in open source software and post a note to the appropriate mailing list, a solution you can use will likely be posted quickly by one of the thousands of people using the program all over the world. Since you can see the source code, you might even be able to fix it yourself. In contrast, if something goes wrong with commercial software, you have to wait for the programmers at the company to fix it (and speaking with them is probably out of the question), and wait for a new version to be released.

We find that Zelig makes students and colleagues more amenable to using R, since the startup costs are lower, and since the manual and software are relatively self-contained. This manual even includes an appendix devoted to the basics of advanced R programming, although you will not need it to run most procedures in Zelig. A large and growing fraction of the world's quantitative methodologists and statisticians are moving to R, and the base of programs available for R is quickly surpassing all alternatives. In addition to built-in functions, R is a complete programming language, which allows you to design new functions to suit your needs. R has the dual advantage that you do not need to understand how to program to use it, but if it turns out that you want to do something more complicated, you do not need to learn another program. In addition, methodologists all over the world add new functions all the time, so if the function you need wasn't there yesterday, it may be available today.

1.2 Getting Help

You may find documentation for Zelig on-line (and hence must be on-line to access it). If you are unable to connect to the Internet, we recommend that you print the pdf version of this document for your reference.

If you are on-line, you may access comprehensive help files for Zelig commands and for each of the models. For example, load the Zelig library and then type at the R prompt:

```
> help.zelig(command)                # For help with all zelig commands.
> help.zelig(logit)                  # For help with the logit model.
```

In addition, `help.zelig()` searches the manual pages for R in addition to the Zelig specific pages. On certain rare occasions, the name of the help topic in Zelig and in R are identical. In these cases, `help.zelig()` will return the Zelig help page by default. If you wish to access the R help page, you should use `help(topic)`.

In addition, built-in examples with sample data and plots are available for each model. For example, type `demo(logit)` to view the demo for the logit model. Commented code for each model is available under the examples section of each model reference page.

Please direct inquiries and problems about Zelig to our listserv at zelig@lists.gking.harvard.edu. We suggest you subscribe to this mailing list while learning and using Zelig: go to `http://lists.hmdc.harvard.edu/index.cgi?info=zelig`. (You can choose to receive email in digest form, so that you will never receive more than one message per day.) You can also browse or search our archive of previous messages before posting your query.

1.3 How to Cite Zelig

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.
>
> Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

To refer to a particular Zelig model, please refer to the "how to cite" portion at the end of each model documentation section.

[2] As specified in the GNU General License v. 2 `http://www.gnu.org/copyleft`.

Chapter 2

Installing Zelig

2.1 Introduction

Zelig's installation procedure is straightforward, though the package itself is not standalone, and requires the installation of R version 2.13 (or greater). That is, because Zelig is written in the R statistical programming language, it cannot be installed without support for the R programming language. As a result of this, installing Zelig and Zelig-compliant packages can be divided into three tasks:

1. Download and Install R,
2. Install Zelig, and
3. Install Optional Zelig Add-ons

The following guide is intended to quickly and easily explain each of these steps.

2.2 Requirements for Installation

The Zelig software suite has only two requirements:

1. **R version 2.13+**, which can be downloaded at `http://r-project.org/`
2. A major operating system, either:
   - Mac OS X 10.4+,
   - Windows or
   - Linux

Installation instructions for R can be found on the R-project website. Simply visit the download page, and select any mirror link, though this one is recommended:
http://cran.opensourceresources.org/

2.3 Installing R

Installing R is typically straightforward, regardless of which operating system is being used. Several useful documents exist on CRAN (The Comprehensive R Archive Network) for explanation and troubleshooting of R installation. These documents can be found on any CRAN mirror. Specifically, the complete guide to install R can be found at:
http://cran.r-project.org/doc/manuals/R-admin.html
This document contains specific documents for installing R on Mac , Windows , and Unix-alike systems.

2.4 Easy Installation

Once R version 2.13 or greater has been installed on the client's machine, setting up Zelig is a breeze. R has built-in facilities for managing the installation of statistical packages. This provides a simple mechanism for installing Zelig, regardless of the operating system that is being used.

To install Zelig, as well as its add-on packages, simply:

1. **Install R version 2.13 or greater**. Download R from the R project's website, which can be found at http://cran.opensourceresources.org/
2. **Launch R**. Once R is installed, this program can be found wherever the computer stores applications (e.g. "Program Files" on a Windows machine)
3. At the R command-prompt, type:

   ```
   source("http://gking.harvard.edu/zelig/install.R")
   ```

   This launches pre-written install script, which directs R to download all the appropriate statistical packages associated with the Zelig software suite.

2.5 Advanced Installation

For users familiar with R and Zelig, it may be useful to selectively install packages. In order to do this, users simply need to use the `install.packages` function built into R's functionality.

2.5.1 Install Zelig without Additional Packages

This installation procedure will install Zelig without any add-on packages. That is, Zelig will only download files necessarily for developing *new* Zelig packages and basic generalized linear model regressions - logit, gamma, gaussian, etc.

To install this "core" package, simply type the following from the R command prompt:

```
install.packages(
                 "Zelig",
                 repos = "http://r.iq.harvard.edu",
                 type  = "source"
                 )
```

2.5.2 Install Add-on Packages

In addition to Zelig's core package, which exclusively contains simple regression models and a Developers' API for producing novel R packages, a myriad of Zelig add-on packages are available. These packages supplement Zelig's features, and add specialized, advanced models to Zelig.

List of Available Packages

These add-on packages include:

- `bivariate.zelig`: Bivariate Models for Logit and Probit Regressions
- `gam.zelig`: Generalized Additive Models for Logit, Gaussian, Poisson and Probit Regressions
- `gee.zelig`: Generalized Estimating Equation Models for Gamma, Logit, Gaussian, Poisson and Probit Regressions

- `mixed.zelig`: Mixed Effect Models (Multi-level) for Gamma, Logit, Least-squares, Poisson and Probit Regressions
- `multinomial.zelig`: Multinomial Models Logit and Probit Regressions
- `ordinal.zelig`: Ordinal Models for Logit and Probit Regressions
- `survey.zelig`: Survey-weighted Models for Gamma, Logit, Normal, Poisson and Probit Regressions

Using `install.packages`

To download any of these packages independently, simply type the following from an R command prompt:

```
install.packages(
                 "MODEL NAME",
                 repos = "http://r.iq.harvard.edu/",
                 type  = "source"
                 )
```

Where `"MODEL NAME"` is replaced with the title of the Add-on packages in the above itemized list. For example, to download "Generalized Estimating Equation Models...", note that its package name is `gee.zelig`, and type from the R command prompt:

```
install.packages(
                 "gee.zelig",
                 repos = "http://r.iq.harvard.edu/",
                 type  = "source"
                 )
```

2.6 Post-Installation

Barring any installation errors, Zelig and any add-on packages that were manually installed, should now be available from an R-session. Simply type from an R command prompt:

```
library(Zelig)
?Zelig
```

To begin interacting and using the Zelig software package. Additionally, demo files can be listed via the command:

```
demo(package="Zelig")
```

Part II

Basic R Commands

Chapter 3

Programming Statements

This chapter introduces the main programming commands. These include functions, if-else statements, for-loops, and special procedures for managing the inputs to statistical models.

3.1 Functions

Functions are either built-in or user-defined sets of encapsulated commands which may take any number of arguments. Preface a function with the `function` statement and use the `<-` operator to assign functions to objects in your workspace.

You may use functions to run the same procedure on different objects in your workspace. For example,

```
check <- function(p, q) {
 result <- (p - q)/q
 result
 }
```

is a simple function with arguments `p` and `q` which calculates the difference between the ith elements of the vector `p` and the ith element of the vector `q` as a proportion of the ith element of `q`, and returns the resulting vector. For example, `check(p = 10, q = 2)` returns 4. You may omit the descriptors as long as you keep the arguments in the correct order: `check(10, 2)` also returns 4. You may also use other objects as inputs to the function. If `again = 10` and `really = 2`, then `check(p = again, q = really)` and `check(again, really)` also returns 4.

Because functions run commands as a set, you should make sure that each command in your function works by testing each line of the function at the R prompt.

3.2 If-Statements

Use `if` (and optionally, `else`) to control the flow of R functions. For example, let `x` and `y` be scalar numerical values:

```
if (x == y) {               # If the logical statement in the ()'s is true,
  x <- NA                   #  then `x' is changed to `NA' (missing value).
}
else {                      # The `else' statement tells R what to do if
  x <- x^2                  #  the if-statement is false.
}
```

As with a function, use `{` and `}` to define the set of commands associated with each if and else statement. (If you include if statements inside functions, you may have multiple sets of nested curly braces.)

3.3 For-Loops

Use `for` to repeat (loop) operations. Avoiding loops by using matrix or vector commands is usually faster and more elegant, but loops are sometimes necessary to assign values. If you are using a loop to assign values to a data structure, you must first initialize an empty data structure to hold the values you are assigning.

Select a data structure compatible with the type of output your loop will generate. If your loop generates a scalar, store it in a vector (with the ith value in the vector corresponding to the the ith run of the loop). If your loop generates vector output, store them as rows (or columns) in a matrix, where the ith row (or column) corresponds to the ith iteration of the loop. If your output consists of matrices, stack them into an array. For list output (such as regression output) or output that changes dimensions in each iteration, use a list. To initialize these data structures, use:

```
> x <- vector()                          # An empty vector of any length.
> x <- list()                            # An empty list of any length.
```

The `vector()` and `list()` commands create a vector or list of any length, such that assigning `x[5] <- 15` automatically creates a vector with 5 elements, the first four of which are empty values (`NA`). In contrast, the `matrix()` and `array()` commands create data structures that are restricted to their original dimensions.

```
> x <- matrix(nrow = 5, ncol = 2)  # A matrix with 5 rows and 2 columns.
> x <- array(dim = c(5,2,3))       # A 3D array of 3 stacked 5 by 2 matrices.
```

If you attempt to assign a value at $(100, 200, 20)$ to either of these data structures, R will return an error message ("subscript is out of bounds"). R does not automatically extend the dimensions of either a matrix or an array to accommodate additional values.

Example 1: Creating a vector with a logical statement

```
x <- array()              # Initializes an empty data structure.
for (i in 1:10) {         # Loops through every value from 1 to 10, replacing
  if (is.integer(i/2)) { #  the even values in `x' with i+5.
    x[i] <- i + 5
  }
}                         # Enclose multiple commands in {}.
```

You may use `for()` inside or outside of functions.

Example 2: Creating dummy variables by hand You may also use a loop to create a matrix of dummy variables to append to a data frame. For example, to generate fixed effects for each state, let's say that you have `mydata` which contains `y`, `x1`, `x2`, `x3`, and `state`, with `state` a character variable with 50 unique values. There are three ways to create dummy variables: 1) with a built-in R command; 2) with one loop; or 3) with 2 for loops.

1. R will create dummy variables on the fly from a single variable with distinct values.

   ```
   > z.out <- zelig(y ~ x1 + x2 + x3 + as.factor(state),
                    data = mydata, model = "ls")
   ```

   This method returns $k - 1$ indicators for k states.

2. Alternatively, you can use a loop to create dummy variables by hand. There are two ways to do this, but both start with the same initial commands. Using vector commands, first create an index of for the states, and initialize a matrix to hold the dummy variables:

   ```
   idx <- sort(unique(mydata$state))
   dummy <- matrix(NA, nrow = nrow(mydata), ncol = length(idx))
   ```

Now choose between the two methods.

(a) The first method is computationally inefficient, but more intuitive for users not accustomed to vector operations. The first loop uses `i` as in index to loop through all the rows, and the second loop uses `j` to loop through all 50 values in the vector `idx`, which correspond to columns 1 through 50 in the matrix `dummy`.

```
for (i in 1:nrow(mydata)) {
  for (j in 1:length(idx)) {
    if (mydata$state[i,j] == idx[j]) {
      dummy[i,j] <- 1
    }
    else {
      dummy[i,j] <- 0
    }
  }
}
```

Then add the new matrix of dummy variables to your data frame:

```
names(dummy) <- idx
mydata <- cbind(mydata, dummy)
```

(b) As you become more comfortable with vector operations, you can replace the double loop procedure above with one loop:

```
for (j in 1:length(idx)) {
  dummy[,j] <- as.integer(mydata$state == idx[j])
}
```

The single loop procedure evaluates each element in `idx` against the vector `mydata$state`. This creates a vector of n `TRUE`/`FALSE` observations, which you may transform to `1`'s and `0`'s using `as.integer()`. Assign the resulting vector to the appropriate column in `dummy`. Combine the `dummy` matrix with the data frame as above to complete the procedure.

Example 3: Weighted regression with subsets Selecting the `by` option in `zelig()` partitions the data frame and then automatically loops the specified model through each partition. Suppose that `mydata` is a data frame with variables `y`, `x1`, `x2`, `x3`, and `state`, with `state` a factor variable with 50 unique values. Let's say that you would like to run a weighted regression where each observation is weighted by the inverse of the standard error on `x1`, estimated for that observation's state. In other words, we need to first estimate the model for each of the 50 states, calculate 1 / SE($\texttt{x1}_j$) for each state $j = 1, \ldots, 50$, and then assign these weights to each observation in `mydata`.

- Estimate the model separate for each state using the `by` option in `zelig()`:

```
z.out <- zelig(y ~ x1 + x2 + x3, by = "state", data = mydata, model = "ls")
```

Now `z.out` is a list of 50 regression outputs.

- Extract the standard error on `x1` for each of the state level regressions.

```
se <- array()                          # Initalize the empty data structure.
for (i in 1:50) {                      # vcov() creates the variance matrix
  se[i] <- sqrt(vcov(z.out[[i]])[2,2]) # Since we have an intercept, the 2nd
}                                      # diagonal value corresponds to x1.
```

- Create the vector of weights.

```
wts <- 1 / se
```

  This vector `wts` has 50 values that correspond to the 50 sets of state-level regression output in `z.out`.

- To assign the vector of weights to each observation, we need to match each observation's state designation to the appropriate state. For simplicity, assume that the states are numbered 1 through 50.

```
mydata$w <- NA            # Initalizing the empty variable
for (i in 1:50) {
  mydata$w[mydata$state == i] <- wts[i]
}
```

  We use `mydata$state` as the index (inside the square brackets) to assign values to `mydata$w`. Thus, whenever state equals 5 for an observation, the loop assigns the fifth value in the vector `wts` to the variable `w` in `mydata`. If we had 500 observations in `mydata`, we could use this method to match each of the 500 observations to the appropriate `wts`.

  If the states are character strings instead of integers, we can use a slightly more complex version

```
mydata$w <- NA
idx <- sort(unique(mydata$state))
for (i in 1:length(idx) {
  mydata$w[mydata$state == idx[i]] <- wts[i]
}
```

- Now we can run our weighted regression:

```
z.wtd <- zelig(y ~ x1 + x2 + x3, weights = w, data = mydata,
               model = "ls")
```

Chapter 4

R Syntax

4.1 Command Syntax

Once R is installed, you only need to know a few basic elements to get started. It's important to remember that R, like any spoken language, has rules for proper syntax. Unlike English, however, the rules for intelligible R are small in number and quite precise (see Section 4.1.2).

4.1.1 Getting Started

1. To start R under Linux or Unix, type R at the terminal prompt or M-x R under ESS.
2. The R prompt is `>`.
3. Type commands and hit enter to execute. (No additional characters, such as semicolons or commas, are necessary at the end of lines.)
4. To quit from R, type `q()` and press enter.
5. The `#` character makes R ignore the rest of the line, and is used in this document to comment R code.
6. We highly recommend that you make a separate working directory or folder for each project.
7. Each R session has a workspace, or working memory, to store the *objects* that you create or input. These objects may be:
   (a) *values*, which include numerical, integer, character, and logical values;
   (b) *data structures* made up of variables (vectors), matrices, and data frames; or
   (c) *functions* that perform the desired tasks on user-specified values or data structures.

After starting R, you may at any time use Zelig's built-in help function to access on-line help for any command. To see help for all Zelig commands, type `help.zelig(command)`, which will take you to the help page for all Zelig commands. For help with a specific Zelig or R command substitute the name of the command for the generic `command`. For example, type `help.zelig(logit)` to view help for the logit model.

4.1.2 Details

.

Zelig uses the syntax of R, which has several essential elements:

1. R is case sensitive. `Zelig`, the package or library, is not the same as `zelig`, the command.

2. R functions accept user-defined arguments: while some arguments are required, other optional arguments modify the function's default behavior. Enclose arguments in parentheses and separate multiple arguments with commas. For example, `print(x)` or `print(x, digits = 2)` prints the contents of the object `x` using the default number of digits or rounds to two digits to the right of the decimal point, respectively. You may nest commands as long as each has its own set of parentheses: `log(sqrt(5))` takes the square root of 5 and then takes the natural log.

3. The `<-` operator takes the output of the function on the right and saves them in the named object on the left. For example, `z.out <- zelig(...)` stores the output from `zelig()` as the object `z.out` in your working memory. You may use `z.out` as an argument in other functions, view the output by typing `z.out` at the R prompt, or save `z.out` to a file using the procedures described in Section 4.2.3.

4. You may name your objects anything, within a few constraints:

   - You may only use letters (in upper or lower case) and periods to punctuate your variable names.
   - You may *not* use any special characters (aside from the period) or spaces to punctuate your variable names.
   - Names cannot begin with numbers. For example, R will not let you save an object as `1997.election` but will let you save `election.1997`.

5. Use the `names()` command to see the contents of R objects, and the `$` operator to extract elements from R objects. For example:

```
# Run least squares regression and save the output in working memory:
> z.out <- zelig(y ~ x1 + x2, model = "ls", data = mydata)
# See what's in the R object:
> names(z.out)
 [1] "coefficients"  "residuals"  "effects"  "rank"
# Extract and display the coefficients in z.out:
> z.out$coefficients
```

6. All objects have a class designation which tells R how to treat it in subsequent commands. An object's class is generated by the function or mathematical operation that created it.

7. To see a list of all objects in your current workspace, type: `ls()`. You can remove an object permanently from memory by typing `remove(goo)` (which deletes the object `goo`), or remove all the objects with `remove(list = ls())`.

8. To run commands in a batch, use a text editor (such as the Windows R script editor or emacs) to compose your R commands, and save the file with a `.R` file extension in your working directory. To run the file, type `source("Code.R")` at the R prompt.

If you encounter a syntax error, check your spelling, case, parentheses, and commas. These are the most common syntax errors, and are easy to detect and correct with a little practice. If you encounter a syntax error in batch mode, R will tell you the line on which the syntax error occurred.

4.2 Data Sets

4.2.1 Data Structures

Zelig uses only three of R's many data structures:

1. A **variable** is a one-dimensional vector of length n.

2. A **data frame** is a rectangular matrix with n rows and k columns. Each column represents a variable and each row an observation. Each variable may have a different class. (See Section 4.3.1 for a list of classes.) You may refer to specific variables from a data frame using, for example, `data$variable`.

3. A **list** is a combination of different data structures. For example, `z.out` contains both `coefficients` (a vector) and `data` (a data frame). Use `names()` to view the elements available within a list, and the `$` operator to refer to an element in a list.

4.2.2 Loading Data

Datasets in Zelig are stored in "data frames." In this section, we explain the standard ways to load data from disk into memory, how to handle special cases, and how to verify that the data you loaded is what you think it is.

Standard Ways to Load Data

Make sure that the data file is saved in your working directory. You can check to see what your working directory is by starting R, and typing `getwd()`. If you wish to use a different directory as your starting directory, use `setwd("dirpath")`, where `"dirpath"` is the full directory path of the directory you would like to use as your working directory.

After setting your working directory, load data using one of the following methods:

1. If your dataset is in a **tab- or space-delimited .txt file**, use `read.table("mydata.txt")`

2. If your dataset is **a comma separated table**, use `read.csv("mydata.csv")`.

3. To import **SPSS, Stata, and other data files**, use the foreign package, which automatically preserves field characteristics for each variable. Thus, variables classed as dates in Stata are automatically translated into values in the date class for R. For example:

```
> library(foreign)                          # Load the foreign package.
> stata.data <- read.dta("mydata.dta")      # For Stata data.
> spss.data <- read.spss("mydata.sav", to.data.frame = TRUE) # For SPSS.
```

4. To load data in R format, use `load("mydata.RData")`.

5. For sample data sets included with R packages such as Zelig, you may use the `data()` command, which is a shortcut for loading data from the sample data directories. Because the locations of these directories vary by installation, it is extremely difficult to locate sample data sets and use one of the three preceding methods; `data()` searches all of the currently used packages and loads sample data automatically. For example:

```
> library(Zelig)                            # Loads the Zelig library.
> data(turnout)                             # Loads the turnout data.
```

Special Cases When Loading Data

These procedures apply to any of the above `read` commands:

1. If your file uses the **first row to identify variable names**, you should use the option `header = TRUE` to import those field names. For example,

```
> read.csv("mydata.csv", header = TRUE)
```

will read the words in the first row as the variable names and the subsequent rows (each with the same number of values as the first) as observations for each of those variables. If you have additional characters on the last line of the file or fewer values in one of the rows, you need to edit the file before attempting to read the data.

2. The R missing value code is `NA`. If this value is in your data, R will recognize your missing values as such. If you have instead used a place-holder value (such as -9) to represent missing data, you need to tell R this on loading the data:

   ```
   > read.table("mydata.tab", header = TRUE, na.strings = "-9")
   ```

   Note: You must enclose your place holder values in quotes.

3. Unlike Windows, the file extension in R does not determine the default method for dealing with the file. For example, if your data is tab-delimited, but saved as a `.sav` file, `read.table("mydata.sav")` will load your data into R.

Verifying You Loaded The Data Correctly

Whichever method you use, try the `names()`, `dim()`, and `summary()` commands to verify that the data was properly loaded. For example,

```
> data <- read.csv("mydata.csv", header = TRUE)             # Read the data.
> dim(data)                     # Displays the dimensions of the data frame
[1] 16000  8                    #  in rows then columns.
> data[1:10,]                   # Display rows 1-10 and all columns.
> names(data)                   # Check the variable names.
[1] "V1" "V2" "V3"              # These values indicate that the variables
                                #  weren't named, and took default values.
> names(data) <- c("income", "educate", "year")     # Assign variable names.
> summary(data)                 # Returning a summary for each variable.
```

In this case, the `summary()` command will return the maximum, minimum, mean, median, first and third quartiles, as well as the number of missing values for each variable.

4.2.3 Saving Data

Use `save()` to write data or any object to a file in your working directory. For example,

```
> save(mydata, file = "mydata.RData")     # Saves `mydata' to `mydata.RData'
                                          #  in your working directory.
> save.image()                            # Saves your entire workspace to
                                          #  the default `.RData' file.
```

R will also prompt you to save your workspace when you use the `q()` command to quit. When you start R again, it will load the previously saved workspace. Restarting R will not, however, load previously used packages. You must remember to load Zelig at the beginning of every R session.

Alternatively, you can recall individually saved objects from .RData files using the `load()` command. For example,

```
> load("mydata.RData")
```

loads the objects saved in the `mydata.RData` file. You may save a data frame, a data frame and associated functions, or other R objects to file.

4.3 Variables

4.3.1 Classes of Variables

R variables come in several types. Certain Zelig models require dependent variables of a certain class of variable. (These are documented under the manual pages for each model.) Use `class(variable)` to determine the class of a variable or `class(data$variable)` for a variable within a data frame.

Types of Variables

For all types of variable (vectors), you may use the `c()` command to "concatenate" elements into a vector, the `:` operator to generate a sequence of integer values, the `seq()` command to generate a sequence of non-integer values, or the `rep()` function to repeat a value to a specified length. In addition, you may use the `<-` operator to save variables (or any other objects) to the workspace. For example:

```
> logic <- c(TRUE, FALSE, TRUE, TRUE, TRUE) # Creates `logic' (5 T/F values).
> var1 <- 10:20                             # All integers between 10 and 20.
> var2 <- seq(from = 5, to = 10, by = 0.5)  # Sequence from 5 to 10 by
                                            #  intervals of 0.5.
> var3 <- rep(NA, length = 20)              # 20 `NA' values.
> var4 <- c(rep(1, 15), rep(0, 15))         # 15 `1's followed by 15 `0's.
```

For the `seq()` command, you may alternatively specify `length` instead of `by` to create a variable with a specific number (denoted by the `length` argument) of evenly spaced elements.

1. **Numeric** variables are real numbers and the default variable class for most dataset values. You can perform any type of math or logical operation on numeric values. If `var1` and `var2` are numeric variables, we can compute

   ```
   > var3 <- log(var2) - 2*var1      # Create `var3' using math operations.
   ```

   `Inf` (infinity), `-Inf` (negative infinity), `NA` (missing value), and `NaN` (not a number) are special numeric values on which most math operations will fail. (Logical operations will work, however.) Use `as.numeric()` to transform variables into numeric variables. Integers are a special class of numeric variable.

2. **Logical** variables contain values of either `TRUE` or `FALSE`. R supports the following logical operators: `==`, exactly equals; `>`, greater than; `<`, less than; `>=`, greater than or equals; `<=`, less than or equals; and `!=`, not equals. The `=` symbol is *not* a logical operator. Refer to Section 4.3.2 for more detail on logical operators. If `var1` and `var2` both have n observations, commands such as

   ```
   > var3 <- var1 < var2
   > var3 <- var1 == var2
   ```

   create n `TRUE`/`FALSE` observations such that the ith observation in `var3` evaluates whether the logical statement is true for the ith value of `var1` with respect to the ith value of `var2`. Logical variables should usually be converted to integer values prior to analysis; use the `as.integer()` command.

3. **Character** variables are sets of text strings. Note that text strings are always enclosed in quotes to denote that the string is a value, not an object in the workspace or an argument for a function (neither of which take quotes). Variables of class character are not normally used in data analysis, but used as descriptive fields. If a character variable is used in a statistical operation, it must first be transformed into a factored variable.

4. **Factor** variables may contain values consisting of either integers or character strings. Use `factor()` or `as.factor()` to convert character or integer variables into factor variables. Factor variables separate unique values into levels. These levels may either be ordered or unordered. In practice, this means that including a factor variable among the explanatory variables is equivalent to creating dummy variables for each level. In addition, some models (ordinal logit, ordinal probit, and multinomial logit), require that the dependent variable be a factor variable.

4.3.2 Recoding Variables

Researchers spend a significant amount of time cleaning and recoding data prior to beginning their analyses. R has several procedures to facilitate the process.

Extracting, Replacing, and Generating New Variables

While it is not difficult to recode variables, the process is prone to human error. Thus, we recommend that before altering the data, you save your existing data frame using the procedures described in Section 4.2.3, that you only recode one variable at a time, and that you recode the variable outside the data frame and then return it to the data frame.

To extract the variable you wish to recode, type:

```
> var <- data$var1               # Copies `var1' from `data', creating `var'.
```

Do *not* sort the extracted variable or delete observations from it. If you do, the ith observation in `var` will no longer match the ith observation in `data`.

To replace the variable or generate a new variable in the data frame, type:

```
> data$var1 <- var                # Replace `var1' in `data' with `var'.
> data$new.var <- var             # Generate `new.var' in `data' using `var'.
```

To remove a variable from a data frame (rather than replacing one variable with another):

```
> data$var1 <- NULL
```

Logical Operators

R has an intuitive method for recoding variables, which relies on logical operators that return statements of `TRUE` and `FALSE`. A mathematical operator (such as `==`, `!=`, `>`, `>=` `<`, and `<=`) takes two objects of equal dimensions (scalars, vectors of the same length, matrices with the same number of rows and columns, or similarly dimensioned arrays) and compares every element in the first object to its counterpart in the second object.

- `==`: checks that one variable "exactly equals" another in a list-wise manner. For example:

  ```
  > x <- c(1, 2, 3, 4, 5)               # Creates the object `x'.
  > y <- c(2, 3, 3, 5, 1)               # Creates the object `y'.
  > x == y                              # Only the 3rd `x' exactly equals
    [1] FALSE FALSE  TRUE FALSE FALSE   #  its counterpart in `y'.
  ```

  (The `=` symbol is *not* a logical operator.)

- `!=`: checks that one variable does not equal the other in a list-wise manner. Continuing the example:

  ```
  > x != y
    [1]  TRUE  TRUE FALSE  TRUE  TRUE
  ```

- `>` (`>=`): checks whether each element in the left-hand object is greater than (or equal to) every element in the right-hand object. Continuing the example from above:

```
> x > y                                     # Only the 5th `x' is greater
  [1] FALSE FALSE FALSE FALSE  TRUE         #  than its counterpart in `y'.
> x >= y                                    # The 3rd `x' is equal to the
  [1] FALSE FALSE  TRUE FALSE  TRUE         #  3rd `y' and becomes TRUE.
```

- `<` (`<=`): checks whether each element in the left-hand object is less than (or equal to) every object in the right-hand object. Continuing the example from above:

```
> x < y                             # The elements 1, 2, and 4 of `x' are
[1]  TRUE  TRUE FALSE  TRUE FALSE #  less than their counterparts in `y'.
> x <= y                            # The 3rd `x' is equal to the 3rd `y'
[1]  TRUE  TRUE  TRUE  TRUE FALSE #  and becomes TRUE.
```

For two vectors of five elements, the mathematical operators compare the first element in `x` to the first element in `y`, the second to the second and so forth. Thus, a mathematical comparison of `x` and `y` returns a vector of five `TRUE`/`FALSE` statements. Similarly, for two matrices with 3 rows and 20 columns each, the mathematical operators will return a 3×20 matrix of logical values.

There are additional logical operators which allow you to combine and compare logical statements:

- `&`: is the logical equivalent of "and", and evaluates one array of logical statements against another in a list-wise manner, returning a `TRUE` only if both are true in the same location. For example:

```
> a <- matrix(c(1:12), nrow = 3, ncol = 4)      # Creates a matrix `a'.
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> b <- matrix(c(12:1), nrow = 3, ncol = 4)      # Creates a matrix `b'.
> b
     [,1] [,2] [,3] [,4]
[1,]   12    9    6    3
[2,]   11    8    5    2
[3,]   10    7    4    1
> v1 <- a > 3                   # Creates the matrix `v1' (T/F values).
> v2 <- b > 3                   # Creates the matrix `v2' (T/F values).
> v1 & v2                       # Checks if the (i,j) value in `v1' and
      [,1] [,2] [,3]  [,4]      #  `v2' are both TRUE.  Because columns
[1,] FALSE TRUE TRUE FALSE      #  2-4 of `v1' are TRUE, and columns 1-3
[2,] FALSE TRUE TRUE FALSE      #  of `var2' are TRUE, columns 2-3 are
[3,] FALSE TRUE TRUE FALSE      #  TRUE here.
> (a > 3) & (b > 3)             # The same, in one step.
```

  For more complex comparisons, parentheses may be necessary to delimit logical statements.

- `|`: is the logical equivalent of "or", and evaluates in a list-wise manner whether either of the values are `TRUE`. Continuing the example from above:

```
> (a < 3) | (b < 3)               # (1,1) and (2,1) in `a' are less
      [,1]  [,2]  [,3]  [,4]      #  than 3, and (2,4) and (3,4) in
```

```
     [1,]  TRUE FALSE FALSE FALSE       #  `b' are less than 3; | returns
     [2,]  TRUE FALSE FALSE  TRUE       #  a matrix with `TRUE' in (1,1),
     [3,] FALSE FALSE FALSE  TRUE       #  (2,1), (2,4), and (3,4).
```

The && (if and only if) and || (either or) operators are used to control the command flow within functions. The && operator returns a TRUE only if every element in the comparison statement is true; the || operator returns a TRUE if any of the elements are true. Unlike the & and | operators, which return arrays of logical values, the && and || operators return only one logical statement irrespective of the dimensions of the objects under consideration. Hence, && and || are logical operators which are *not* appropriate for recoding variables.

Coding and Recoding Variables

R uses vectors of logical statements to indicate how a variable should be coded or recoded. For example, to create a new variable var3 equal to 1 if var1 < var2 and 0 otherwise:

```
> var3 <- var1 < var2               # Creates a vector of n T/F observations.
> var3 <- as.integer(var3)          # Replaces the T/F values in `var3' with
                                    #  1's for TRUE and 0's for FALSE.
> var3 <- as.integer(var1 < var2)   # Combine the two steps above into one.
```

In addition to generating a vector of dummy variables, you can also refer to specific values using logical operators defined in Section 4.3.2. For example:

```
> v1 <- var1 == 5                   # Creates a vector of T/F statements.
> var1[v1] <- 4                     # For every TRUE in `v1', replaces the
                                    #  value in `var1' with a 4.
> var1[var1 == 5] <- 4              # The same, in one step.
```

The index (inside the square brackets) can be created with reference to other variables. For example,

```
> var1[var2 == var3] <- 1
```

replaces the ith value in var1 with a 1 when the ith value in var2 equals the ith value in var3. If you use = in place of ==, however, you will replace all the values in var1 with 1's because = is another way to assign variables. Thus, the statement var2 = var3 is of course true.

Finally, you may also replace any (character, numerical, or logical) values with special values (most commonly, NA).

```
> var1[var1 == "don't know"] <- NA   # Replaces all "don't know"'s with NA's.
```

After recoding the var1 replace the old data$var1 with the recoded var1: data$var1 <- var1. You may combine the recoding and replacement procedures into one step. For example:

```
> data$var1[data$var1 =< 0] <- -1
```

Alternatively, rather than recoding just specific values in variables, you may calculate new variables from existing variables. For example,

```
> var3 <- var1 + 2 * var2
> var3 <- log(var1)
```

After generating the new variables, use the assignment mechanism <- to insert the new variable into the data frame.

In addition to generating vectors of dummy variables, you may transform a vector into a matrix of dummy indicator variables. For example, see Section 3.3 to transform a vector of k unique values (with n observations in the complete vector) into a $n \times k$ matrix.

Missing Data

To deal with missing values in some of your variables:

1. You may generate multiply imputed datasets using Amelia (or other programs).

2. You may omit missing values. Zelig models automatically apply list-wise deletion, so no action is required to run a model. To obtain the total number of observations or produce other summary statistics using the analytic dataset, you may manually omit incomplete observations. To do so, first create a data frame containing only the variables in your analysis. For example:

   ```
   > new.data <- cbind(data$dep.var, data$var1, data$var2, data$var3)
   ```

   The `cbind()` command "column binds" variables into a data frame. (A similar command `rbind()` "row binds" observations with the same number of variables into a data frame.) To omit missing values from this new data frame:

   ```
   > new.data <- na.omit(new.data)
   ```

   If you perform `na.omit()` on the full data frame, you risk deleting observations that are fully observed in your experimental variables, but missing values in other variables. Creating a new data frame containing only your experimental variables usually increases the number of observations retained after `na.omit()`.

Chapter 5

R Objects

In R, objects can have one or more classes, consisting of the class of the scalar value and the class of the data structure holding the scalar value. Use the `is()` command to determine what an object *is*. If you are already familiar with R objects, you may skip to Section 4.2.2 for loading data.

5.1 Scalar Values

R uses several classes of scalar values, from which it constructs larger data structures. R is highly class-dependent: certain operations will only work on certain types of values or certain types of data structures. We list the three basic types of scalar values here for your reference:

1. **Numeric** is the default value type for most numbers. An `integer` is a subset of the `numeric` class, and may be used as a `numeric` value. You can perform any type of math or logical operation on numeric values, including:

```
> log(3 * 4 * (2 + pi))          # Note that pi is a built-in constant,
   [1] 4.122270                  #   and log() the natural log function.
> 2 > 3                         # Basic logical operations, including >,
   [1] FALSE                     #   <, >= (greater than or equals),
                                 #   <= (less than or equals), == (exactly
                                 #   equals), and != (not equals).
> 3 >= 2 && 100 == 1000/10      # Advanced logical operations, including
   [1] TRUE                      #   & (and), && (if and only if), | (or),
                                 #   and || (either or).
```

   Note that `Inf` (infinity), `-Inf` (negative infinity), `NA` (missing value), and `NaN` (not a number) are special numeric values on which most math operations will fail. (Logical operations will work, however.)

2. **Logical** operations create logical values of either `TRUE` or `FALSE`. To convert logical values to numerical values, use the `as.integer()` command:

```
> as.integer(TRUE)
   [1] 1
> as.integer(FALSE)
   [1] 0
```

3. **Character** values are text strings. For example,

```
> text <- "supercalafragilisticxpaladocious"
> text
[1] "supercalafragilisticxpaladocious"
```

assigns the text string on the right-hand side of the `<-` to the named object in your workspace. Text strings are primarily used with data frames, described in the next section. R always returns character strings in quotes.

5.2 Data Structures

5.2.1 Arrays

Arrays are data structures that consist of only one type of scalar value (e.g., a vector of character strings, or a matrix of numeric values). The most common versions, one-dimensional and two-dimensional arrays, are known as *vectors* and *matrices*, respectively.

Ways to create arrays

1. Common ways to create **vectors** (or one-dimensional arrays) include:

```
> a <- c(3, 7, 9, 11)    # Concatenates numeric values into a vector
> a <- c("a", "b", "c")  # Concatenates character strings into a vector
> a <- 1:5               # Creates a vector of integers from 1 to 5 inclusive
> a <- rep(1, times = 5) # Creates a vector of 5 repeated `1's
```

   To manipulate a vector:

```
> a[10]                # Extracts the 10th value from the vector `a'
> a[5] <- 3.14         # Inserts 3.14 as the 5th value in the vector `a'
> a[5:7] <- c(2, 4, 7) # Replaces the 5th through 7th values with 2, 4, and 7
```

   Unlike larger arrays, vectors can be extended without first creating another vector of the correct length. Hence,

```
> a <- c(4, 6, 8)
> a[5] <- 9       # Inserts a 9 in the 5th position of the vector,
                  #  automatically inserting an `NA' values position 4
```

2. A **factor vector** is a special type of vector that allows users to create j indicator variables in one vector, rather than using j dummy variables (as in Stata or SPSS). R creates this special class of vector from a pre-existing vector `x` using the `factor()` command, which separates `x` into levels based on the discrete values observed in `x`. These values may be either integer value or character strings. For example,

```
> x <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 9, 9, 9, 9)
> factor(x)
   [1] 1 1 1 1 1 2 2 2 2 9 9 9 9
   Levels: 1 2 9
```

   By default, `factor()` creates unordered factors, which are treated as discrete, rather than ordered, levels. Add the optional argument `ordered = TRUE` to order the factors in the vector:

```
> x <- c("like", "dislike", "hate", "like", "don't know", "like", "dislike")
> factor(x, levels = c("hate", "dislike", "like", "don't know"),
+        ordered = TRUE)
  [1] like    dislike    hate    like    don't know   like    dislike
Levels: hate < dislike < like < don't know
```

The `factor()` command orders the levels according to the order in the optional argument `levels`. If you omit the levels command, R will order the values as they occur in the vector. Thus, omitting the `levels` argument sorts the levels as `like < dislike < hate < don't know` in the example above. If you omit one or more of the levels in the list of levels, R returns levels values of `NA` for the missing level(s):

```
> factor(x, levels = c("hate", "dislike", "like"), ordered = TRUE)
  [1] like    dislike hate    like    <NA>    like    dislike
Levels: hate < dislike < like
```

Use factored vectors within data frames for plotting (see Section 7.1), to set the values of the explanatory variables using `setx`.

3. Build **matrices** (or two-dimensional arrays) from vectors (one-dimensional arrays). You can create a matrix in two ways:

   (a) From a vector: Use the command `matrix(vector, nrow =` k`, ncol =` n`)` to create a $k \times n$ matrix from the vector by filling in the columns from left to right. For example,

   ```
   > matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
         [,1] [,2] [,3]       # Note that when assigning a vector to a
   [1,]    1    3    5       #  matrix, none of the rows or columns
   [2,]    2    4    6       #  have names.
   ```

   (b) From two or more vectors of length k: Use `cbind()` to combine n vectors vertically to form a $k \times n$ matrix, or `rbind()` to combine n vectors horizontally to form a $n \times k$ matrix. For example:

   ```
   > x <- c(11, 12, 13)          # Creates a vector `x' of 3 values.
   > y <- c(55, 33, 12)          # Creates another vector `y' of 3 values.
   > rbind(x, y)                 # Creates a 2 x 3 matrix.  Note that row
        [,1] [,2] [,3]           #  1 is named x and row 2 is named y,
   x   11   12   13              #  according to the order in which the
   y   55   33   12              #  arguments were passed to rbind().
   > cbind(x, y)                 # Creates a 3 x 2 matrix.  Note that the
          x  y                   #  columns are named according to the
   [1,] 11 55                    #  order in which they were passed to
   [2,] 12 33                    #  cbind().
   [3,] 13 12
   ```

   R supports a variety of matrix functions, including: `det()`, which returns the matrix's determinant; `t()`, which transposes the matrix; `solve()`, which inverts the the matrix; and `%*%`, which multiplies two matricies. In addition, the `dim()` command returns the dimensions of your matrix. As with vectors, square brackets extract specific values from a matrix and the assignment mechanism `<-` replaces values. For example:

   ```
   > loo[,3]        # Extracts the third column of loo.
   > loo[1,]         # Extracts the first row of loo.
   > loo[1,3] <- 13          # Inserts 13 as the value for row 1, column 3.
   > loo[1,] <- c(2,2,3)     # Replaces the first row of loo.
   ```

If you encounter problems replacing rows or columns, make sure that the `dims()` of the vector matches the `dims()` of the matrix you are trying to replace.

4. An **n-dimensional array** is a set of stacked matrices of identical dimensions. For example, you may create a three dimensional array with dimensions (x, y, z) by stacking z matrices each with x rows and y columns.

```
> a <- matrix(8, 2, 3)       # Creates a 2 x 3 matrix populated with 8's.
> b <- matrix(9, 2, 3)       # Creates a 2 x 3 matrix populated with 9's.
> array(c(a, b), c(2, 3, 2)) # Creates a 2 x 3 x 2 array with the first
   , , 1                     #  level [,,1] populated with matrix a (8's),
                             #  and the second level [,,2] populated
        [,1] [,2] [,3]       #  with matrix b (9's).
   [1,]    8    8    8
   [2,]    8    8    8       # Use square brackets to extract values.  For
                             #  example, [1, 2, 2] extracts the second
   , , 2                     #  value in the first row of the second level.
                             # You may also use the <- operator to
        [,1] [,2] [,3]       #  replace values.
   [1,]    9    9    9
   [2,]    9    9    9
```

If an array is a one-dimensional vector or two-dimensional matrix, R will treat the array using the more specific method.

Three functions especially helpful for arrays:

- `is()` returns both the type of scalar value that populates the array, as well as the specific type of array (vector, matrix, or array more generally).

- `dims()` returns the size of an array, where

```
> dims(b)
 [1]  33  5
```

  indicates that the array is two-dimensional (a matrix), and has 33 rows and 5 columns.

- The single bracket `[ ]` indicates specific values in the array. Use commas to indicate the index of the specific values you would like to pull out or replace:

```
> dims(a)
 [1]  14
> a[10]       # Pull out the 10th value in the vector `a'
> dims(b)
 [1]  33  5
> b[1:12, ]   # Pull out the first 12 rows of `b'
> c[1, 2]     # Pull out the value in the first row, second column of `c'
> dims(d)
 [1]  1000  4  5
> d[ , 3, 1]  # Pulls out a vector of 1,000 values
```

5.2.2 Lists

Unlike arrays, which contain only one type of scalar value, lists are flexible data structures that can contain heterogeneous value types and heterogeneous data structures. Lists are so flexible that one list can contain another list. For example, the list `output` can contain `coef`, a vector of regression coefficients; `variance`, the variance-covariance matrix; and another list `terms` that describes the data using character strings. Use the `names()` function to view the named elements in a list, and to extract a named element, use

```
> names(output)
 [1] coefficients   variance   terms
> output$coefficients
```

For lists where the elements are not named, use double square brackets `[[ ]]` to extract elements:

```
> L[[4]]      # Extracts the 4th element from the list `L'
> L[[4]] <- b # Replaces the 4th element of the list `L' with a matrix `b'
```

Like vectors, lists are flexible data structures that can be extended without first creating another list of with the correct number of elements:

```
> L <- list()                        # Creates an empty list
> L$coefficients <- c(1, 4, 6, 8)  # Inserts a vector into the list, and
                                     #  names that vector `coefficients'
   #  within the list
> L[[4]] <- c(1, 4, 6, 8)          # Inserts the vector into the 4th position
                                     #  in the list.  If this list doesn't
                                     #  already have 4 elements, the empty
                                     #  elements will be `NULL' values
```

Alternatively, you can easily create a list using objects that already exist in your workspace:

```
> L <- list(coefficients = k, variance = v) # Where `k' is a vector and
                                            #   `v' is a matrix
```

5.2.3 Data Frames

A data frame (or data set) is a special type of list in which each variable is constrained to have the same number of observations. A data frame may contain variables of different types (numeric, integer, logical, character, and factor), so long as each variable has the same number of observations.

Thus, a data frame can use both matrix commands and list commands to manipulate variables and observations.

```
> dat[1:10,]         # Extracts observations 1-10 and all associated variables
> dat[dat$grp == 1,] # Extracts all observations that belong to group 1
> group <- dat$grp   # Saves the variable `grp' as a vector `group' in
                     #   the workspace, not in the data frame
> var4 <- dat[[4]]   # Saves the 4th variable as a `var4' in the workspace
```

For a comprehensive introduction to data frames and recoding data, see Section 4.2.2.

5.2.4 Identifying Objects and Data Structures

Each data structure has several *attributes* which describe it. Although these attributes are normally invisible to users (e.g., not printed to the screen when one types the name of the object), there are several helpful functions that display particular attributes:

- For arrays, `dims()` returns the size of each dimension.

- For arrays, `is()` returns the scalar value type and specific type of array (vector, matrix, array). For more complex data structures, `is()` returns the default methods (classes) for that object.
- For lists and data frames, `names()` returns the variable names, and `str()` returns the variable names and a short description of each element.

For almost all data types, you may use `summary()` to get summary statistics.

Part III

R Basics

Chapter 6

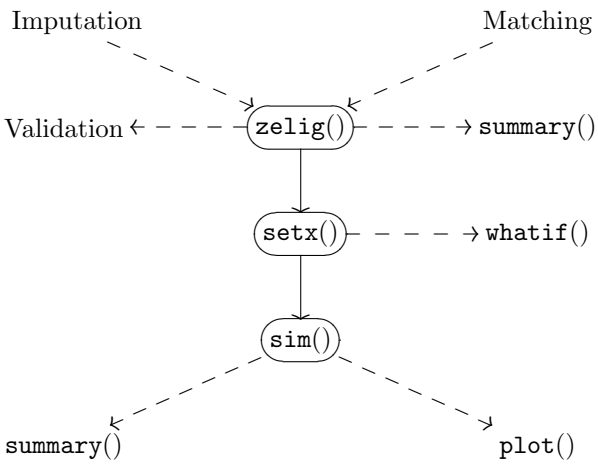
Zelig Commands

6.1 Zelig Commands

6.1.1 Quick Overview

For any statistical model, Zelig does its work with a combination of three commands.

Figure 6.1: Main Zelig commands (solid arrows) and some options (dashed arrows)



1. Use `zelig()` to run the chosen statistical model on a given data set, with a specific set of variables. For standard likelihood models, for example, this step estimates the coefficients, other model parameters, and a variance-covariance matrix. In addition, you may choose from a variety of options:

   - Pre-process data: Prior to calling `zelig()`, you may choose from a variety of data pre-processing commands (matching or multiple imputation, for example) to make your statistical inferences more accurate.
   - Summarize model: After calling `zelig()`, you may summarize the fitted model output using `summary()`.

- Validate model: After calling `zelig()`, you may choose to validate the fitted model. This can be done, for example, by using cross-validation procedures and diagnostics tools.

2. Use `setx()` to set each of the explanatory variables to chosen (actual or counterfactual) values in preparation for calculating quantities of interest. After calling `setx()`, you may use WhatIf to evaluate these choices by determining whether they involve interpolation (i.e., are inside the convex hull of the observed data) or extrapolation, as well as how far these counterfactuals are from the data. Counterfactuals chosen in `setx()` that involve extrapolation far from the data can generate considerably more model dependence (see [27], [28], [43]).

3. Use `sim()` to draw simulations of your quantity of interest (such as a predicted value, predicted probability, risk ratio, or first difference) from the model. (These simulations may be drawn using an asymptotic normal approximation (the default), bootstrapping, or other methods when available, such as directly from a Bayesian posterior.) After calling `sim()`, use any of the following to summarize the simulations:

   - The `summary()` function gives a numerical display. For multiple `setx()` values, `summary()` lets you summarize simulations by choosing one or a subset of observations.
   - If the `setx()` values consist of only one observation, `plot()` produces density plots for each quantity of interest.

Whenever possible, we use `z.out` as the `zelig()` output object, `x.out` as the `setx()` output object, and `s.out` as the `sim()` output object, but you may choose other names.

6.1.2 Examples

- Use the `turnout` data set included with Zelig to estimate a logit model of an individual's probability of voting as function of race and age. Simulate the predicted probability of voting for a white individual, with age held at its mean:

```
> data(turnout)
> z.out <- zelig(vote ~ race + age, model = "logit", data = turnout)
> x.out <- setx(z.out, race = "white")
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

- Compute a first difference and risk ratio, changing education from 12 to 16 years, with other variables held at their means in the data:

```
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.low <- setx(z.out, educate = 12)
> x.high <- setx(z.out, educate = 16)
> s.out <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out)                                    # Numerical summary.
> plot(s.out)                                       # Graphical summary.
```

- Calculate expected values for every observation in your data set:

```
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.out <- setx(z.out, fn = NULL)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

- Use five multiply imputed data sets from [42] in an ordered logit model:

```
> data(immi1, immi2, immi3, immi4, immi5)
> z.out <- zelig(as.factor(ipip) ~ wage1992 + prtyid + ideol,
                 model = "ologit",
                 data = mi(immi1, immi2, immi3, immi4, immi5))
```

- Use the nearest propensity score matching via *MatchIt* package, and then calculate the conditional average treatment effect of the job training program based on the linear regression model:

```
> library(MatchIt)
> data(lalonde)
> m.out <- matchit(treat ~ re74 + re75 + educ + black + hispan + age,
                   data = lalonde, method = "nearest")
> m.data <- match.data(m.out)
> z.out <- zelig(re78 ~ treat + distance + re74 + re75 + educ + black +
                 hispan + age, data = m.data, model = "ls")
> x.out0 <- setx(z.out, fn = NULL, treat = 0)
> x.out1 <- setx(z.out, fn = NULL, treat = 1)
> s.out <- sim(z.out, x=x.out0, x1=x.out1)
> summary(s.out)
```

- Validate the fitted model using the leave-one-out cross validation procedure and calculating the average squared prediction error via *boot* package. For example:

```
> library(boot)
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> cv.out <- cv.glm(z.out, data = turnout)
> print(cv.out$delta)
```

6.1.3 Details

1. `z.out <- zelig(formula, model, data, by = NULL, ...)`

   The `zelig()` command estimates a selected statistical model given the specified data. You may name the output object (`z.out` above) anything you desire. You must include three required arguments, in the following order:

   (a) `formula` takes the form `y ~ x1 + x2`, where `y` is the dependent variable and `x1` and `x2` are the explanatory variables, and `y`, `x1`, and `x2` are contained in the same dataset. The `+` symbol means "inclusion" not "addition." You may include interaction terms in the form of `x1*x2` without having to compute them in prior steps or include the main effects separately. For example, R treats the formula `y ~ x1*x2` as `y ~ x1 + x2 + x1*x2`. To prevent R from automatically including the separate main effect terms, use the `I()` function, thus: `y ~ I(x1 * x2)`.

   (b) `model` lets you choose which statistical model to run. You must put the name of the model in quotation marks, in the form `model = "ls"`, for example. See Section 6.2 for a list of currently supported models.

   (c) `data` specifies the data frame containing the variables called in the formula, in the form `data = mydata`. Alternatively, you may input multiply imputed datasets in the form `data = mi(data1, data2, ...)`.[1] If you are working with matched data created using MatchIt, you may create a

[1] Multiple imputation is a method of dealing with missing values in your data which is more powerful than the usual list-wise deletion approach. You can create multiply imputed datasets with a program such as Amelia; see King, Honaker, Joseph, Scheve (2000).

data frame within the `zelig()` statement by using `data = match.data(...)`. In all cases, the data frame or MatchIt object must have been previously loaded into the working memory.

(d) `by` (an optional argument which is by default `NULL`) allows you to choose a factor variable (see Section 2) in the data frame as a subsetting variable. For each of the unique strata defined in the `by` variable, `zelig()` does a separate run of the specified model. The variable chosen should *not* be in the formula, because there will be no variance in the `by` variable in the subsets. If you have one data set for all 191 countries in the UN, for example, you may use the `by` option to run the same model 191 times, once on each country, all with a single `zelig()` statement. You may also use the `by` option to run models on MatchIt subclasses.

(e) The output object, `z.out`, contains all of the options chosen, including the name of the data set. Because data sets may be large, Zelig does not store the full data set, but only the name of the dataset. Every time you use a Zelig function, it looks for the dataset with the appropriate name in working memory. (Thus, it is critical that you do *not* change the name of your data set, or perform any additional operations on your selected variables between calling `zelig()` and `setx()`, or between `setx()` and `sim()`.)

(f) If you would like to view the regression output at this intermediate step, type `summary(z.out)` to return the coefficients, standard errors, t-statistics and p-values. We recommend instead that you calculate quantities of interest; creating `z.out` is only the first of three steps in this task.

2. `x.out <- setx(z.out, fn = list(numeric = mean, ordered = median, others = mode), data = NULL, cond = FALSE, ...)`

   The `setx()` command lets you choose values for the explanatory variables, with which `sim()` will simulate quantities of interest. There are two types of `setx()` procedures:

   - You may perform the usual *unconditional* prediction (by default, `cond = FALSE`), by explicitly choosing the values of each explanatory variable yourself or letting `setx()` compute them, either from the data used to create `z.out` or from a new data set specified in the optional `data` argument. You may also compute predictions for all observed values of your explanatory variables using `fn = NULL`.
   - Alternatively, for advanced uses, you may perform *conditional* prediction (`cond = TRUE`), which predicts certain quantities of interest by conditioning on the observed value of the dependent variable. In a simple linear regression model, this procedure is not particularly interesting, since the conditional prediction is merely the observed value of the dependent variable for that observation. However, conditional prediction is extremely useful for other models and methods, including the following:
     - In a matched sampling design, the sample average treatment effect for the treated can be estimated by computing the difference between the observed dependent variable for the treated group and their expected or predicted values of the dependent variable under no treatment [5].
     - With censored data, conditional prediction will ensure that all predicted values are greater than the censored observed values [22].
     - In ecological inference models, conditional prediction guarantees that the predicted values are on the tomography line and thus restricted to the known bounds [21, 1].
     - The conditional prediction in many linear random effects (or Bayesian hierarchical) models is a weighted average of the unconditional prediction and the value of the dependent variable for that observation, with the weight being an estimable function of the accuracy of the unconditional prediction [see 4]. When the unconditional prediction is highly certain, the weight on the value of the dependent variable for this observation is very small, hence reducing inefficiency; when the unconditional prediction is highly uncertain, the relative weight on the unconditional prediction is very small, hence reducing bias. Although the simple weighted

average expression no longer holds in nonlinear models, the general logic still holds and the mean square error of the measurement is typically reduced [see 24].

In these and other models, conditioning on the observed value of the dependent variable can vastly increase the accuracy of prediction and measurement.

The `setx()` arguments for **unconditional** prediction are as follows:

(a) `z.out`, the `zelig()` output object, must be included first.

(b) You can set particular explanatory variables to specified values. For example:

```
> z.out <- zelig(vote ~ age + race, model = "logit", data = turnout)
> x.out <- setx(z.out, age = 30)
```

`setx()` sets the variables *not* explicitly listed to their mean if numeric, and their median if ordered factors, and their mode if unordered factors, logical values, or character strings. Alternatively, you may specify one explanatory variable as a range of values, creating one observation for every unique value in the range of values:[2]

```
> x.out <- setx(z.out, age = 18:95)
```

This creates 78 observations with with age set to 18 in the first observation, 19 in the second observation, up to 95 in the 78th observation. The other variables are set to their default values, but this may be changed by setting `fn`, as described next.

(c) Optionally, `fn` is a list which lets you to choose a different function to apply to explanatory variables of class

- `numeric`, which is `mean` by default,
- `ordered` factor, which is `median` by default, and
- `other` variables, which consist of logical variables, character string, and unordered factors, and are set to their `mode` by default.

While any function may be applied to numeric variables, `mean` will default to median for ordered factors, and mode is the only available option for other types of variables. In the special case, `fn = NULL`, `setx()` returns all of the observations.

(d) You cannot perform other math operations within the `fn` argument, but can use the output from one call of `setx` to create new values for the explanatory variables. For example, to set the explanatory variables to one standard deviation below their mean:

```
> X.sd <- setx(z.out, fn = list(numeric = sd))
> X.mean <- setx(z.out, fn = list(numeric = mean))
> x.out <- X.mean - X.sd
```

(e) Optionally, `data` identifies a new data frame (rather than the one used to create `z.out`) from which the `setx()` values are calculated. You can use this argument to set values of the explanatory variables for hold-out or out-of-sample fit tests.

(f) The `cond` is always `FALSE` for unconditional prediction.

If you wish to calculate risk ratios or first differences, call `setx()` a second time to create an additional set of the values for the explanatory variables. For example, continuing from the example above, you may create an alternative set of explanatory variables values one standard deviation above their mean:

```
> x.alt <- X.mean + X.sd
```

The required arguments for **conditional** prediction are as follows:

[2] If you allow more than one variable to vary at a time, you risk confounding the predictive effect of the variables in question.

(a) `z.out`, the `zelig()` output object, must be included first.

(b) `fn`, which equals `NULL` to indicate that all of the observations are selected. You may only perform conditional inference on actual observations, not the mean of observations or any other function applied to the observations. Thus, if `fn` is missing, but `cond = TRUE`, `setx()` coerces `fn = NULL`.

(c) `data`, the data for conditional prediction.

(d) `cond`, which equals `TRUE` for conditional prediction.

Additional arguments, such as any of the variable names, are ignored in conditional prediction since the actual values of that observation are used.

3. `s.out <- sim(z.out, x = x.out, x1 = NULL, num = c(1000, 100), bootstrap = FALSE, bootfn = NULL, ...)`

   The `sim()` command simulates quantities of interest given the output objects from `zelig()` and `setx()`. This procedure uses only the assumptions of the statistical model. The `sim()` command performs either unconditional or conditional prediction depending on the options chosen in `setx()`.

   The arguments are as follows for **unconditional** prediction:

   (a) `z.out`, the model output from `zelig()`.

   (b) `x`, the output from the `setx()` procedure performed on the model output.

   (c) Optionally, you may calculate first differences by specifying `x1`, an additional `setx()` object. For example, using the `x.out` and `x.alt`, you may generate first differences using:

   ```
   > s.out <- sim(z.out, x = x.out, x1 = x.alt)
   ```

   (d) By default, the number of simulations, `num`, equals 1000 (or 100 simulations if bootstrap is selected), but this may be decreased to increase computational speed, or increased for additional precision.

   (e) Zelig simulates parameters from classical *maximum likelihood* models using asymptotic normal approximation to the log-likelihood. This is the same assumption as used for frequentist hypothesis testing (which is of course equivalent to the asymptotic approximation of a Bayesian posterior with improper uniform priors). See King, Tomz, and Wittenberg (2000). For *Bayesian models*, Zelig simulates quantities of interest from the posterior density, whenever possible. For *robust Bayesian models*, simulations are drawn from the identified class of Bayesian posteriors.

   (f) Alternatively, you may set `bootstrap = TRUE` to simulate parameters using bootstrapped data sets. If your dataset is large, bootstrap procedures will usually be more memory intensive and time-consuming than simulation using asymptotic normal approximation. The type of bootstrapping (including the sampling method) is determined by the optional argument `bootfn`, described below.

   (g) If `bootstrap = TRUE` is selected, `sim()` will bootstrap parameters using the default `bootfn`, which re-samples from the data frame with replacement to create a sampled data frame of the same number of observations, and then re-runs `zelig()` (inside `sim()`) to create one set of bootstrapped parameters. Alternatively, you may create a function outside the `sim()` procedure to handle different bootstrap procedures. Please consult `help(boot)` for more details.[3]

   For **conditional** prediction, `sim()` takes only two required arguments:

   (a) `z.out`, the model output from `zelig()`.

   (b) `x`, the conditional output from `setx()`.

[3]If you choose to create your own `bootfn`, it must include the the following three arguments: `data`, the original data frame; one of the sampling methods described in `help(boot)`; and `object`, the original `zelig()` output object. The alternative bootstrapping function must sample the data, fit the model, and extract the model-specific parameters.

(c) Optionally, for duration models, `cond.data`, which is the `data` argument from `setx()`. For models for duration dependent variables (see Section 6), `sim()` must impute the uncensored dependent variables before calculating the average treatment effect. Inputting the `cond.data` allows `sim()` to generate appropriate values.

Additional arguments are ignored or generate error messages.

Presenting Results

1. Use `summary(s.out)` to print a summary of your simulated quantities. You may specify the number of significant digits as:

   ```
   > print(summary(s.out), digits = 2)
   ```

2. Alternatively, you can plot your results using `plot(s.out)`.

3. You can also use `names(s.out)` to see the names and a description of the elements in this object and the `$` operator to extract particular results. For most models, these are: `s.out$qi$pr` (for predicted values), `s.out$qi$ev` (for expected values), and `s.out$qi$fd` (for first differences in expected values). For the logit, probit, multinomial logit, ordinal logit, and ordinal probit models, quantities of interest also include `s.out$qi$rr` (the risk ratio).

6.2 Supported Models

We list here all models implemented in Zelig, organized by the nature of the dependent variable(s) to be predicted, explained, or described.

1. **Continuous Unbounded** dependent variables can take any real value in the range $(-\infty, \infty)$. While most of these models take a continuous dependent variable, Bayesian factor analysis takes multiple continuous dependent variables.

   (a) `"ls"`: The *linear least-squares* (see Section 14.1) calculates the coefficients that minimize the sum of squared residuals. This is the usual method of computing linear regression coefficients, and returns unbiased estimates of β and σ^2 (conditional on the specified model).

   (b) `"normal"`: The *Normal* (see Section 16.1) model computes the maximum-likelihood estimator for a Normal stochastic component and linear systematic component. The coefficients are identical to `ls`, but the maximum likelihood estimator for σ^2 is consistent but biased.

2. **Dichotomous** dependent variables consist of two discrete values, usually $(0, 1)$.

   (a) `"logit"`: *Logistic regression* (see Section 13.1) specifies $\Pr(Y = 1)$ to be a(n inverse) logistic transformation of a linear function of a set of explanatory variables.

   (b) `"probit"`: *Probit regression* (see Section 18.1) Specifies $\Pr(Y = 1)$ to be a(n inverse) CDF normal transformation as a linear function of a set of explanatory variables.

   (c) `"blogit"`: The *bivariate logistic* model models $\Pr(Y_{i1} = y_1, Y_{i2} = y_2)$ for $(y_1, y_2) = (0, 0), (0, 1), (1, 0), (1, 1)$ according to a bivariate logistic density.

   (d) `"bprobit"`: The *bivariate probit* model models $\Pr(Y_{i1} = y_1, Y_{i2} = y_2)$ for $(y_1, y_2) = (0, 0), (0, 1), (1, 0), (1, 1)$ according to a bivariate normal density.

3. **Ordinal** are used to model ordered, discrete dependent variables. The values of the outcome variables (such as kill, punch, tap, bump) are ordered, but the distance between any two successive categories is not known exactly. Each dependent variable may be thought of as linear, with one continuous, unobserved dependent variable observed through a mechanism that only returns the ordinal choice.

(a) `"ologit"`: The *ordinal logistic* model (see Section 30.1) specifies the stochastic component of the unobserved variable to be a standard logistic distribution.

(b) `"oprobit"`: The *ordinal probit* distribution (see Section 31.1) specifies the stochastic component of the unobserved variable to be standardized normal.

4. **Multinomial** dependent variables are unordered, discrete categorical responses. For example, you could model an individual's choice among brands of orange juice or among candidates in an election.

   (a) `"mlogit"`: The *multinomial logistic* model (see specifies categorical responses distributed according to the multinomial stochastic component and logistic systematic component.

5. **Count** dependent variables are non-negative integer values, such as the number of presidential vetoes or the number of photons that hit a detector.

   (a) `"poisson"`: The *Poisson* model (see Section 17.1) specifies the expected number of events that occur in a given observation period to be an exponential function of the explanatory variables. The Poisson stochastic component has the property that, $\lambda = \mathrm{E}(Y_i|X_i) = \mathrm{V}(Y_i|X_i)$.

   (b) `"negbin"`: The *negative binomial* model (see Section 15) has the same systematic component as the Poisson, but allows event counts to be over-dispersed, such that $\mathrm{V}(Y_i|X_i) > \mathrm{E}(Y_i|X_i)$.

6. **Continuous Bounded** dependent variables that are continuous only over a certain range, usually $(0, \infty)$. In addition, some models (exponential, lognormal, and Weibull) are also censored for values greater than some censoring point, such that the dependent variable has some units fully observed and others that are only partially observed (censored).

   (a) `"gamma"`: The *Gamma* model (see Section 12.1) for positively-valued, continuous dependent variables that are fully observed (no censoring).

7. **Mixed** dependent variables include models that take more than one dependent variable, where the dependent variables come from two or more of categories above. (They do not need to be of a homogeneous type.)

   (a) The *Bayesian mixed factor analysis* model, in contrast to the Bayesian factor analysis model and ordinal factor analysis model, can model both types of dependent variables as a function of latent explanatory variables.

8. **Ecological inference** models estimate unobserved internal cell values given contingency tables with observed row and column marginals.

6.3 Replication Procedures

A large part of any statistical analysis is documenting your work such that given the same data, anyone may replicate your results. In addition, many journals require the creation and dissemination of "replication data sets" in order that others may replicate your results (see King, 1995). Whether you wish to create replication materials for your own records, or contribute data to others as a companion to your published work, Zelig makes this process easy.

6.3.1 Saving Replication Materials

Let `mydata` be your final data set, `z.out` be your `zelig()` output, and `s.out` your `sim()` output. To save all of this in one file, type:

```
> save(mydata, z.out, s.out, file = "replication.RData")
```

This creates the file replication.RData in your working directory. You may compress this file using zip or gzip tools.

If you have run several specifications, all of these estimates may be saved in one .RData file. Even if you only created quantities of interest from one of these models, you may still save all the specifications in one file. For example:

```
> save(mydata, z.out1, z.out2, s.out, file = "replication.RData")
```

Although the .RData format can contain data sets as well as output objects, it is not the most space-efficient way of saving large data sets. In an uncompressed format, ASCII text files take up less space than data in .RData format. (When compressed, text-formatted data is still smaller than .RData-formatted data.) Thus, if you have more than 100,000 observations, you may wish to save the data set separately from the Zelig output objects. To do this, use the `write.table()` command. For example, if `mydata` is a data frame in your workspace, use `write.table(mydata, file = "mydata.tab")` to save this as a tab-delimited ASCII text file. You may specify other delimiters as well; see `help.zelig("write.table")` for options.

6.3.2 Replicating Analyses

If the data set and analyses are all saved in one .RData file, located in your working directory, you may simply type:

```
> load("replication.RData")                 # Loads the replication file.
> z.rep <- repl(z.out)                      # To replicate the model only.
> s.rep <- repl(s.out)                      # To replicate the model and
                                            #  quantities of interest.
```

By default, `repl()` uses the same options used to create the original output object. Thus, if the original `s.out` object used bootstrapping with 245 simulations, the `s.rep` object will similarly have 245 bootstrapped simulations. In addition, you may use the `prev` option when replicating quantities of interest to reuse rather than recreate simulated parameters. Type `help.zelig("repl")` to view the complete list of options for `repl()`.

If the data were saved in a text file, use `read.table()` to load the data, and then replicate the analysis:

```
> dat <- read.table("mydata.tab", header = TRUE)  # Where `dat' is the same
> load("replication.RData")                       #   as the name used in
> z.rep <- repl(z.out)                            #   `z.out'.
> s.rep <- repl(s.out)
```

If you have problems loading the data, please refer to Section 4.2.2.

Finally, you may use the `identical()` command to ensure that the replicated regression output is in every way identical to the original `zelig()` output.[4] For example:

```
> identical(z.out$coef, z.rep$coef)              # Checks the coefficients.
```

Simulated quantities of interest will vary from the original quantities if parameters are re-simulated or re-sampled. If you wish to use `identical()` to verify that the quantities of interest are identical, you may use

```
# Re-use the parameters simulated (and stored) in the original sim() output.
> s.rep <- repl(s.out, prev = s.out$par)

# Check that the expected values are identical.  You may do this for each qi.
> identical(s.out$qi$ev, s.rep$qi$ev)
```

[4]The `identical()` command checks that numeric values are identical to the maximum number of decimal places (usually 16), and also checks that the the two objects have the same class (numeric, character, integer, logical, or factor). Refer to `help(identical)` for more information.

Chapter 7

Graphing Commands

R, and thus Zelig, can produce exceptionally beautiful plots. Many built-in plotting functions exist, including scatter plots, line charts, histograms, bar charts, pie charts, ternary diagrams, contour plots, and a variety of three-dimensional graphs. If you desire, you can exercise a high degree of control to generate just the right graphic. Zelig includes several default plots for one-observation simulations for each model. To view these plots on-screen, simply type `plot(s.out)`, where `s.out` is the output from `sim()`. Depending on the model chosen, `plot()` will return different plots.

If you wish to create your own plots, this section reviews the most basic procedures for creating and saving two-dimensional plots. R plots material in two steps:

1. You must call an output device (discussed in Section 7.3), select a type of plot, draw a plotting region, draw axes, and plot the given data. At this stage, you may also define axes labels, the plot title, and colors for the plotted data. Step one is described in Section 7.1 below.

2. Optionally, you may add points, lines, text, or a legend to the existing plot. These commands are described in Section 7.2.

7.1 Drawing Plots

The most generic plotting command is `plot()`, which automatically recognizes the type of R object(s) you are trying to plot and selects the best type of plot. The most common graphs returned by `plot()` are as follows:

1. If `X` is a variable of length n, `plot(X)` returns a scatter plot of (x_i, i) for $i = 1, \ldots n$. If X is unsorted, this procedure produces a messy graph. Use `plot(sort(X))` to arrange the plotted values of (x_i, i) from smallest to largest.

2. With two numeric vectors `X` and `Y`, both of length n, `plot(X, Y)` plots a scatter plot of each point (x_i, y_i) for $i = 1, \ldots n$. Alternatively, if Z is an object with two vectors, `plot(Z)` also creates a scatter plot.

Optional arguments specific to `plot` include:

- `main` creates a title for the graph, and `xlab` and `ylab` label the x and y axes, respectively. For example,

```
plot(x, y, main = "My Lovely Plot", xlab = "Explanatory Variable",
     ylab = "Dependent Variable")
```

- `type` controls the type of plot you request. The default is `plot(x, y, type = "p")`, but you may choose among the following types:

`"p"`	points
`"l"`	lines
`"b"`	both points and lines
`"c"`	lines drawn up to but not including the points
`"h"`	histogram
`"s"`	a step function
`"n"`	a blank plotting region (with the axes specified)

- If you choose `type = "p"`, R plots open circles by default. You can change the type of point by specifying the `pch` argument. For example, `plot(x, y, type = "p", pch = 19)` creates a scatterplot of filled circles. Other options for `pch` include:

19	solid circle (a disk)
20	smaller solid circle
21	circle
22	square
23	diamond
24	triangle pointed up
25	triangle pointed down

  In addition, you can specify your own symbols by using, for example, `pch = "*"` or `pch = "."`.

- If you choose `type = "l"`, R plots solid lines by default. Use the optional `lty` argument to set the line type. For example, `plot(x, y, type = "l", lty = "dashed")` plots a dashed line. Other options are dotted, dotdash, longdash, and twodash.

- `col` sets the color of the points, lines, or bars. For example, `plot(x, y, type = "b", pch = 20, lty = "dotted", col = "violet")` plots small circles connected by a dotted line, both of which are violet. (The axes and labels remain black.) Use `colors()` to see the full list of available colors.

- `xlim` and `ylim` set the limits to the x-axis and y-axis. For example, `plot(x, y, xlim = c(0, 25), ylim = c(-15, 5))` sets range of the x-axis to $[0, 25]$ and the range of the y-axis to $[-15, 5]$.

For additional plotting options, refer to `help(par)`.

7.2 Adding Points, Lines, and Legends to Existing Plots

Once you have created a plot, you can *add* points, lines, text, or a legend. To place each of these elements, R uses coordinates defined in terms of the x-axes and y-axes of the plot area, not coordinates defined in terms of the the plotting window or device. For example, if your plot has an x-axis with values between $[0, 100]$, and a y-axis with values between $[50, 75]$, you may add a point at $(55, 55)$.

- **points()** plots one or more sets of points. Use `pch` with `points` to add points to an existing plot. For example, `points(P, Q, pch = ".", col = "forest green")` plots each (p_i, q_i) as tiny green dots.

- **lines()** joins the specified points with line segments. The arguments `col` and `lty` may also be used. For example, `lines(X, Y, col = "blue", lty = "dotted")` draws a blue dotted line from each set of points (x_i, y_i) to the next. Alternatively, `lines` also takes command output which specifies (x, y) coordinates. For example, `density(Z)` creates a vector of x and a vector of y, and `plot(density(Z))` draws the kernel density function.

- **text()** adds a character string at the specified set of (x, y) coordinates. For example, `text(5, 5, labels = "Key Point")` adds the label "Key Point" at the plot location $(5, 5)$. You may also choose the font using the `font` option, the size of the font relative to the axis labels using the `cex` option, and choose a color using the `col` option. The full list of options may be accessed using `help(text)`.

- **legend()** places a legend at a specified set of (x, y) coordinates. Type `demo(vertci)` to see an example for `legend()`.

7.3 Saving Graphs to Files

By default, R displays graphs in a window on your screen. To save R plots to file (to include them in a paper, for example), preface your plotting commands with:

```
> ps.options(family = c("Times"), pointsize = 12)
> postscript(file = "mygraph.eps", horizontal = FALSE, paper = "special",
             width = 6.25, height = 4)
```

where the `ps.options()` command sets the font type and size in the output file, and the `postscript` command allows you to specify the name of the file as well as several additional options. Using `paper = special` allows you to specify the width and height of the encapsulated postscript region in inches (6.25 inches long and 4 inches high, in this case), and the statement `horizontal = FALSE` suppresses R's default landscape orientation. Alternatively, you may use `pdf()` instead of `postscript()`. If you wish to select postscript options for .pdf output, you may do so using options in `pdf()`. For example:

```
> pdf(file = "mygraph.pdf", width = 6.25, height = 4, family = "Times",
+     pointsize = 12)
```

At the end of every plot, you should close your output device. The command `dev.off()` stops writing and saves the `.eps` or `.pdf` file to your working directory. If you forget to close the file, you will write all subsequent plots to the same file, overwriting previous plots. You may also use `dev.off()` to close on-screen plot windows.

To write multiple plots to the same file, you can use the following options:

- For plots on separate pages in the same `.pdf` document, use

  ```
  > pdf(file = "mygraph.pdf", width = 6.25, height = 4, family = "Times",
  +     pointsize = 12, onefile = TRUE)
  ```

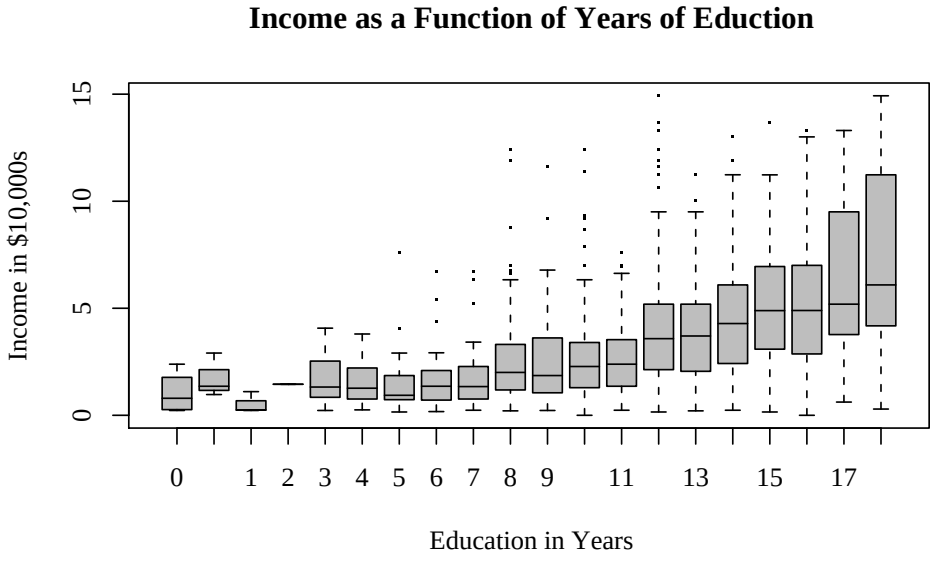
- For multiple plots on one page, initialize either a `.pdf` or `.eps` file, then (before any plotting commands) type:

  ```
  par(mfrow = c(2, 4))
  ```

  This creates a grid that has two rows and four columns. Your plot statements will populate the grid going across the first row, then the second row, from left to right.

7.4 Examples

7.4.1 Descriptive Plots: Box-plots

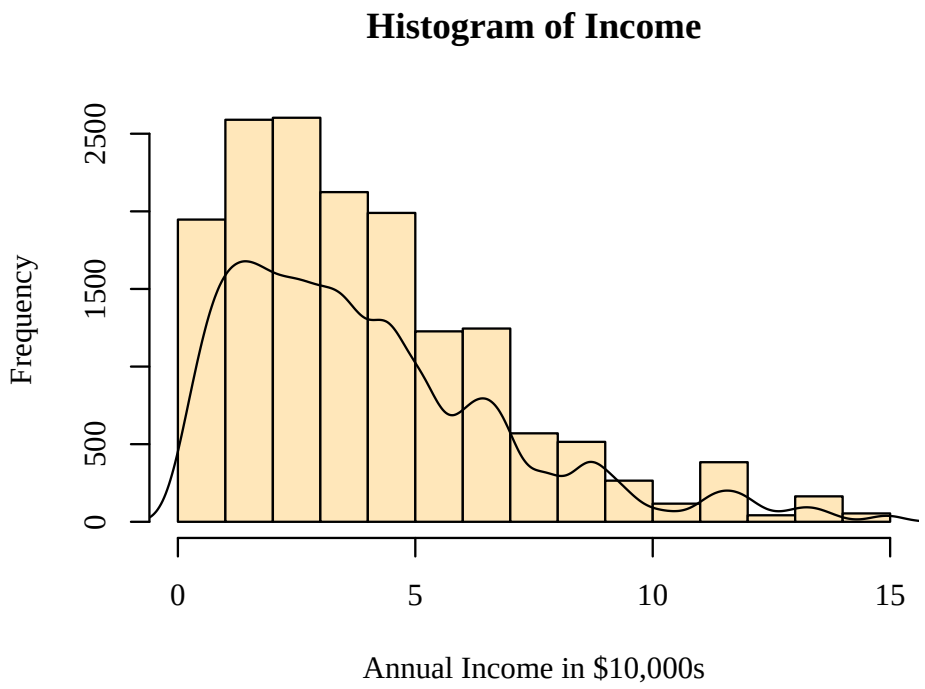


Using the sample `turnout` data set included with Zelig, the following commands will produce the graph above.

```
> library(Zelig)                              # Loads the Zelig package.
> data (turnout)                              # Loads the sample data.
> boxplot(income ~ educate,                   # Creates a boxplot with income
+  data = turnout, col = "grey", pch = ".",   #  as a function of education.
+  main = "Income as a Function of Years of Education",
+  xlab = "Education in Years", ylab = "Income in \$10,000s")
```

7.4.2 Density Plots: A Histogram

Histograms are easy ways to evaluate the density of a quantity of interest.



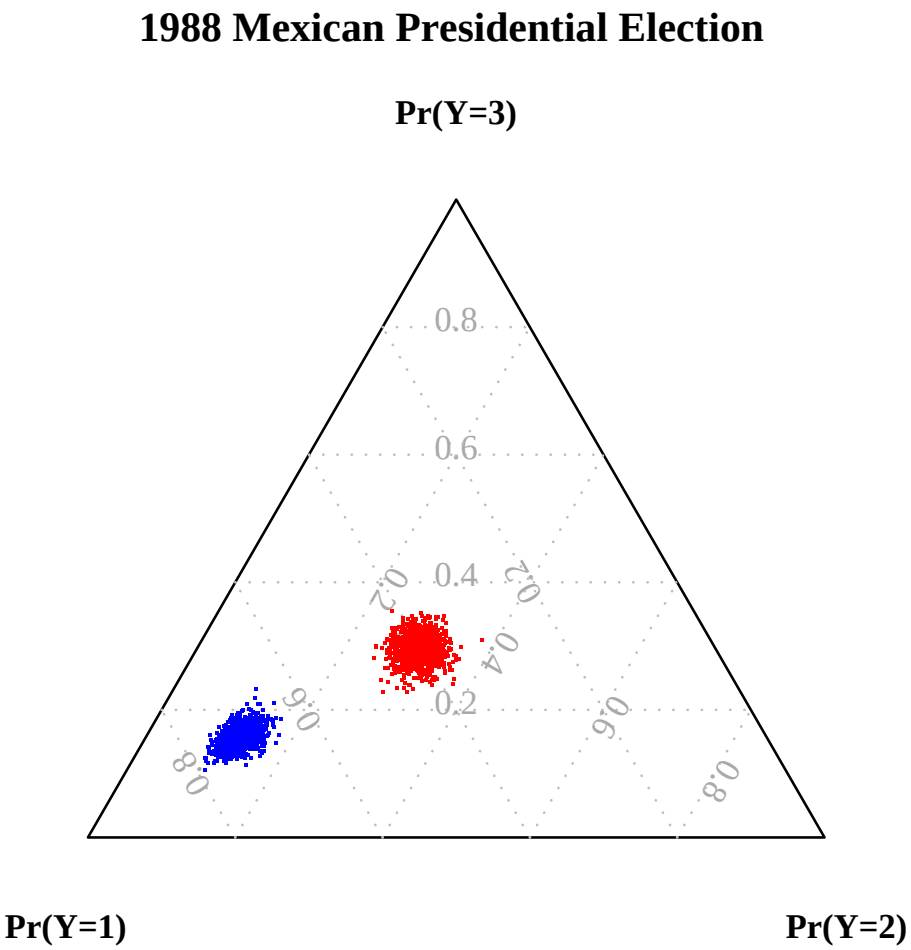
Here's the code to create this graph:

```
> library(Zelig)                              # Loads the Zelig package.
> data(turnout)                               # Loads the sample data set.
> truehist(turnout$income,  col = "wheat1",   # Calls the main plot, with
+       xlab = "Annual Income in $10,000s",   #  options.
+       main = "Histogram of Income")
> lines(density(turnout$income))              # Adds the kernel density line.
```

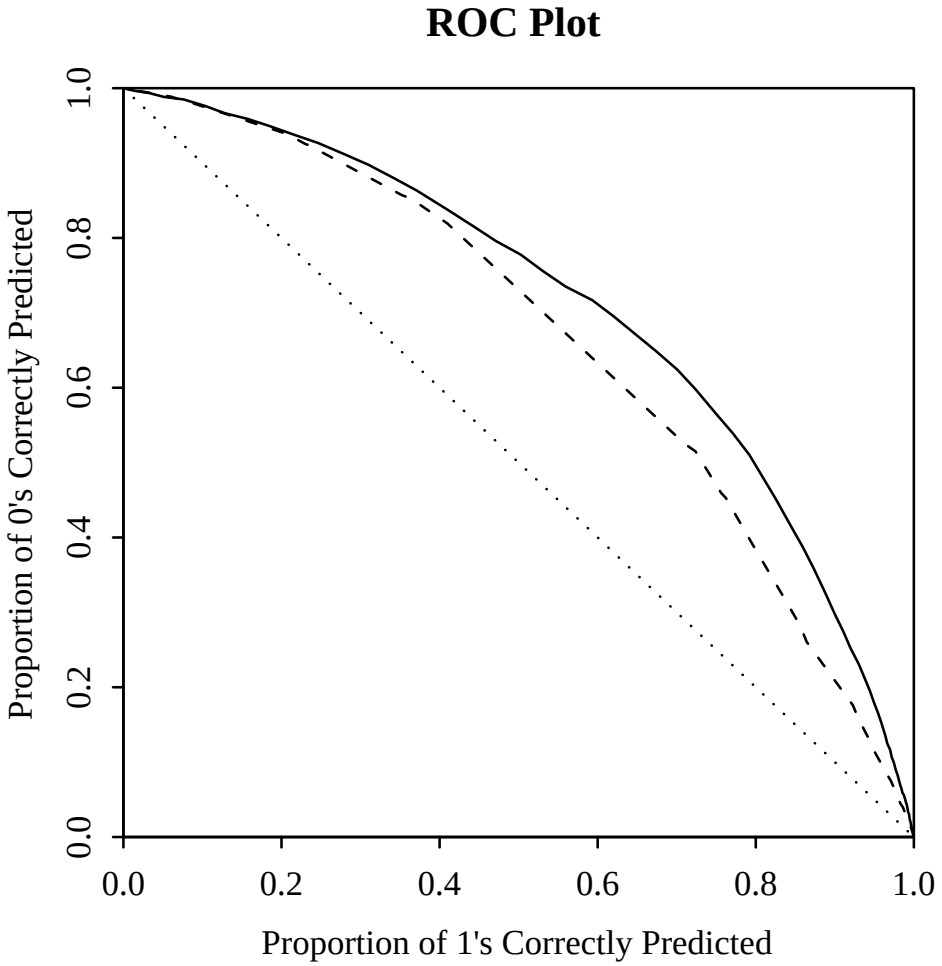
7.4.3 Advanced Examples

The examples above are simple examples which only skim the surface of R's plotting potential. We include more advanced, model-specific plots in the Zelig demo scripts, and have created functions that automate some of these plots, including:

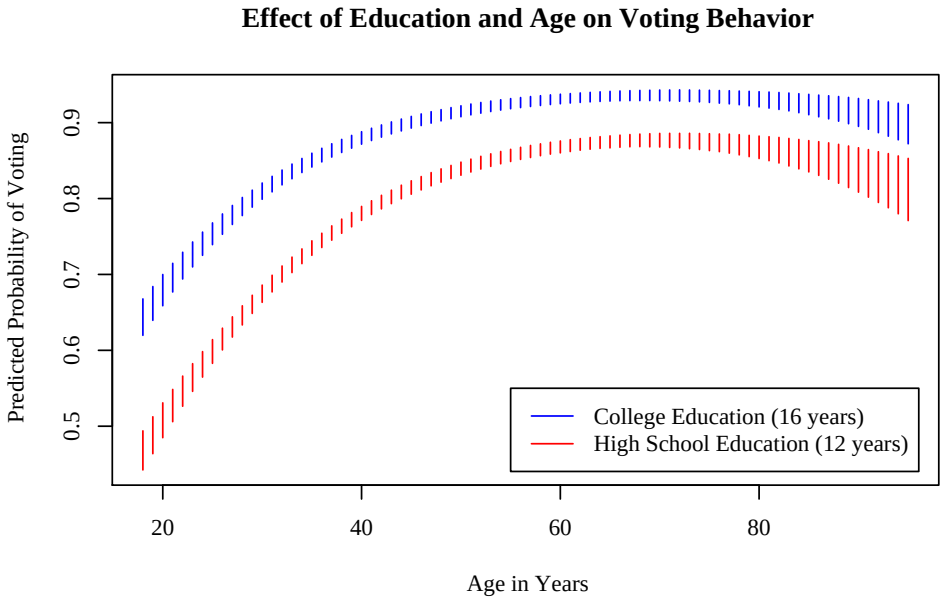
1. **Ternary Diagrams** describe the predicted probability of a categorical dependent variable that has three observed outcomes. You may choose to use this plot with the multinomial logit, the ordinal logit, or the ordinal probit models (Katz and King, 1999)..



2. **ROC Plots** summarize how well models for binary dependent variables (logit and probit) fit the data. The ROC plot evaluates the fraction of 0's and 1's correctly predicted for every possible threshold value at which the continuous Prob($Y = 1$) may be realized as a dichotomous prediction. The closer the ROC curve is to the upper right corner of the plot, the better the fit of the model specification (King and Zeng, 2002*b*). See Section 3 for the sample code, and type `demo(roc)` at the R prompt to run the example.



3. **Vertical Confidence Intervals** may be used for almost any model, and describe simulated confidence intervals for any quantity of interest while allowing one of the explanatory variables to vary over a given range of values (King, Tomz and Wittenberg, 2000). Type `demo(vertci)` at the R prompt to run the example, and `help.zelig(plot.ci)` for the manual page.



Part IV

The Zelig API

Chapter 8

Rapid Development Guide

8.1 Introduction

Programming a Zelig module is a simple procedure. By following several simple steps, any statistical model can be implemented in the Zelig software suite. The following document places emphasis on speed and practicality, rather than the numerous, technical details involved in developing statistical models. That is, this guide will explain how to quickly and most simply include existing statistical software in the Zelig suite.

8.2 Overview

In order for a Zelig model to function correctly, four components need to exist:

***a statistical model*:** This can be any statistical model of the developer's choosing, though it is suggested that it be written in R. Examples of statistical models already implemented in Zelig include: Brian Ripley's `glm` and Kosuke Imai's `MNP` models.

`zelig2`***model*** This method acts as a bridge between the external statistical model and the Zelig software suite

`param.`***model*** This method specifies the simulated parameters used to compute quantities of interest

`qi.`***model*** This method computes - using the fitted statistical model, simulated parameters, and explanatory data - the *quantities of interest*. Compared with the `zelig2` and `param` methods,

In the above description, replace the italicized *model* text with the name of the developer's model. For example, if the model's name is "logit", then the corresponding methods will be titled `zelig2logit`, `param.logit`, and `qi.logit`.

8.3 `zelig.skeleton`: Automating Zelig Model Creation

The fastest way to setup and begin programming a Zelig model is the use the `zelig.skeleton` function, available within the `Zelig` package. This function allows a fast, simple way to create the `zelig2`, `describe`, `param`, and `qi` methods with the necessary boilerplate. As a result, `zelig.skeleton` closely mirrors the `package.skeleton` method included in core R.

8.3.1 A Demonstrative Example

```
library(Zelig)  # [1]

zelig.skeleton(
               "my.zelig.package",                # [2]
               models = c("gamma", "logit"),      # [3]
               author = "Your Name",              # [4]
               email = "your.email@someplace.com" # [5]
               )
```

8.3.2 Explanation of the `zelig.skeleton` Example

The above numbered comments correspond to the following:

[**1**] The Zelig package must be imported when using `zelig.skeleton`.

[**2**] The first parameter of `zelig.skeleton` specifies the name of the package

[**3**] The `models` parameter specifies the titles of the Zelig models to be included in the package. In the above example, all necessary files and methods for building the "gamma" and "logit" models will be included in Zelig package.

[**4**] Specify the author's name

[**5**] Specify the email address of the software maintainer

8.3.3 Conclusion

The `zelig.skeleton` function provides a way to automatically generate the necessary methods and file to create an arbitrary Zelig package. The method body, however, will be incomplete, save for some light documentation additions and programming boilerplate. For a detailed specification of the `zelig.skeleton` method, refer to Zelig help file by typing:

```
library(Zelig)

?zelig.skeleton
```

in an interactive R-session.

8.4 *zelig2*: Interacting with Existing Statistical Models in Zelig

The `zelig2` function acts as the bridge between the Zelig module and the existing statistical model. That is, the results of this function specify the parameters to be passed to a *previously completed* statistical model-fitting function. In this sense, there is nothing tricky about the `zelig2` function. Simply construct a list with key-value pairs in the following fashion:

- **Keys** (names on the lefthand-side of an equal sign) represent parameters that are submitted to the existing model function
- **Values** (variables, etc. on the righthand-side of an equal sign) represent values to set the corresponding the parameter to.
- **Keys with leading periods** are typically reserved for specific `zelig2` purposes. In particular, the key `.function` specifies the name of the function that calls the existing statistical model.

an ellipsis (. . .) specifies that all additional, optional parameters not specified in the signature of the `zelig2model_function` method, will be included in the external method's call, despite not being specifically set.

8.4.1 A Simple Example

For example, if a developer wanted to call an existing model `"SomeModel"` with the parameter `weights` set to `1`, the appropriate return-value (a list) for the `zelig2` function would be:

```
zelig2some.model <- function(formula, data) {
    list(.function = "SomeModel",
         formula   = formula,
         weights   = 1
         )
}
```

8.4.2 A More Detailed Example

A more typical example would be the case of fitting a basic logistic regression. The following code, already implemented in Zelig, acts as an interface between Zelig packages and R's built-in `glm` function:

```
zelig2logit <- function (formula, weights = NULL, ..., data) {
  list(.function = "glm",    # [1]

       formula = formula,    # [2]
       weights = weights,    # ...
       data    = data,       # ...

       family  =             # [3]
                 binomial(link="logit"),
       model   = FALSE       # ...
       )
}
```

The comments in the above code correspond to the following:

[**1**] Pass all parameters to the `glm` function

[**2**] Specify that the parameters `formula`, `weights`, and `data` be given the same values as those passed into the `zelig2` function itself. That is, whichever values the end-user passes to `zelig` will be passed to the `glm` function

[**3**] Specify that the parameters `family` and `model` *always* be given the corresponding values - `binomial(link="logit")` and `FALSE` - regardless of what the end-user passes as a parameter.

Note that the parameters - `formula`, `weights`, `data`, `family`, `model` - correspond to those of the `glm` function. In general, this will be the case for any `zelig2` method. That is, every `zelig2` method should return a list containing the parameters belonging to the external model, as well as, the reserved keyword `.function`.

If you are unsure about the parameters that are passed to an existing statistical model, simply use the `args` or `formals` functions (included in R). For example, to get a list of acceptable parameters to the `glm` function, simply type:

```
args(glm)
```

8.4.3 An Even-More Detailed Example

Occasionally the statistical model and the standard style of Zelig input differ. In these instances, it may be necessary to manipulate information about the `formula` and `constraints`. This additional step in building the `zelig2` method is common only amongst multivariate models, as seen below in the `bprobit` model (bivariate probit regression for Zelig).

```
zelig2bprobit <- function(formula, ..., data) {

  # [1]
  formula <- parse.formula(formula, "bprobit")

  # [2]
  tmp <- cmvglm(formula, "bprobit", 3)


  # return list
  list(
       .function = "vglm",    # [3]

       formula = tmp$formula, # [4]
       family  = bprobit,     # [5]
       data = data,
       # [6]
       constraints = tmp$constraints
       )
}
```

The following is an explanation of the above code:

[**1**] Convert Zelig-style `formula` data-types into the style that the `vglm` function understands

[**2**] Extract constraint information from the `formula` object, as is the style commonly supported by Zelig

[**3**] Specify the `vglm` as the statistical model fitting function

[**4**] Specify the formula to be used by the `vglm` function when performing the model fitting. Note that this object is created by using both the `parse.formula` and `cmvglm` functions

[**5**] Specify the `family` of the model

[**6**] Specify the constraints to be used by the `vglm` function when performing the model fitting. Note that this object is created by using both the `parse.formula` and `cmvglm` functions

Note that the functions `parse.formula` and `cmvglm` are included in the core Zelig software package. Information concerning these functions can be found by typing:

```
library(Zelig)

?parase.formula
?cmvglm
```

in an interactive R-session.

8.4.4 Summary and More Information aboput `zelig2` Methods

`zelig2` functions can be of varying difficulty - from simple parameter passing to reformatting and creating new data objects to be used by the external model-fitting function. To see more examples of this usage, please refer to the `survey.zelig` and `multinomial.zelig` packages. Regardless of the model's complexity, it ends with a simple list specifying which parameters to pass to a preexisting statistical model.

For more information on the `zelig2` function's full features, see the *Advanced zelig2 Manual*, or type:

```
library(Zelig)

?zelig2
```

within an interactive R-session.

8.5 *param*: Simulating Parameters

The `param` function simulates and specifies parameters necessary for computing *quantities of interest*. That is, the `param` function is the ideal place to specify information necessary for the `qi` method. This includes:

Ancillary parameters These parameters specifying information about the underlying probability distribution. For example, in the case of the Normal Distribution, σ (standard deviation) and μ (mean) would be considered ancillary parameters.

Link function That is, the function providing the relationship between the predictors and the mean of the distribution function. This is typically of very little importance (compared to the inverse link function), but is frequently included for completeness. For Gamma distribution, the link function is the inverse function: $f(x) = \frac{1}{x}$

Inverse link function Typically crucial for simulating *quantities of interest* of *Generalized Linear Models*. For the binomial distribution, the inverse-link function is the logit function: $f(x) = \frac{e^x}{1+e^x}$

Simulated Parameters These random draws simulate the parameters of the fitted statistical model. Typically, the `qi` method uses these to simulate *quantities of interest* for the given model. As a result, these are of paramount importance.

The following sections describe how these ideas correspond to the structure of a well-written `param` function.

8.5.1 The Function Signature

The `param` function takes only two parameters, but outputs a wealth of information important in computing *quantities of interest*. The following is the function signature:

```
param.logit <- function (obj, num)
```

The above parameters are:

obj An object of class `zelig` [1]. This contains the fitted statistical model and associated information.

num An integer specifying the number of simulations to be drawn. This value is specified by the end-user, and defaults to `1000` if no value is specified.

[1] For a detailed specification of the `zelig` class, type: `?zelig` within a interactive Zelig-session.

8.5.2 The Function Return Value

In similar fashion to the `zelig2` method, the `param` method takes return values as a list of key-value pairs. However, the options are not as diverse. That is, the list can only be given a set of specific values: `ancillary`, `coef`, `simulations`, `link`, `linkinv`, and `family`.

In most cases, however, the parameters `ancillary`, `simulations`, and `linkinv` are sufficient. The following is an example take from Zelig's `gamma` model:

```
# Simulate Parameters for the gamma Model
param.gamma <- function(obj, num) {

# NOTE: gamma.shape is a method belonging to the
#       GLM class, specifying maximum likelihood
#       estimates of the distribution's shape
#       parameter. It is a list containing two
#       values: 'alpha' and 'SE'

  shape <- gamma.shape(obj)

  # simulate ancillary parameters
  alpha <- rnorm(n=num, mean=shape$alpha, sd=shape$SE)

  # simulate maximum
  sims <- mvrnorm(n = num, mu = coef(obj), Sigma = vcov(obj))

# return results
  list(
       alpha = alpha,       # [1]
       simulations  = sims, # [2]
                            # ...

                            # [3]
       linkinv = function (x) 1/x
       )
}
```

The above code does the following:

[**1**] Specify the ancillary parameters, typically referred to as the greek letter α. In the above example, `alpha` is the *shape* of the model's underlying gamma distribution.

[**2**] Specify the parameter simulations, typically referred to as the greek letter β, to be used in the `qi` function.

[**3**] Specify the inverse-link function [2], used to compute *expected values* and a variety of other *quantities of interest*, once samples are extracted from the model's statistical distribution.

8.5.3 Summary and More Information `param` Methods

The `param` method's basic purpose is to describe the statistical and systematic variables of the Zelig model's underlying distribution. Defining this method is an important step towards simplifying the `sim` method. That is, by specifying features of the model - coefficients, systematic components, inverse link functions, etc. - and simulating specific parameters, the `sim` method can focus entirely on simulating *quantities of interest*.

8.6 qi: Simulating Quantities of Interest

The `qi` function of any Zelig model simulates *quantities of interest* using the fitted statistical model, taken from the `zelig2` function, and the simulated parameters, taken from the `param` function. As a result, the

[2] The "inverse-link" function is also commonly referred to as the "mean" function. Typically, this function specifies the relationship between linear predictors and the mean of a distribution function. As a result, it is only used in describing *generalized linear models*

`qi` function is the most important component of a Zelig model.

8.6.1 The `qi` Function Signature

While the implementation of the `qi` function can differ greatly from one model to another, the signature always remains the same and closely parallels the signature of the `sim` function.

```
qi.logit <- function(obj, x=NULL, x1=NULL, y=NULL, param=NULL)
```

8.6.2 The `qi` Function Return Values

Similar to the return values of both the `zelig2` and `param` function, the `qi` function takes an list of key-value pairs as a return value. The keys, however, follow a much simpler convention, and a single rule: the key (left-side of the equal sign) is a *quoted* character-string naming the *quantity of interest* and the value (right-side of the equal sign) are the actual simulations.

The following is a short example:

```
list(
     "Expected Value"  = ev,
     "Predicted Value" = pv
     )
```

where `ev` and `pv` are respectively simulations of the model's *expected values* and *predicted values*.

8.6.3 Coding Conventions for the `qi` Function

While the following is unnecessary, it provides a few simple guidelines to simplifying and improving readability of a model's `qi` function:

- Divide repetitive work amongst other functions. For example, if you simulate an *expected value* for both the `x` and `x1`, it is better to write a `.compute.ev` function and simply call it twice
- Always compute an *expected values* and *predicted values* independently and before writing code to create *first differences*, *risk ratios*, and *average treatment effects*
- Write code for *average treatment effects* only after all the other code has been debugged and completed

8.6.4 A Simplified Example

The following is a simplified example of the `qi` function for the logit model. Note that the example is divided into two sections: one specifying the return values and titles of the *quantities of interest* (see Section 8.6.4) and one computing the simulated *expected values* of the model (see Section 8.6.4).

`qi.logit` Function

```
#' simulate quantities of interest for the logit models
qi.logit <- function(obj, x=NULL, x1=NULL, y=NULL, num=1000,
                     param=NULL) {
  # [1]
  ev1 <- .compute.ev(obj, x, num, param)
  ev2 <- .compute.ev(obj, x1, num, param)

  # [2]
  list(
       "Expected Values: E(Y|X)"  = ev1,
       "Expected Values (for X1)" = ev2,

  # [3]
       "First Differences: E(Y|X1) - E(Y|X)" = ev2 - ev1
       )
}
```

`.compute.ev` Function

```
.compute.ev <- function(obj, x=NULL, num=1000, param=NULL) {
  # values of NA are ignored by the summary function
  if (is.null(x))
    return(NA)

  # extract simulations
  coef <- coef(param)

  link.inverse <- linkinv(param)

  eta <- coef %*% t(x)

  # invert link function
  theta <- matrix(link.inverse(eta), nrow = nrow(coef))
  ev <- matrix(theta, ncol=ncol(theta))

  ev
}
```

The above code illustrates a few of the ideas:

[**1**] Compute *quantities of interest* using re-usable functions that express the idea clearly. This both reduces the amount of code necessary to produce the simulations, and improves readability of the source code.

[**2**] Return *quantities of interest* as a list. Note: titles of *quantities of interest* are on the left of the equal signs, while simulated values are on the right.

[**3**] Simulate *first differences* by using two previous computed *quantities of interest*.

[**4**] Define an additional function that simulates *expected values*, rather than placing such code in the actual `qi` method.

In addition, this function two *generic functions* that are defined in the Zelig software suite, and are particularly used with the `param` class:

coef Extract the simulations of the parameters. Specifically, this returns the simulations produced in the `param` function

linkinv Return the inverse of the link function. Specifically, this returns the inverse-link functions specified in the `param` function

8.6.5 Summary and More Information about `qi` Methods

The `qi` function offers a simple template for computing *quantities of interest*. Particularly, if a few a coding conventions are followed, the `qi` function can provide transparent, easy-to-read simulation methods.

8.7 Conclusion

The above sections detail the fastest way to develop Zelig models. For the vast majority of applications and external statistical packages, this should suffice. However, at times, more elaborate measures may need to be taken. If this is the case, the API specifications for each particular methods should be read, since a wealth of information has been omitted in order to simplify this tutorial.

For more detailed information, consult the `zelig2`, `param`, and `qi` sections of the Zelig Development manual.

Chapter 9

zelig2: Interfacing External Methods with Zelig

9.1 Introduction

Developers can develop a model, write the model-fitting function, and test it within the Zelig framework without explicit intervention from the Zelig team. This modularity relies on two R programming conventions:

1. **wrappers**, which pass arguments from R functions to other R functions or foreign function calls (such as in C, C++, or Fortran). This step is facilitated by - as will be explained in detail in the upcoming chapter - the `zelig2` function.
2. **classes**, which tell generic functions how to handle objects of a given class. For a statistical model to be compliant with Zelig, the model-fitting function *must* return a classed object.

Zelig implements a unique and simple method for incorporating existing statistical models which lets developers test *within* the Zelig framework *without* any modification of both their own code or the `zelig` function itself. The heart of this procedure is the `zelig2` function, which acts as an interface between the `zelig` function and the existing statistical model. That is, the `zelig2` function maps the user-input from the `zelig` function into input for the existing statistical model's constructor function. Specifically, a Zelig model requires:

1. An existing statistical model, which is invoked through a function call and returns an object
2. A `zelig2` function which maps user-input from the `zelig` function to the existing statistical model
3. A name for the **zelig** model, which can differ from the original name of the statistical model.

9.2 `zelig2` Method Signature

The `zelig2` method's signature typically differs between Zelig models. This is essential, since statistical models generally have a wide-array of available parameters. To accommodate this, the `zelig2` method's signature can be any legal function declaration that adhere to the following guidelines:

1. The `zelig2` method should be simply named `zelig2`*model*, where *model* is the name of the model, that will be used by zelig to reference it
2. The first parameter *must* be titled `formula`
3. The final parameter *must* be titled `data`

4. The ellipsis parameter *must* exist somewhere between the `formula` and `data` parameters
5. Any parameter necessary for use by the external model should be included in the method's parameters

The following is an example taken from the `logit` model:

```
zelig2logit <- function(formula, weights=NULL, ..., data) {
  # ...
}
```

9.3 `zelig2` Return Value

The `zelig2` method's return value should be a list. The return value of the `zelig` method has two reserved keywords:

1. `.function`: the name of the external method[1] as a character-string
2. `data`: the `data.frame` used by the external method to compute the statistical fit
3. **all other keywords without a leading dot** specify a parameter to be passed to the external. This

In addition to these parameters, two other other *optional* reserved keywords exist:

1. `.hook`: a character-string specifying a function to run immediately *after* the external function executes
2. `.post`: a character-string specifying a function to run immediately *before* the `param` method executes

For more details on the `.hook` and `.post` reserved keywords, see Zelig's hook API specification.

9.4 Notable Features of the `zelig2` Method

The `zelig2` method is designed to closely resemble the function call to the external model. That is, this method's parameters and return value should always closely resemble the allowable parameters of the external model's function. As a result, a good rule of thumb is to include the exact parameters from the model in the `zelig2` method, and only remove those that are irrelevant in the developer's Zelig implementation.

9.5 Details in Coding the `zelig2` Method

Typically, the `zelig2` method can be coded in a straightforward manner, needing little additional code aside from its return-value. This may however not be the case for models that contain an atypical style of formula. These types of `formula` include:

- multivariate and multinomial regressions
- regressions containing unique syntax, such as special tags
- regressions that accept lists of formulas

One of the main goals of the Zelig software suite is to unify the language and syntax used in the man disparate statistical models. As a result, Zelig models that fall into the above categories often benefit from the existence of helper functions that convert the Zelig-style formula syntax into that used by the external model. Useful examples of this type of `zelig2` method can be found in both the `mixed.zelig` and `bivariate.zelig` packages.

[1] "The external method" can be any model-fitting function which returns a valid `formula` object, when the `formula` method is called with it as a parameter. That is, basically any R object can be used as a fitted model, so long as it as a valid slot that can be considered a formula.

9.6 Example of a zelig2 Method

The following is an illustrative example taken from the Zelig core package and its explanation.

9.6.1 zelig2logit.R

```
# [1]
zelig2logit <- function(formula, ..., weights=NULL, data)
  list(
       # [2]
       .function = "glm",

       # [3]
       formula = formula,
       data    = data,
       weights = weights,

       # [4]
       family = binomial(link="logit"),
       model  = FALSE,

       # [5]
       ...
       )
```

9.6.2 Explanation of zelig2logit.R

The following correspond to the above example:

[1] The method name and parameter list specify two things:

- the name of the `zelig2` method by naming the function `zelig2`*model*, where `model` is the name of the model being developed
- the list of acceptable parameter to pass to this Zelig model

[2] Specify the name of the external model

[3] Specify parameters that are user-defined. Note that the value of `formula`, `data`, and `weights` vary with user-input, because they are part of the `zelig2` method's signature

[4] Specify parameter that do not vary with user-input.

[5] Specify that any additional parameters to the `zelig2` method be include in the call to the external model

A `zelig2model` function must always return a list as its return value.
The entries of the returned list have the following format:

Chapter 10

param:Generalizing Common Components of Fitted Statistical Models

10.1 Introduction

Several general features - sampling distribution, link function, systematic component, ancillary parameters, etc. - comprise statistical models. These features, while vastly differing between any two given specific models, share features that are easily classifiable, and usually necessary in the simulation of *quantities of interest*. That is, all statistical models have similar traits, and can be simulated using similar methods. Using this fact, the *parameters* class provides a set of functions and data-structures to aid in the planning and implementation of the statistical model.

10.2 Method Signature of `param`

The signature of the `param` method is straightforward and does not vary between differ Zelig models.

```
param.logit <- function (obj, num, ...) {
  # ...
}
```

10.3 Return Value of `param`

The return value of a `param` method is simply a list containing several entries:

simulations A vector or matrix of random draws taken from the model's distribution function. For example, a logit model will take random draws from a Multivariate Normal distribution.

alpha A vector specifying parameters to be passed into the distribution function. Values for this range from scaling factors to statistical means.

fam An optional parameter. *fam* must be an object of class "family". This allows for the implicit specification of the link and link-inverse function. It is recommend that the developer set either this, the link, or the linkinv parameter explicitly. Setting the family object implicitly defines *link* and *linkinv*.

link An optional parameter. *link* must be a function. Setting the link function explicitly is useful for defining arbitrary statistical models. *link* is used primarily to numerically approximate its inverse - a necessary step for simulating *quantities of interest*.

linkinv An optional parameter. *linkinv* must be a function. Setting the link's inverse explicitly allows for faster computations than a numerical approximation provides. If the inverse function is known, it is recommended that this function is explicitly defined.

10.4 Writing the *param* Method

The "param" function of an arbitrary Zelig model draws samples from the model, and describes the statistical model. In practice, this may be done in a variety of fashions, depending upon the complexity of the model

10.4.1 List Method: Returning an Indexed List of Parameters

While the simple method of returning a vector or matrix from a *param* function is extremely simple, it has no method for setting link or link-inverse functions for use within the actual simulation process. That is, it does not provide a clear, easy-to-read method for simulating *quantities of interest*. By returning an indexed list - or a parameters object - the developer can provide clearly labeled and stored link and link-inverse functions, as well as, ancillary parameters.

Example of Indexed List Method with *fam* Object Set

```
param.logit <- function(z, x, x1=NULL, num=num)
  list(
       coef  = mvrnorm(n=num, mu=coef(z), Sigma=vcov(z)),
       alpha = NULL,
       fam   = binomial(link="logit")
       )
```

Explanation of Indexed List with *fam* Object Set Example

The above example shows how link and link-inverse functions (for a "logit" model) can be set using a "family" object. Family objects exist for most statistical models - logit, probit, normal, Gaussian, et cetera - and come preset with values for link and link-inverses. This method does not differ immensely from the simple, vector-only method; however, it allows for the use of several API functions - *link*, *linkinv*, *coef*, *alpha* - that improve the readability and simplicity of the model's implementation.

The *param* function and the *parameters* class offer methods for automating and simplifying a large amount of repetitive and cumbersome code that may come with building the arbitrary statistical model. While both are in principle entirely optional - so long as the *qi* function is well-written - they serve as a means to quickly and elegantly implement Zelig models.

Example of Indexed List Method (with *link* Function) Set

```
param.poisson <- function(z, x, x1=NULL, num=num) {
  list(
       coef = mvrnorm(n=num, mu=coef(z), Sigma=vcov(z)),
       link = log,

       # because ``link'' is set,
       # the next line is purely optional
       linkinv = exp
       )
```

```
}
```

Explanation of Indexed List (with *link* Function) Example

The above example shows how a *parameters* object can be created with by explicitly setting the statistical model's link function. The *linkinv* parameter is purely optional, since Zelig will create a numerical inverse if it is undefined. However, the computation of the inverse is typically slower than non-iterative methods. As a result of this, if the link-inverse is known, it should be set, using the *linkinv* parameter.

The above example can also contain an *alpha* parameter, in order to store important ancillary parameters - mean, standard deviation, gamma-scale, etc. - that would be necessary in the computation of *quantities of interest*.

10.5 Using a *parameters* Object

Typically, a *parameters* object is used within a model's *qi* function. While the developer can typically omit the *param* function and the *parameters* object, it is not recommended. This is because making use of this function can vastly improve readability and functionality of a Zelig model. That is, *param* and *parameters* automate a large amount of repetitive, cumbersome code, and offer allow access to an easy-to-use API.

10.5.1 Example *param* Function

```
qi.logit <- function(z, x, x1=NULL, sim.param=NULL, num=1000) {
  coef <- coef(sim.param)
  inverse <- linkinv(sim.param)

  eta <- coef %*% t(x)
  theta <- link.inverse(eta)

  # et cetera...
}
```

10.5.2 Explanation of Above *qi* Code

The above is a portion of the actual code used to simulate *quantities of interest* for a "logit" model. By using the sim.par object, which is automatically passed into the function if a *param* function is written, *quantities of interest* can be computed extremely generically. The step-by-step process of the above function is as follows:

- Assign the simulations from *param.logit* to the variable "coef"
- Assign the link-inverse from *param.logit* to the variable "inverse"
- Compute η (eta) by matrix-multiplying our simulations with our explanatory results
- Conclude "simulating" the *quantities of interest* by applying the inverse of the link function. The result is a vector whose median is an approximate value of the *quantity of interest* and has a standard deviation that will define the confidence interval around this value

10.6 Future Improvements

In future releases of Zelig, *parameters* will have more API functions to facilitate common operations - sample drawing, matrix-multiplication, et cetera - so that the developer's focus can be exclusively on implementing important components of the model.

Chapter 11

qi: Simulating Quantities of Interest

11.1 Introduction

For any Zelig module, the *qi* function is ultimately the most important piece of code that must be written; it describes the actual process which simulates the *quantities of interest*. Because of the nature of this process - and the gamut of statistical packages and their underlying statistical model - it is rare that the simulation process can be generalized for arbitrary fitted models. Despite this, it is possible to break down the simulation process into smaller steps.

11.2 Notable Features of *qi* Function

The typical *qi* function has several basic procedures:

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 sample provided by 'param' to generate simulations of the *quantities of interest*

   (b) 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

11.3 Basic Layout of a *qi* Function

Now with the general outline of a *qi* function defined, it is important to discuss the expected procedures and specifics of implementation.

11.3.1 The Function's Signature

The *qi* function's signature accepts 4 parameters:

obj: An object of type `zelig`. This wraps the fitted model in the slot "result"

x: An object of type `setx`. This object is used to compute important coefficients, parameters, and features of the data.frame passed to the function call

x1: Also an object of type "*setx*". This object is used in a similar fashion, however its presence allows a variety of *quantities of interest* to be computed. Notably, this is a necessary parameter to compute first-differences

num: The number of simulations to compute

param: An object of type `param`. This is the resulting object from the `param` function, typically containing a variety of important quantities - `simulations`, the `inverse link function`,

11.3.2 Code Example: *qi* Function Signature

```
qi.your_model_name <- function(z, x=NULL, x1=NULL, num=1000) {
# start typing your code here
# ...
# ...
```

Note: In the above example, the function name "qi.your_model_name" is merely a placeholder. In order to register a *qi* function with zelig, the developer must follow the naming convention qi.*your mode name*, where *your_model_name* is the name of the developer's module. For example, if a developer titled his or her zelig module "logit", then the corresponding *qi* function is titled "*qi.logit*".

11.3.3 The Function Body

The function body of *qi* function varies largely from model to model. As a result, it is impossible to create general guidelines to simulate *quantities of interest* - or even determine what the *quantity of interest* is. Typical methods for computing *quantities of interest* include:

- Implementing sampling algorithms based on the underlying fitted model, or
- "Predicting" a large number of values from the fitted model

11.3.4 The Return Value

In order for Zelig to process the simulations, they must be returned in one of several formats:

- ```
  list(
       "TITLE OF QI 1" = val1,
       "TITLE OF QI 2" = val2,
       # any number of title-val pairs
       # ...
       "TITLE OF QI N" = val.n
       )
  ```

- ```
  make.qi(
           titles = list(title1, title2),
           stats  = list(val1, val2)
           )
  ```

-

In the above example,*val1, val2*are data.frames, matrices, or lists representing the simulations of the *quantities of interests*, and *title1, title2* - and any number of titles - are character-strings that will act as human-readable descriptions of the *quantities of interest*. Once results are returned in this format, Zelig will convert the results into a machine-readable format and summarize the simulations into a comprehensible format.

NOTE: Because of its readability, it is suggested that the first method is used when returning *quantities of interest*.

11.4 Simple Example qi function (qi.logit.R)

```
#' simulate quantities of interest for the logit models
qi.logit <- function(z, x=NULL, x1=NULL, y=NULL, num=1000, param=NULL) {

  # compute expected values using the function ".compute.ev"
  ev1 <- .compute.ev(obj, x, num, param)
  ev2 <- .compute.ev(obj, x1, num, param)

  # return simulations of quantities of interest
  list(
       "Expected Values: E(Y|X)"  = ev1,
       "Expected Values (for X1): E(Y|X1)" = ev2,
       "First Differences: E(Y|X1) - E(Y|X)" = ev2 - ev1
       )
}
```

Part V

Zelig Reference Manual

Chapter 12

gamma: Gamma Regression for Continuous, Positive Dependent Variables

12.1 gamma: Gamma Regression for Continuous, Positive Dependent Variables

Use the gamma regression model if you have a positive-valued dependent variable such as the number of years a parliamentary cabinet endures, or the seconds you can stay airborne while jumping. The gamma distribution assumes that all waiting times are complete by the end of the study (censoring is not allowed).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for gamma regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Example

Attach the sample data:

```
> data(coalition)
```

Estimate the model:

```
> z.out <- zelig(duration ~ fract + numst2, model = "gamma", data = coalition)

How to cite this model in Zelig:
  Kosuke Imai, Gary King, and Olivia Lau. 2011.
  "gamma: Gamma Regression for Continuous, Positive Dependent Variables"
  in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software,"
  http://gking.harvard.edu/zelig
```

View the regression output:

```
> summary(z.out)

Call:
glm(formula = duration ~ fract + numst2, family = Function, data = Data.frame,
    model = FALSE)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2510  -0.9112  -0.2278   0.4132   1.5360

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.296e-02  1.329e-02  -0.975  0.33016
fract        1.149e-04  1.723e-05   6.668 1.19e-10 ***
numst2      -1.739e-02  5.881e-03  -2.957  0.00335 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for Gamma family taken to be 0.6291004)

    Null deviance: 300.74  on 313  degrees of freedom
Residual deviance: 272.19  on 311  degrees of freedom
AIC: 2428.1

Number of Fisher Scoring iterations: 6
```

Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

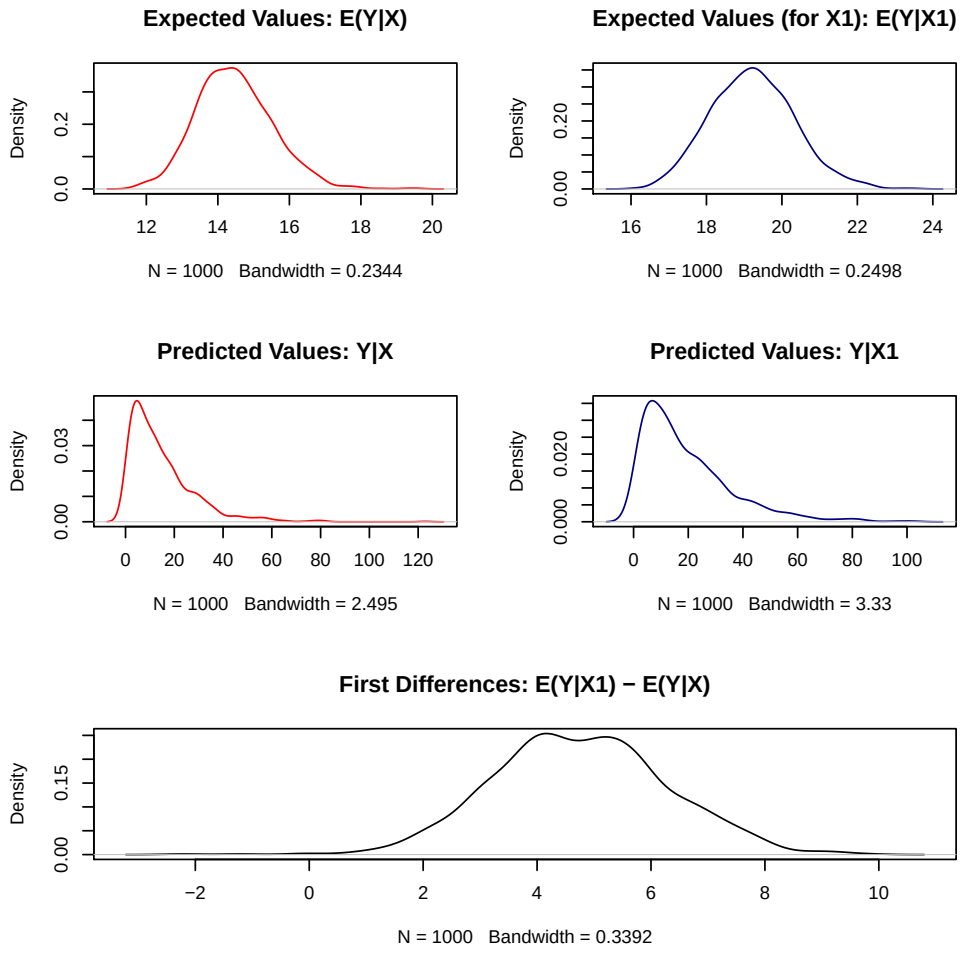
```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

Simulate expected values (`qi$ev`) and first differences (`qi$fd`):

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)

> summary(s.out)
```

```
Model:  gamma
Number of simulations:  1000

Values of X
     fract numst2
1 718.8121      0

Values of X1
     fract numst2
1 718.8121      1


Expected Values: E(Y|X)
  mean    sd    50%   2.5% 97.5%
 14.47 1.069 14.401 12.577 16.73

Expected Values (for X1): E(Y|X1)
   mean    sd    50%   2.5%  97.5%
 19.221 1.105 19.201 17.193 21.495

Predicted Values: Y|X
   mean     sd    50%  2.5%  97.5%
 14.488 13.356 10.896 0.609 50.562

Predicted Values: Y|X1
   mean     sd    50%  2.5%  97.5%
 19.198 16.407 14.441 1.266 62.279

First Differences: E(Y|X1) - E(Y|X)
  mean    sd   50%  2.5% 97.5%
 4.751 1.526 4.716 2.001 7.759
```

> plot(s.out)



Model

- The Gamma distribution with scale parameter α has a *stochastic component*:

$$\begin{aligned} Y &\sim \text{Gamma}(y_i \mid \lambda_i, \alpha) \\ f(y) &= \frac{1}{\alpha^{\lambda_i}\,\Gamma\lambda_i}\, y_i^{\lambda_i - 1} \exp - \left\{\frac{y_i}{\alpha}\right\} \end{aligned}$$

  for $\alpha, \lambda_i, y_i > 0$.

- The *systematic component* is given by

$$\lambda_i = \frac{1}{x_i \beta}$$

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component given draws of α and β from their posteriors:

$$E(Y) = \alpha\lambda_i.$$

- The predicted values (`qi$pr`) are draws from the gamma distribution for each given set of parameters (α, λ_i).
- If `x1` is specified, `sim()` also returns the differences in the expected values (`qi$fd`),

$$E(Y \mid x_1) - E(Y \mid x)$$

.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.
- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.

- `cov.scaled`: a $k \times k$ matrix of scaled covariances.
- `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from a distribution defined by (α, λ_i).
  - `qi$fd`: the simulated first difference in the expected values for the specified values in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Gamma Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37]. Robust standard errors are implemented via the sandwich package by **(author?)** [50]. Sample data are from [25].

Chapter 13

logit: Logistic Regression for Dichotomous Dependent Variables

13.1 `logit`: Logistic Regression for Dichotomous Dependent Variables

Logistic regression specifies a dichotomous dependent variable as a function of a set of explanatory variables.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for logistic regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame) The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Examples

1. Basic Example

   Attaching the sample turnout dataset:

   ```
   > data(turnout)
   ```

   Estimating parameter values for the logistic regression:

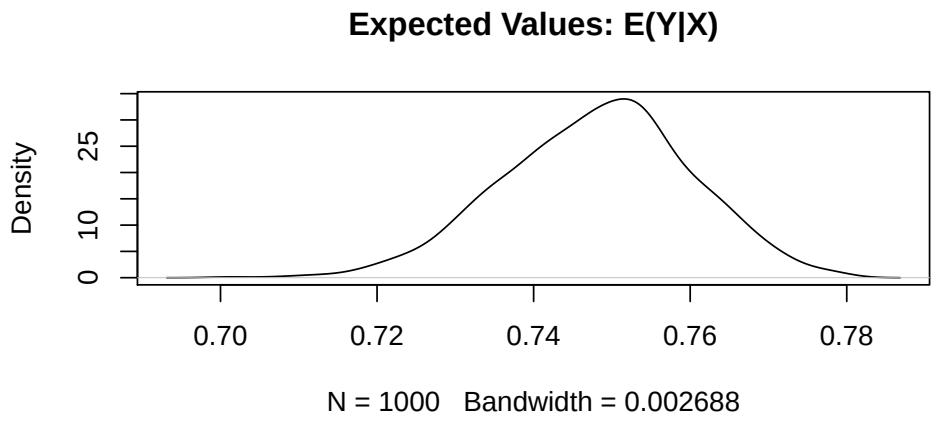
   ```
   > z.out1 <- zelig(vote ~ age + race, model = "logit", data = turnout)
   ```

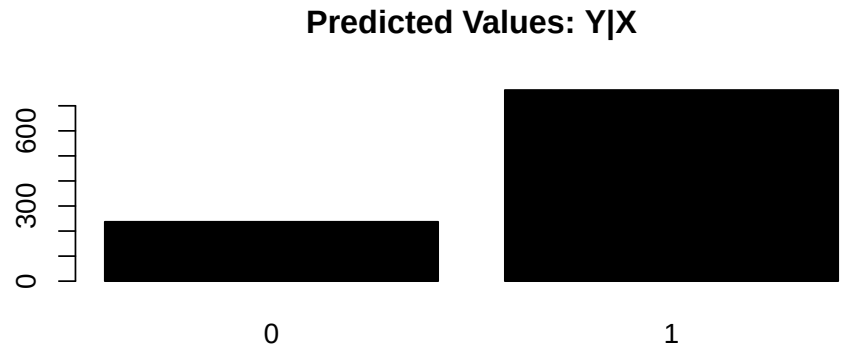
   Setting values for the explanatory variables:

   ```
   > x.out1 <- setx(z.out1, age = 36, race = "white")
   ```

   Simulating quantities of interest from the posterior distribution.

   ```
   > s.out1 <- sim(z.out1, x = x.out1)

   > summary(s.out1)

   > plot(s.out1)
   ```

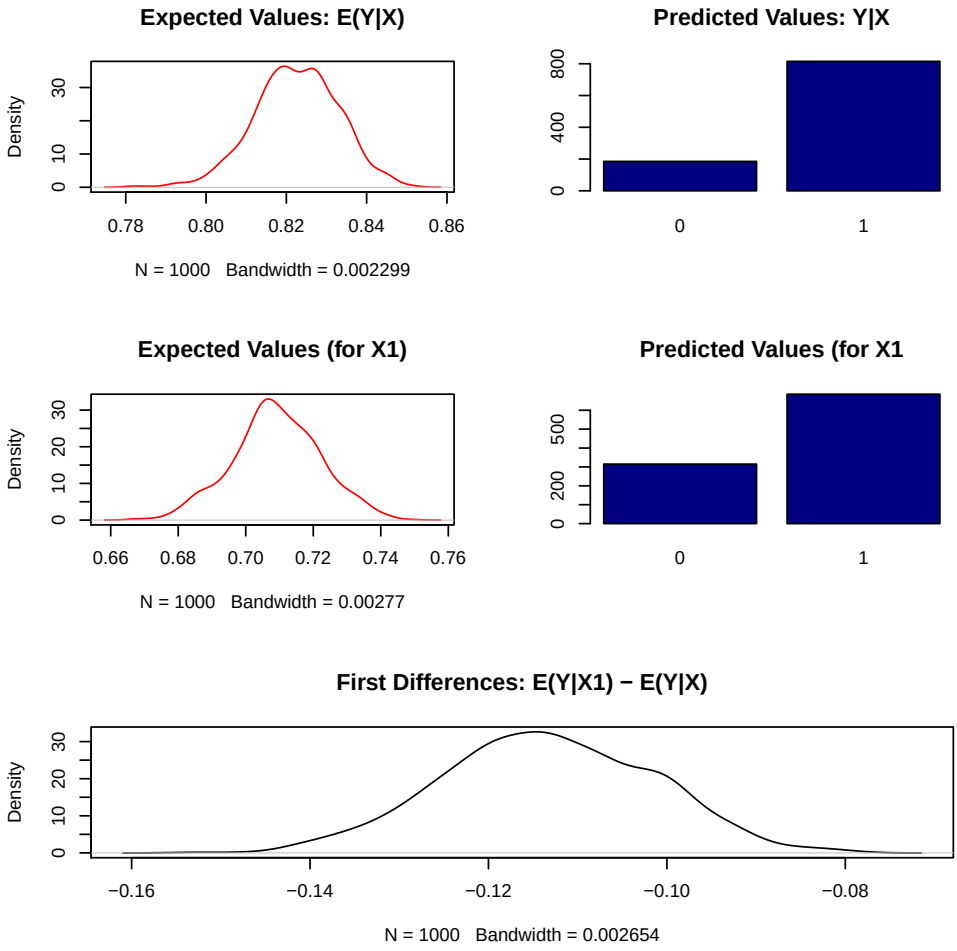




2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> z.out2 <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.high <- setx(z.out2, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out2, educate = quantile(turnout$educate, prob = 0.25))

> s.out2 <- sim(z.out2, x = x.high, x1 = x.low)

> summary(s.out2)

> plot(s.out2)
```

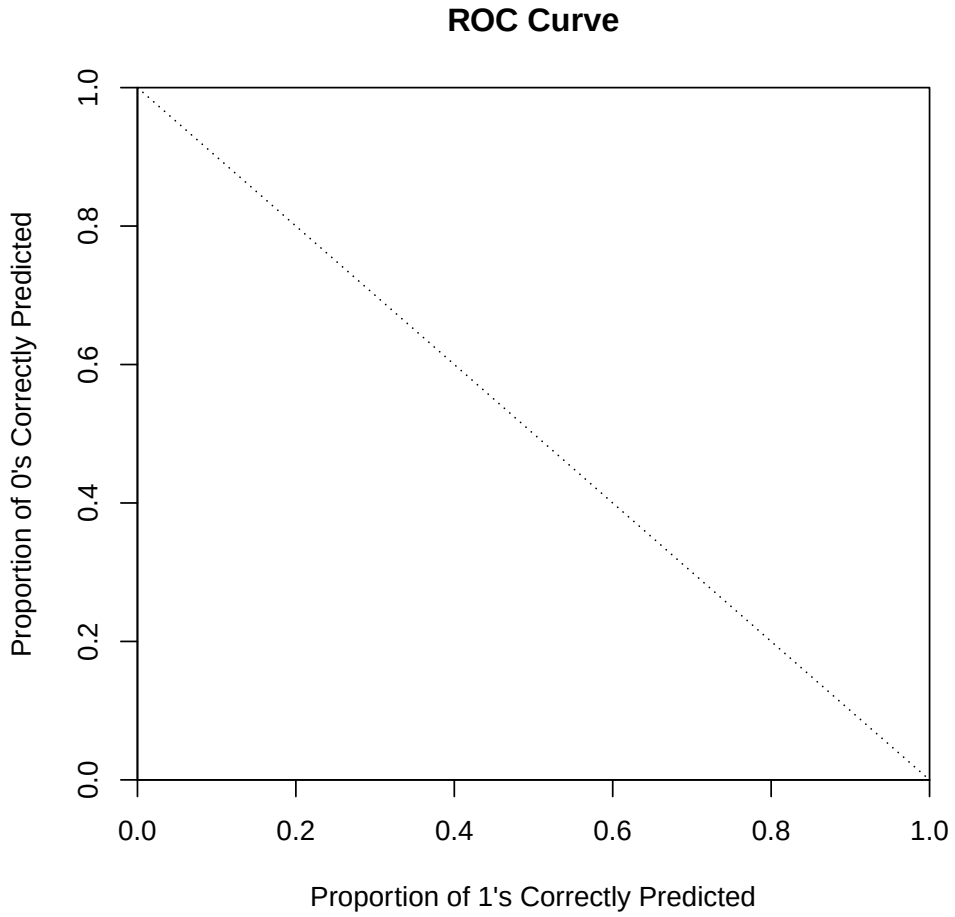


3. Presenting Results: An ROC Plot

   One can use an ROC plot to evaluate the fit of alternative model specifications. (Use `demo(roc)` to view this example, or see King and Zeng (2002).)

```
> z.out1 <- zelig(vote ~ race + educate + age, model = "logit",
+     data = turnout)
> z.out2 <- zelig(vote ~ race + educate, model = "logit", data = turnout)
```

```
> rocplot(z.out1$y, z.out2$y, fitted(z.out1), fitted(z.out2))
```



Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i \mid \pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp(-x_i\beta)}.$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = \frac{1}{1 + \exp(-x_i\beta)},$$

  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \;/\; \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i\beta$

- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the name of the input data frame.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The logit model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37]. Robust standard errors are implemented via the sandwich package by **(author?)** [50]. Sample data are from [25].

Chapter 14

ls: Least Squares Regression for Continuous Dependent Variables

14.1 ls: Least Squares Regression for Continuous Dependent Variables

Use least squares regression analysis to estimate the best linear predictor for the specified dependent variables.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "ls", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for least squares regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors based on sandwich estimators (see [50], [7], and [45]). The default type of robust standard error is heteroskedastic consistent (HC), *not* heteroskedastic and autocorrelation consistent (HAC).

  In addition, `robust` may be a list with the following options:

  - `method`: choose from
    - `"vcovHC"`: (the default if `robust = TRUE`), HC standard errors.
    - `"vcovHAC"`: HAC standard errors without weights.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: only applies to the HAC methods above. Defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a time index (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Examples

1. Basic Example with First Differences

   Attach sample data:

   ```
   > data(macro)
   ```

   Estimate model:

   ```
   > z.out1 <- zelig(unem ~ gdp + capmob + trade, model = "ls", data = macro)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values for the trade variable:

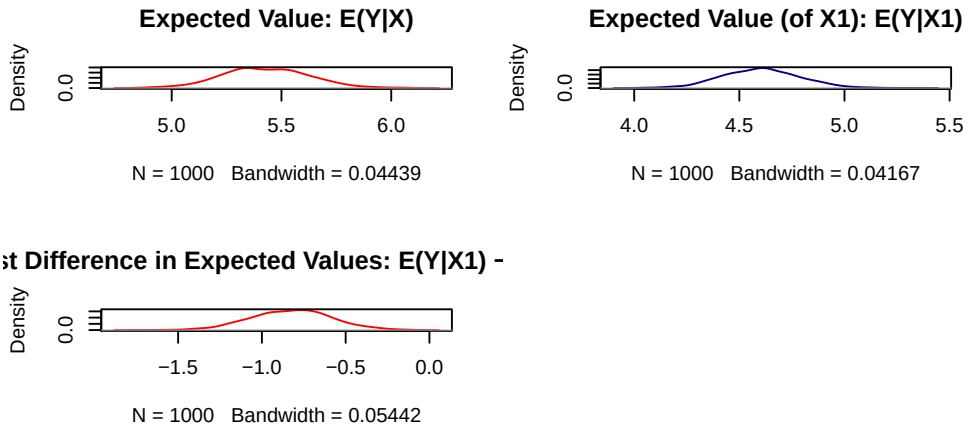
   ```
   > x.high <- setx(z.out1, trade = quantile(macro$trade, 0.8))
   > x.low <- setx(z.out1, trade = quantile(macro$trade, 0.2))
   ```

   Generate first differences for the effect of high versus low trade on GDP:

   ```
   > s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Using Dummy Variables

   Estimate a model with fixed effects for each country (see Section 2 for help with dummy variables). Note that you do not need to create dummy variables, as the program will automatically parse the unique values in the selected variable into discrete levels.

```
> z.out2 <- zelig(unem ~ gdp + trade + capmob + as.factor(country),
+     model = "ls", data = macro)
```

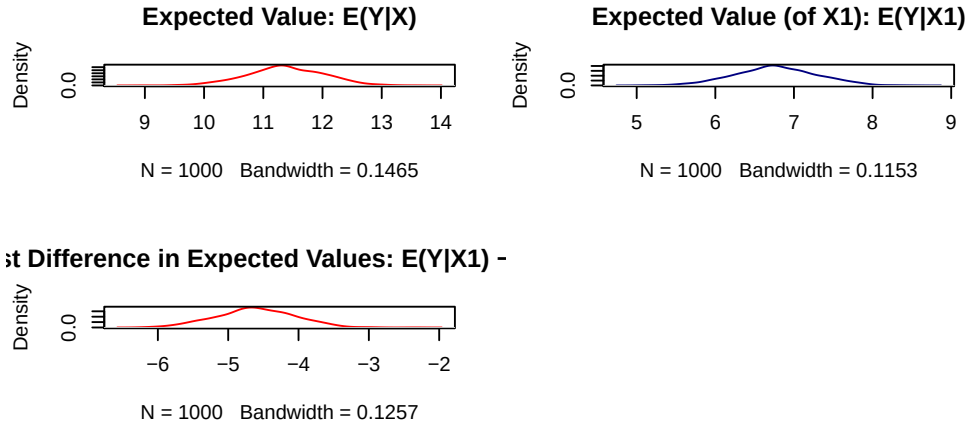
Set values for the explanatory variables, using the default mean/mode values, with country set to the United States and Japan, respectively:

```
> x.US <- setx(z.out2, country = "United States")
> x.Japan <- setx(z.out2, country = "Japan")
```

Simulate quantities of interest:

```
> s.out2 <- sim(z.out2, x = x.US, x1 = x.Japan)
```

```
> plot(s.out2)
```



Model

- The *stochastic component* is described by a density with mean μ_i and the common variance σ^2

$$Y_i \sim f(y_i \mid \mu_i, \sigma^2).$$

- The *systematic component* models the conditional mean as

$$\mu_i = x_i \beta$$

where x_i is the vector of covariates, and β is the vector of coefficients.

The least squares estimator is the best linear predictor of a dependent variable given x_i, and minimizes the sum of squared residuals, $\sum_{i=1}^{n}(Y_i - x_i\beta)^2$.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = x_i\beta,$$

  given a draw of β from its sampling distribution.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "ls", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: fitted values.
  - `df.residual`: the residual degrees of freedom.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.

$$\hat{\beta} = \left(\sum_{i=1}^{n} x_i' x_i \right)^{-1} \sum x_i y_i$$

  - `sigma`: the square root of the estimate variance of the random error e:

$$\hat{\sigma} = \frac{\sum (Y_i - x_i\hat{\beta})^2}{n-k}$$

  - `r.squared`: the fraction of the variance explained by the model.

$$R^2 = 1 - \frac{\sum (Y_i - x_i\hat{\beta})^2}{\sum (y_i - \bar{y})^2}$$

  - `adj.r.squared`: the above R^2 statistic, penalizing for an increased number of explanatory variables.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first differences (or differences in expected values) for the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Least Squares Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The least squares regression is part of the stats package by William N. Venables and Brian D. Ripley [44].In addition, advanced users may wish to refer to `help(lm)` and `help(lm.fit)`.Robust standard errors are implemented via the sandwich package by Achim Zeileis [50].Sample data are from [25].

Chapter 15

negbinom: Negative Binomial Regression for Event Count Dependent Variables

15.1 negbinom: Negative Binomial Regression for Event Count Dependent Variables

Use the negative binomial regression if you have a count of events for each observation of your dependent variable. The negative binomial model is frequently used to estimate over-dispersed event count models.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "negbinom", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for negative binomial regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Example

Load sample data:

```
> data(sanction)
```

Estimate the model:

```
> z.out <- zelig(num ~ target + coop, model = "negbinom", data = sanction)
```

```
> summary(z.out)
```

Set values for the explanatory variables to their default mean values:

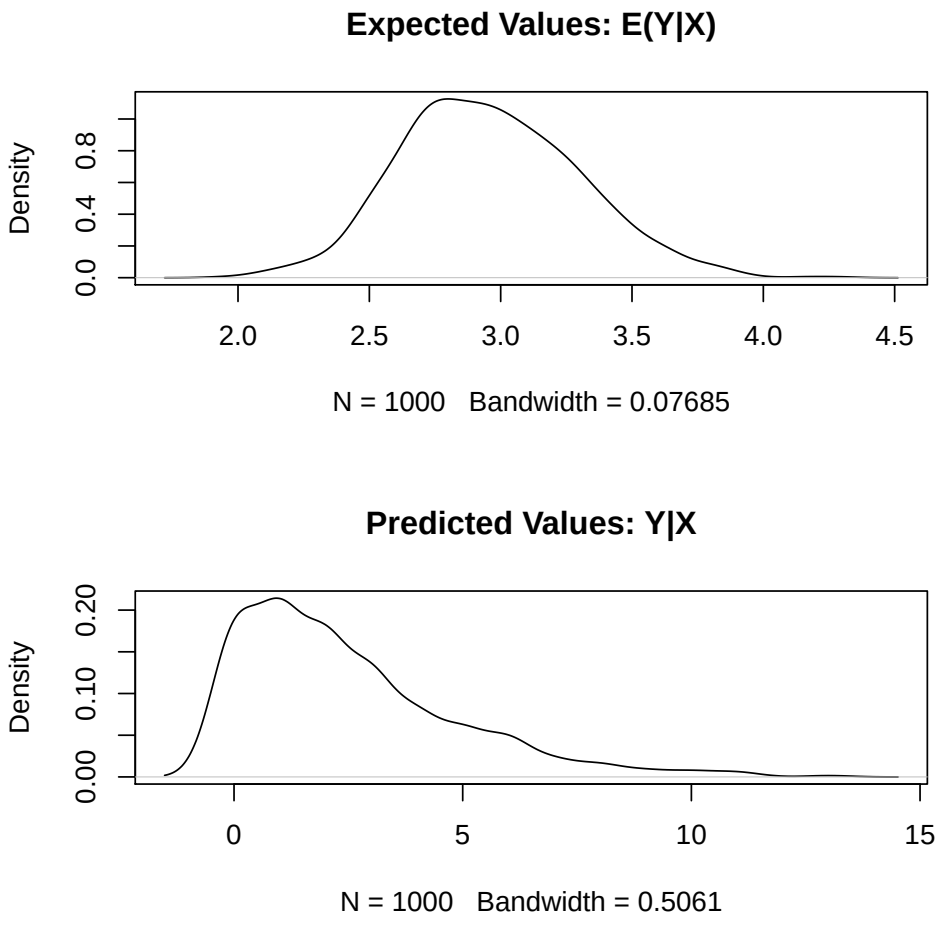
```
> x.out <- setx(z.out)
```

Simulate fitted values:

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer value.

- The negative binomial distribution is derived by letting the mean of the Poisson distribution vary according to a fixed parameter ζ given by the Gamma distribution. The *stochastic component* is given by

$$\begin{aligned} Y_i \mid \zeta_i &\sim \text{Poisson}(\zeta_i \mu_i), \\ \zeta_i &\sim \frac{1}{\theta}\text{Gamma}(\theta). \end{aligned}$$

  The marginal distribution of Y_i is then the negative binomial with mean μ_i and variance $\mu_i + \mu_i^2/\theta$:

$$\begin{aligned} Y_i &\sim \text{NegBinom}(\mu_i, \theta), \\ &= \frac{\Gamma(\theta + y_i)}{y!\,\Gamma(\theta)} \frac{\mu_i^{y_i}\,\theta^{\theta}}{(\mu_i + \theta)^{\theta + y_i}}, \end{aligned}$$

  where θ is the systematic parameter of the Gamma distribution modeling ζ_i.

- The *systematic component* is given by

$$\mu_i = \exp(x_i \beta)$$

  where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component. Thus,

$$E(Y) = \mu_i = \exp(x_i \beta),$$

  given simulations of β.

- The predicted value (`qi$pr`) drawn from the distribution defined by the set of parameters (μ_i, θ).

- The first difference (`qi$fd`) is

$$\text{FD} \;=\; E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "negbinom", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `theta`: the maximum likelihood estimate for the stochastic parameter θ.
  - `SE.theta`: the standard error for `theta`.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the fitted values for the systemic component λ.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, θ).
  - `qi$fd`: the simulated first differences in the simulated expected values given the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Negative Binomial Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The negative binomial model is part of the MASS package by William N. Venable and Brian D. Ripley [44]. Advanced users may wish to refer to `help(glm.nb)` as well as [37]. Robust standard errors are implemented via sandwich package by Achim Zeileis [50].Sample data are from [36].

Chapter 16

normal: Normal Regression for Continuous Dependent Variables

16.1 `normal`: Normal Regression for Continuous Dependent Variables

The Normal regression model is a close variant of the more standard least squares regression model (see Section 14.1). Both models specify a continuous dependent variable as a linear function of a set of explanatory variables. The Normal model reports maximum likelihood (rather than least squares) estimates. The two models differ only in their estimate for the stochastic parameter σ.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for normal regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Examples

1. Basic Example with First Differences

   Attach sample data:

```
> data(macro)
```

   Estimate model:

```
> z.out1 <- zelig(unem ~ gdp + capmob + trade, model = "normal",
+     data = macro)
```

   Summarize of regression coefficients:

```
> summary(z.out1)
```

   Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values for trade:

```
> x.high <- setx(z.out1, trade = quantile(macro$trade, 0.8))
> x.low <- setx(z.out1, trade = quantile(macro$trade, 0.2))
```

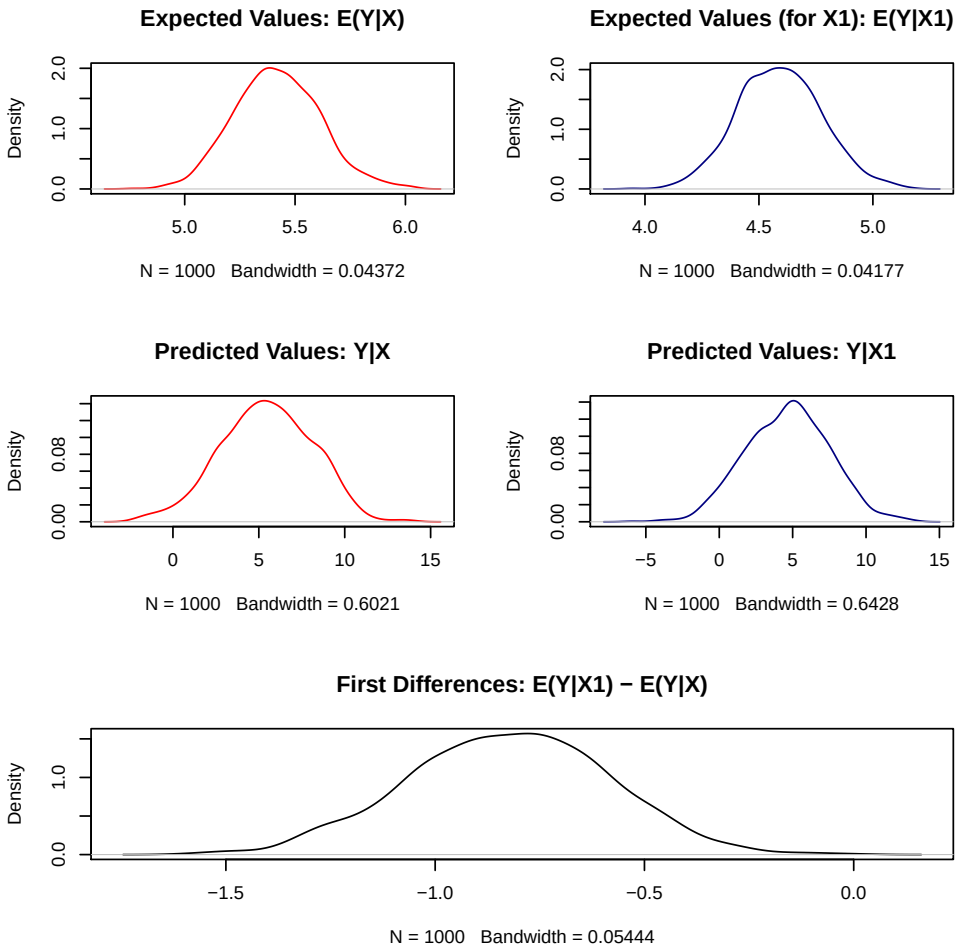
   Generate first differences for the effect of high versus low trade on GDP:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

   A visual summary of quantities of interest:

```
> plot(s.out1)
```



2. Using Dummy Variables

   Estimate a model with a dummy variable for each year and country (see 2 for help with dummy variables). Note that you do not need to create dummy variables, as the program will automatically parse the unique values in the selected variables into dummy variables.

```
> z.out2 <- zelig(unem ~ gdp + trade + capmob + as.factor(year) +
+     as.factor(country), model = "normal", data = macro)
```

   Set values for the explanatory variables, using the default mean/mode variables, with country set to the United States and Japan, respectively: Simulate quantities of interest:

Model

Let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is

$$\mu_i \;=\; x_i\beta,$$

where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the the stochastic component,

$$E(Y) = \mu_i = x_i\beta,$$

given a draw of β from its posterior.

- The predicted value (`qi$pr`) is drawn from the distribution defined by the set of parameters (μ_i, σ).

- The first difference (`qi$fd`) is:

$$\text{FD} \;=\; E(Y \mid x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: fitted values. For the normal model, these are identical to the `linear predictors`.
  - `linear.predictors`: fitted values. For the normal model, these are identical to `fitted.values`.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.

- `df.null`: the residual degrees of freedom for the null model.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, σ).
  - `qi$fd`: the simulated first difference in the simulated expected values for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Normal Regression Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The normal model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37]. Robust standard errors are implemented via the sandwich package by **(author?)** [50]. Sample data are from [25].

Chapter 17

poisson: Poisson Regression for Event Count Dependent Variables

17.1 poisson: Poisson Regression for Event Count Dependent Variables

Use the Poisson regression model if the observations of your dependent variable represents the number of independent events that occur during a fixed period of time (see the negative binomial model, Section 15.1, for over-dispersed event counts.).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for poisson regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Example

Load sample data:

```
> data(sanction)
```

Estimate Poisson model:

```
> z.out <- zelig(num ~ target + coop, model = "poisson", data = sanction)
```

```
> summary(z.out)
```

Set values for the explanatory variables to their default mean values:

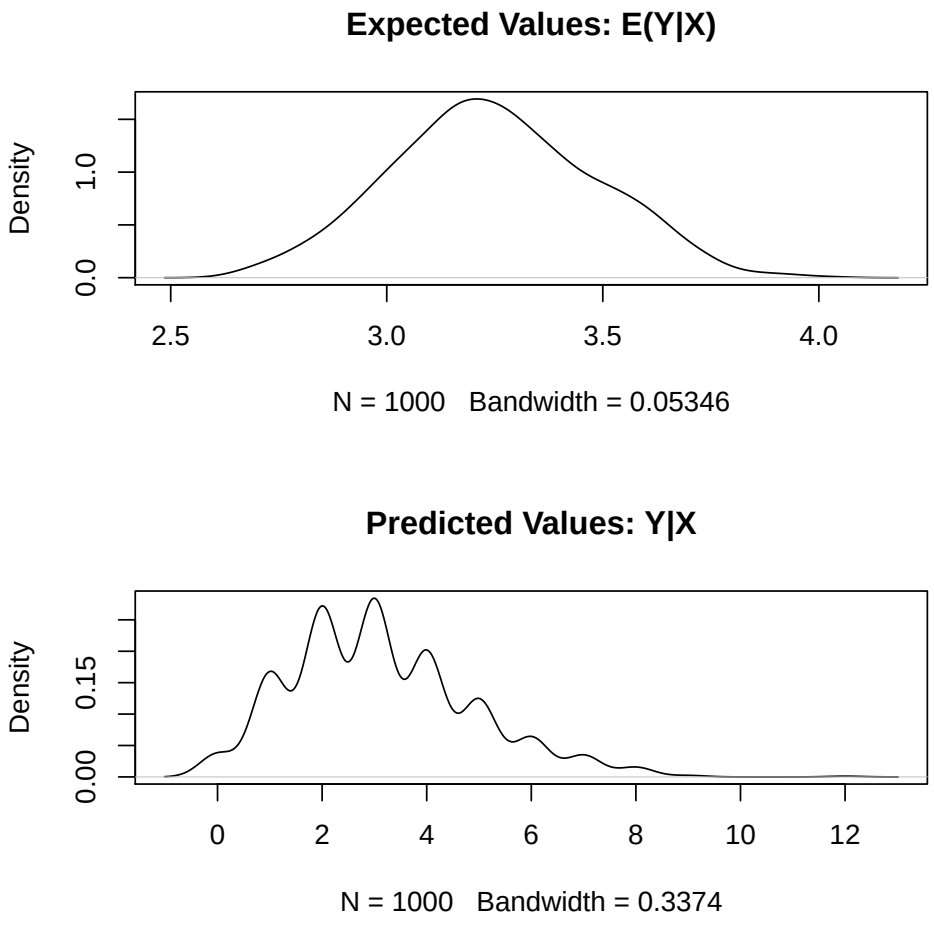
```
> x.out <- setx(z.out)
```

Simulate fitted values:

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is

$$\lambda_i = \exp(x_i\beta),$$

  where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp(x_i\beta),$$

  given draws of β from its sampling distribution.

- The predicted value (`qi$pr`) is a random draw from the poisson distribution defined by mean λ_i.

- The first difference in the expected values (`qi$fd`) is given by:

$$\text{FD} = E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the fitted values for the systemic component λ.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by λ_i.
  - `qi$fd`: the simulated first differences in the expected values given the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Poisson Regression Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The poisson model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37]. Robust standard errors are implemented via the sandwich package by **(author?)** [50]. Sample data are from [36].

Chapter 18

probit: Probit Regression for Dichotomous Dependent Variables

18.1 probit: Probit Regression for Dichotomous Dependent Variables

Use probit regression to model binary dependent variables specified as a function of a set of explanatory variables.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for probit regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see [50]). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in [2].
    - `"weave"`: HAC standard errors using the weights given in [35].
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and [50] for more options.

Examples

Attach the sample turnout dataset:

```
> data(turnout)
```

Estimate parameter values for the probit regression:

```
> z.out <- zelig(vote ~ race + educate, model = "probit", data = turnout)
```

```
> summary(z.out)
```

Set values for the explanatory variables to their default values.

```
> x.out <- setx(z.out)
```

Simulate quantities of interest from the posterior distribution.

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

Model

Let Y_i be the observed binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i),$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is

$$\pi_i = \Phi(x_i\beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance.

Quantities of Interest

- The expected value (`qi$ev`) is a simulation of predicted probability of success

$$E(Y) = \pi_i = \Phi(x_i\beta),$$

  given a draw of β from its sampling distribution.

- The predicted value (`qi$pr`) is a draw from a Bernoulli distribution with mean π_i.

- The first difference (`qi$fd`) in expected values is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) / \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the in-sample fitted values.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the name of the input data frame.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values, or predicted probabilities, for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the predicted probabilities.

- `qi$fd`: the simulated first differences in the predicted probabilities for the values specified in `x` and `x1`.
- `qi$rr`: the simulated risk ratio for the predicted probabilities simulated from `x` and `x1`.
- `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
- `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The probit model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37]. Robust standard errors are implemented via the sandwich package by **(author?)** [50]. Sample data are from [25].

Chapter 19

logit.gam: Generalized Additive Model for Dichotomous Dependent Variables

19.1 `logit.gam`: Generalized Additive Model for Dichotomous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for dichotomous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "logit.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Logit models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.

- `...`: additional options passed to the `logit.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> g <- (f-5)/3
> g <- binomial()$linkinv(g)
> y <- rbinom(g,1,g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

   Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "logit.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

   Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

   Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

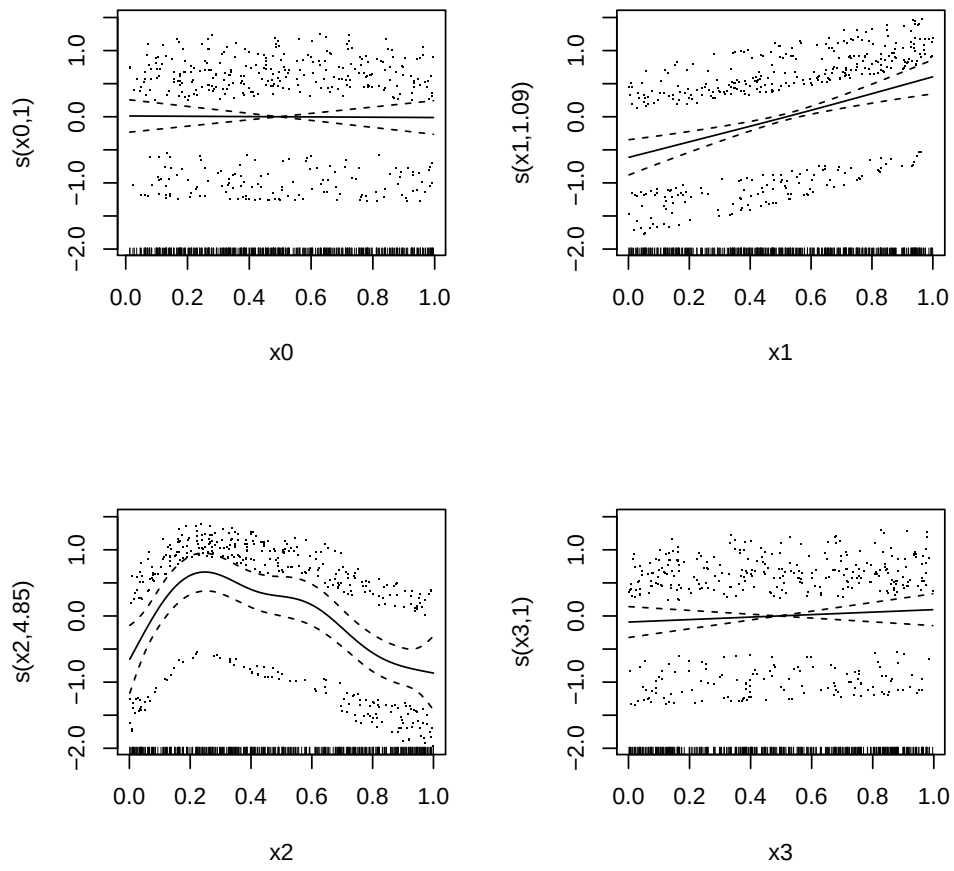
   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```



Set the smoothing parameter for the first term, estimate the rest:

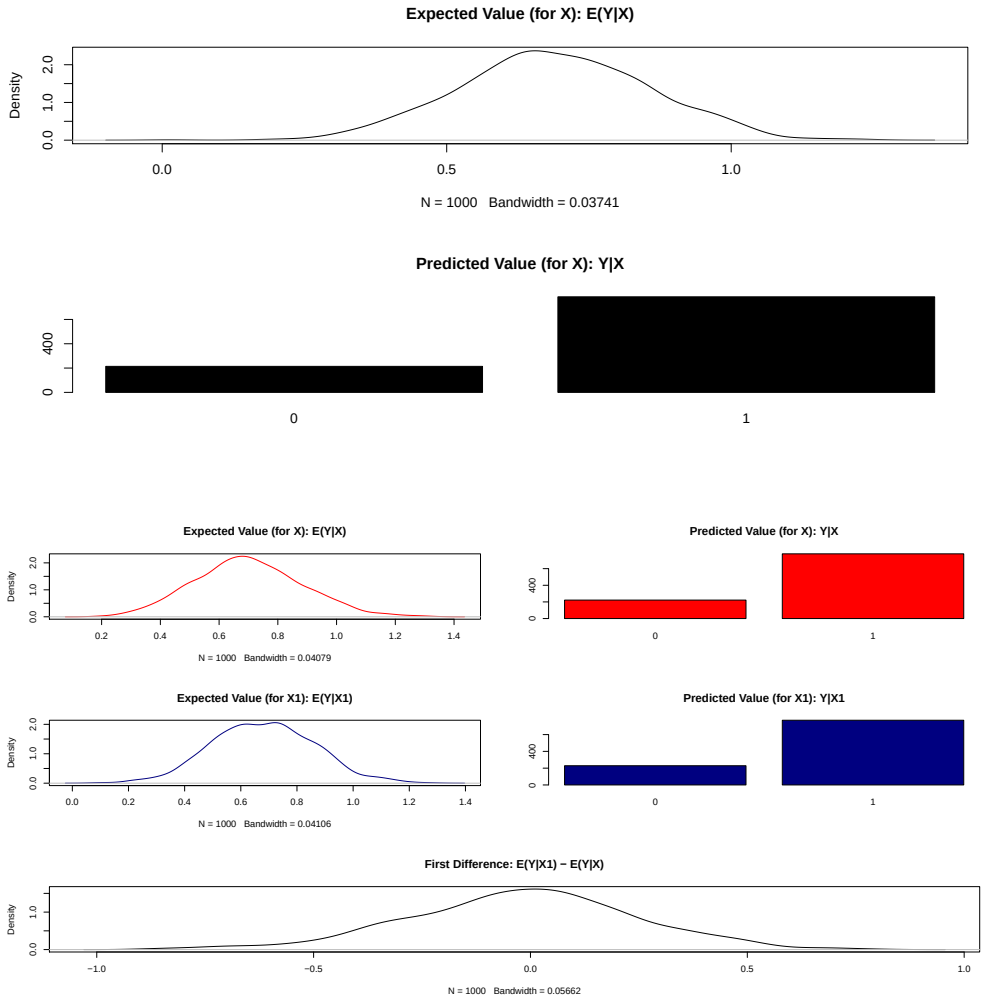
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```



Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `logit.gam` a logistic link function is used. Specifically, let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The logistic distribution has *stochastic component*

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i|\pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp\left(-x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right)},$$

  where x_i is the vector of covariates, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms..

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n-DF)^2}, \tag{19.1}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n}+2s\frac{DF}{n-s}, \tag{19.2}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `logit.gam` model are the same as those for the standard logistic regression.

- The expected value (`qi$ev`) for the `logit.gam` model is the mean of simulations from the stochastic component,
$$\pi_i = \frac{1}{1+\exp\left(-x_i\beta+\sum_{j=1}^{J} f_j(Z_j)\right)},$$
- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.
- The first difference (`qi$fd`) for the logit.gam model is defined as
$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$
for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.

- `fitted.values`: the vector of fitted values for the explanatory variables.
- `residuals`: the working residuals in the final iteration of the IRLS fit.
- `linear.predictors`: the vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `method`: the fitting method used.
- `converged`: logical indicating weather the model converged or not.
- `smooth`: information about the smoothed parameters.
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the input data frame.
- `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite the Logitistic General Additive Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The `logit.gam` model is adapted from the `mgcv` package by Simon N. Wood [48]. Advanced users may wish to refer to `help(gam)`, [47], [46], and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

Chapter 20

normal.gam: Generalized Additive Model for Continuous Dependent Variables

20.1 normal.gam: Generalized Additive Model for Continuous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for continuous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "normal.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Normal models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `normal.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example:

   Create some data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> f0 <- function(x) 2 * sin(pi * x)
> f1 <- function(x) exp(2 * x)
> f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 * (10 *
+     x)^3 * (1 - x)^10
> f3 <- function(x) 0 * x
> f <- f0(x0) + f1(x1) + f2(x2)
> e <- rnorm(n, 0, sig); y <- f + e
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

   Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "normal.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

   Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

   Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

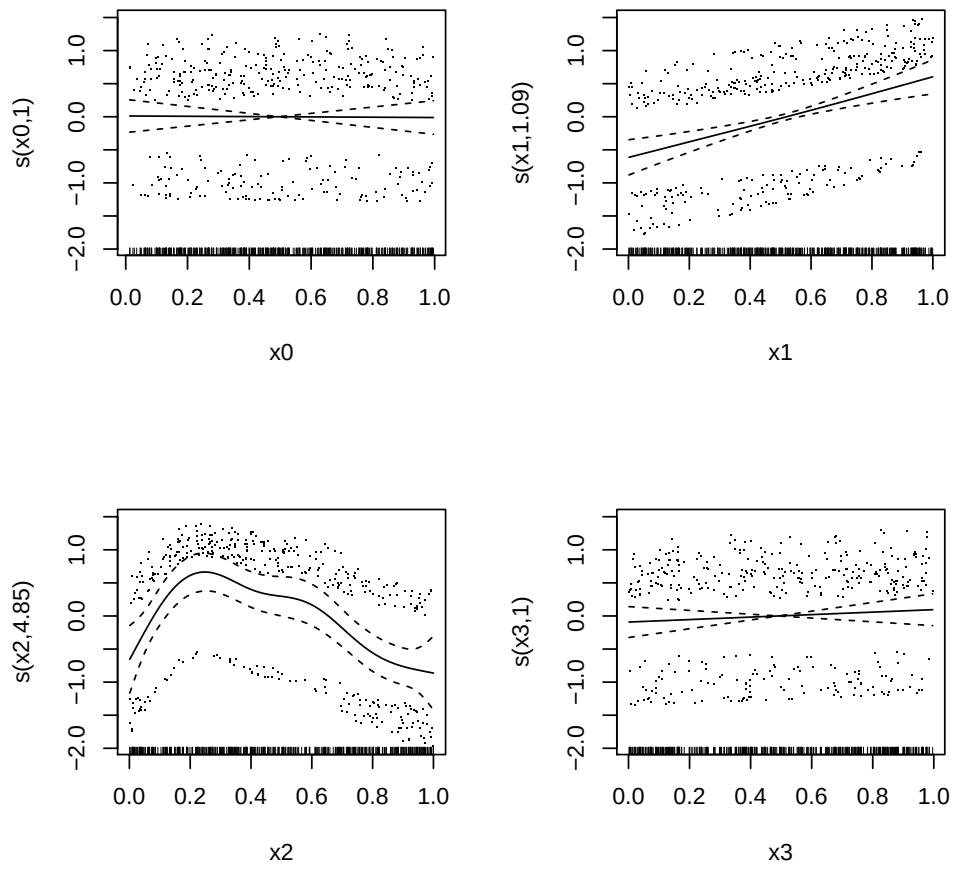
   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```



Set the smoothing parameter for the first term, estimate the rest:

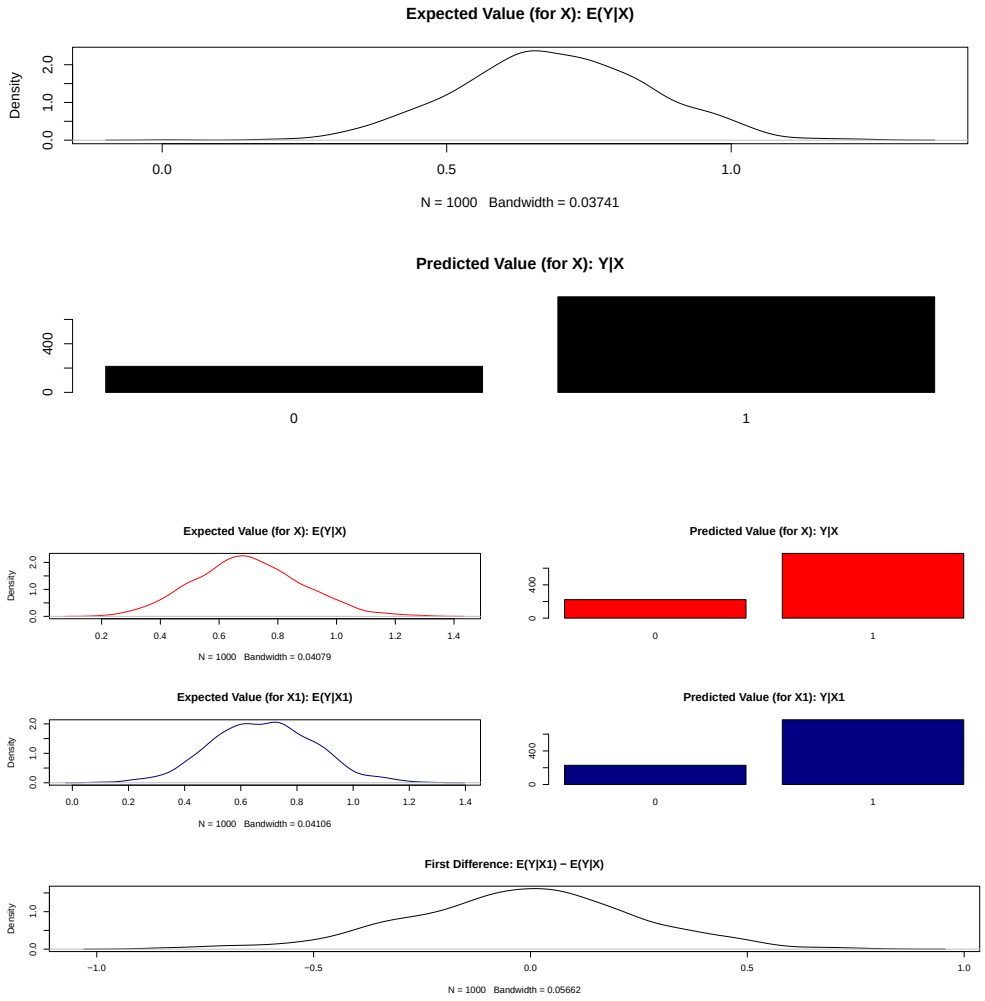
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```



Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `normal.gam` a normal link function is used. Specifically, let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is given by:

$$\mu_i = x_i\beta + \sum_{j=1}^{J} f_j(Z_j).$$

  where x_i is the vector of k explanatory variables, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can

have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n-DF)^2}, \tag{20.1}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n}+2s\frac{DF}{n-s}, \tag{20.2}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `normal.gam` model are the same as those for the standard Normal regression.

- The expected value (`qi$ev`) for the `normal.gam` model is the mean of simulations from the stochastic component,

$$E(Y)=\mu_i=x_i\beta+\sum_{j=1}^{J}f_j(Z_j).$$

- The predicted value (`qi$pr`) is a draw from the Normal distribution defined by the set of parameters (μ_i,σ^2).

- The first difference (`qi$fd`) for the normal.gam model is defined as

$$FD=\Pr(Y|w_1)-\Pr(Y|w)$$

  for $w=\{X,Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.

- `residuals`: the working residuals in the final iteration of the IRLS fit.
- `linear.predictors`: the vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `method`: the fitting method used.
- `converged`: logical indicating weather the model converged or not.
- `smooth`: information about the smoothed parameters.
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the input data frame.
- `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite the Normal General Addtitive Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood [48]. Advanced users may wish to refer to `help(gam)`, [47], [46], and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

Chapter 21

poisson.gam: Generalized Additive Model for Count Dependent Variables

21.1 poisson.gam: Generalized Additive Model for Count Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for count dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "poisson.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Poisson models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `poisson.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> f0 <- function(x) 2 * sin(pi * x)
> f1 <- function(x) exp(2 * x)
> f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 * (10 *
+     x)^3 * (1 - x)^10
> f3 <- function(x) 0 * x
> f <- f0(x0) + f1(x1) + f2(x2)
> g <- exp(f/4);  y <- rpois(rep(1, n), g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

   Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "poisson.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

   Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

   Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

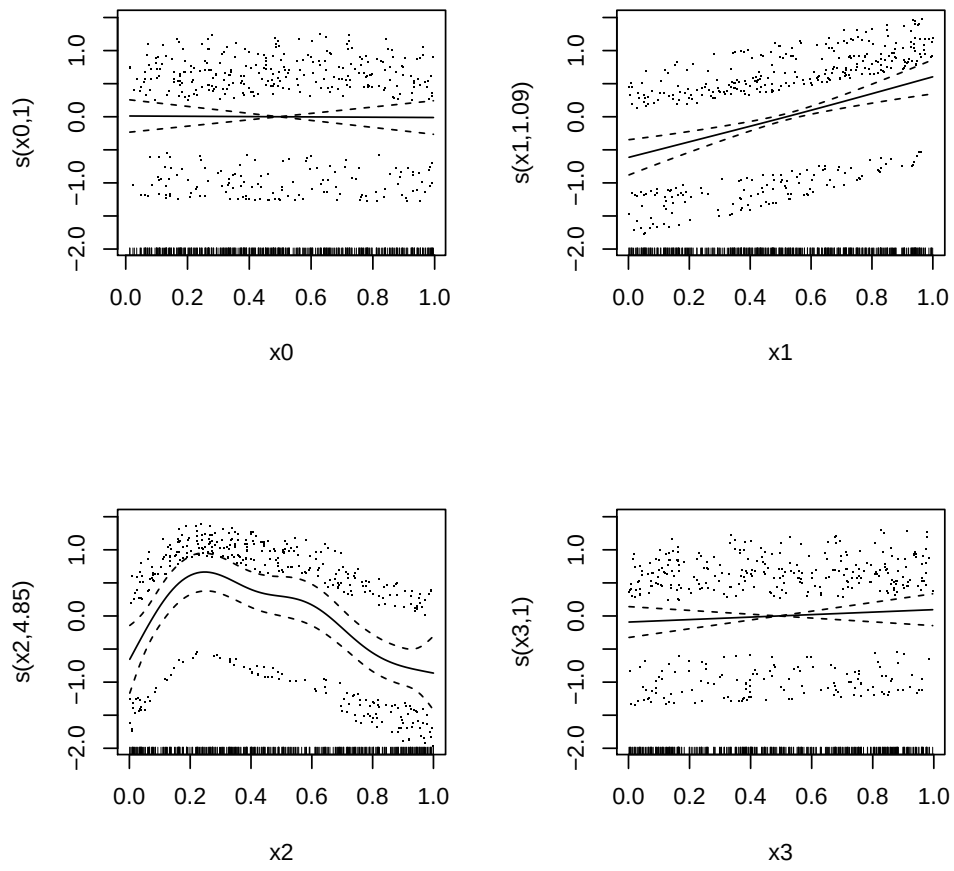
   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```



Set the smoothing parameter for the first term, estimate the rest:

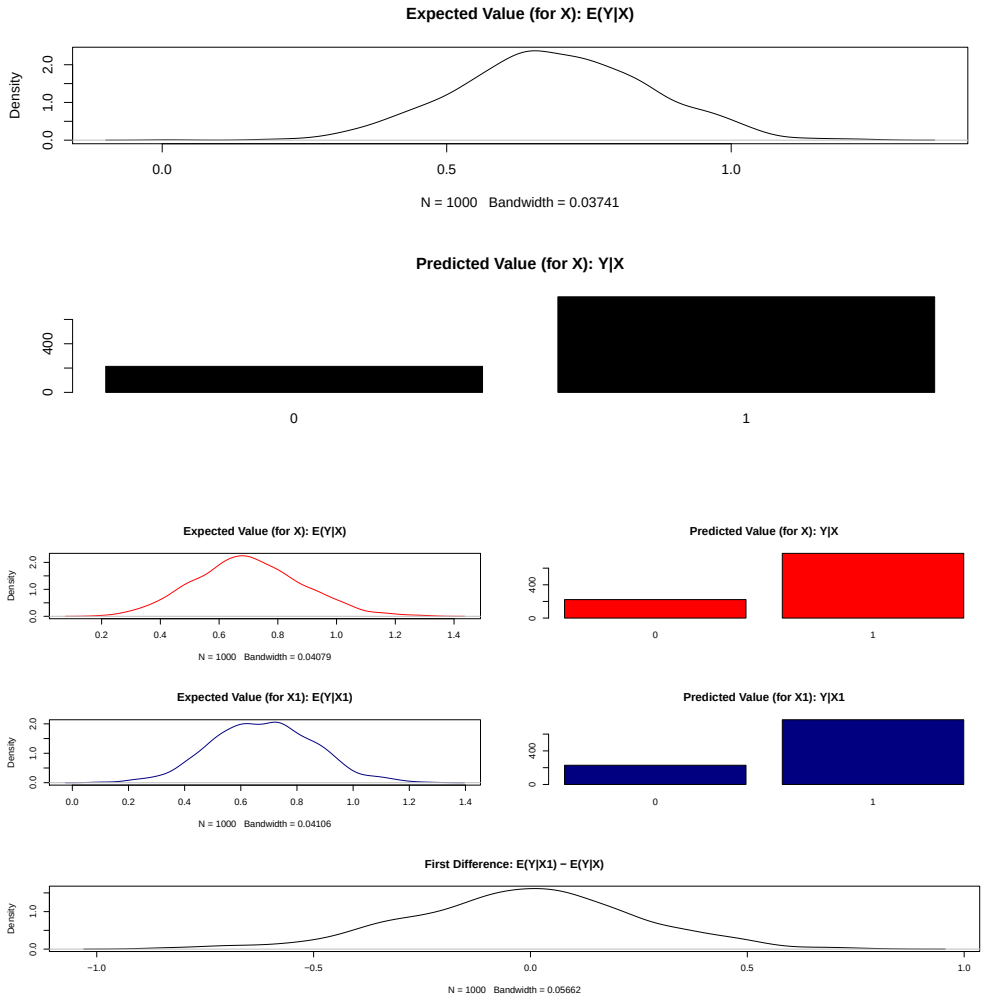
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```



Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `poisson.gam` a Poisson link function is used. Specifically, let Y_i be the dependent variable for observation i. Y_i is thus the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is given by:

$$\lambda_i = \exp\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right).$$

  where x_i is the vector of explanatory variables, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n-DF)^2}, \tag{21.1}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n}+2s\frac{DF}{n-s}, \tag{21.2}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `poisson.gam` model are the same as those for the standard Poisson regression.

- The expected value (`qi$ev`) for the `poisson.gam` model is the mean of simulations from the stochastic component,

$$E(Y)=\lambda_i=\exp\left(x_i\beta\sum_{j=1}^{J}f_j(Z_j)\right).$$

- The predicted value (`qi$pr`) is a random draw from the Poisson distribution defined by mean λ_i.
- The first difference (`qi$fd`) for the poisson.gam model is defined as

$$FD=\Pr(Y|w_1)-\Pr(Y|w)$$

for $w=\{X,Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:

- `coefficients`: parameter estimates for the explanatory variables.
- `fitted.values`: the vector of fitted values for the explanatory variables.
- `residuals`: the working residuals in the final iteration of the IRLS fit.
- `linear.predictors`: the vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `method`: the fitting method used.
- `converged`: logical indicating weather the model converged or not.
- `smooth`: information about the smoothed parameters.
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the input data frame.
- `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite the Poisson General Addtitive Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood [48]. Advanced users may wish to refer to `help(gam)`, [47], [46], and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

Chapter 22

probit.gam: Generalized Additive Model for Dichotomous Dependent Variables

22.1 probit.gam: Generalized Additive Model for Dichotomous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for dichotomous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "probit.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Logit models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.

- `...`: additional options passed to the `probit.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> g <- (f-5)/3
> g <- binomial()$linkinv(g)
> y <- rbinom(g,1,g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

   Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "probit.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

   Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

   Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

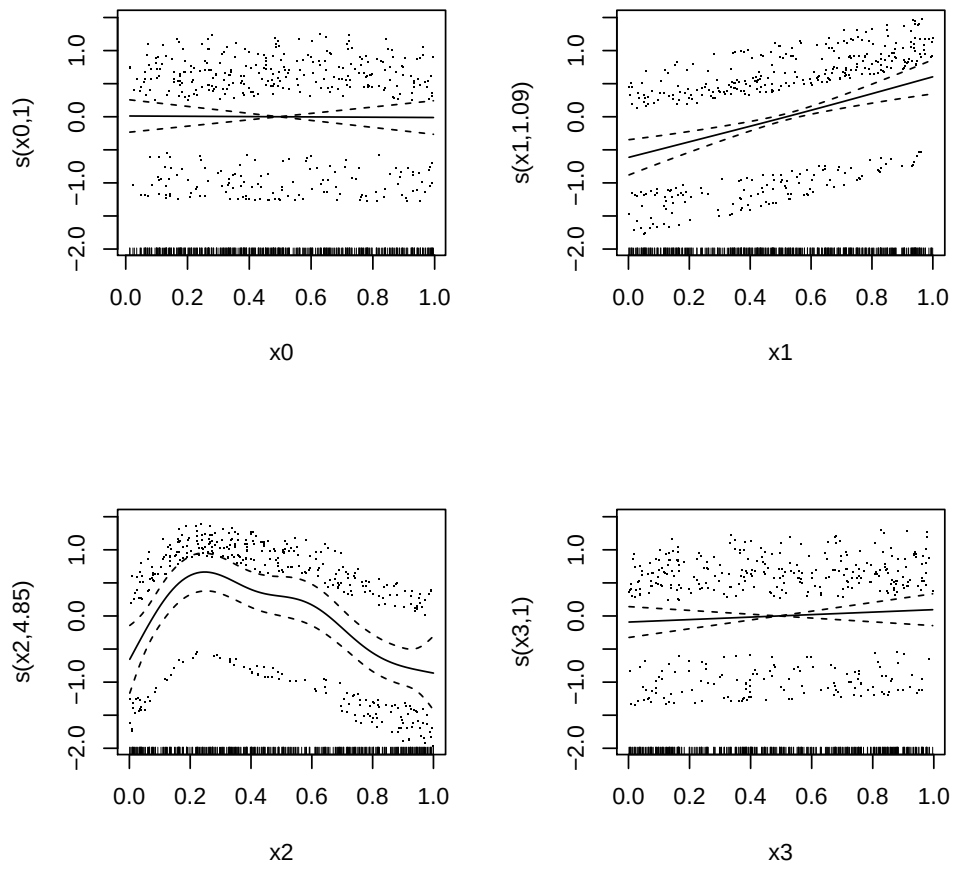
   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```



Set the smoothing parameter for the first term, estimate the rest:

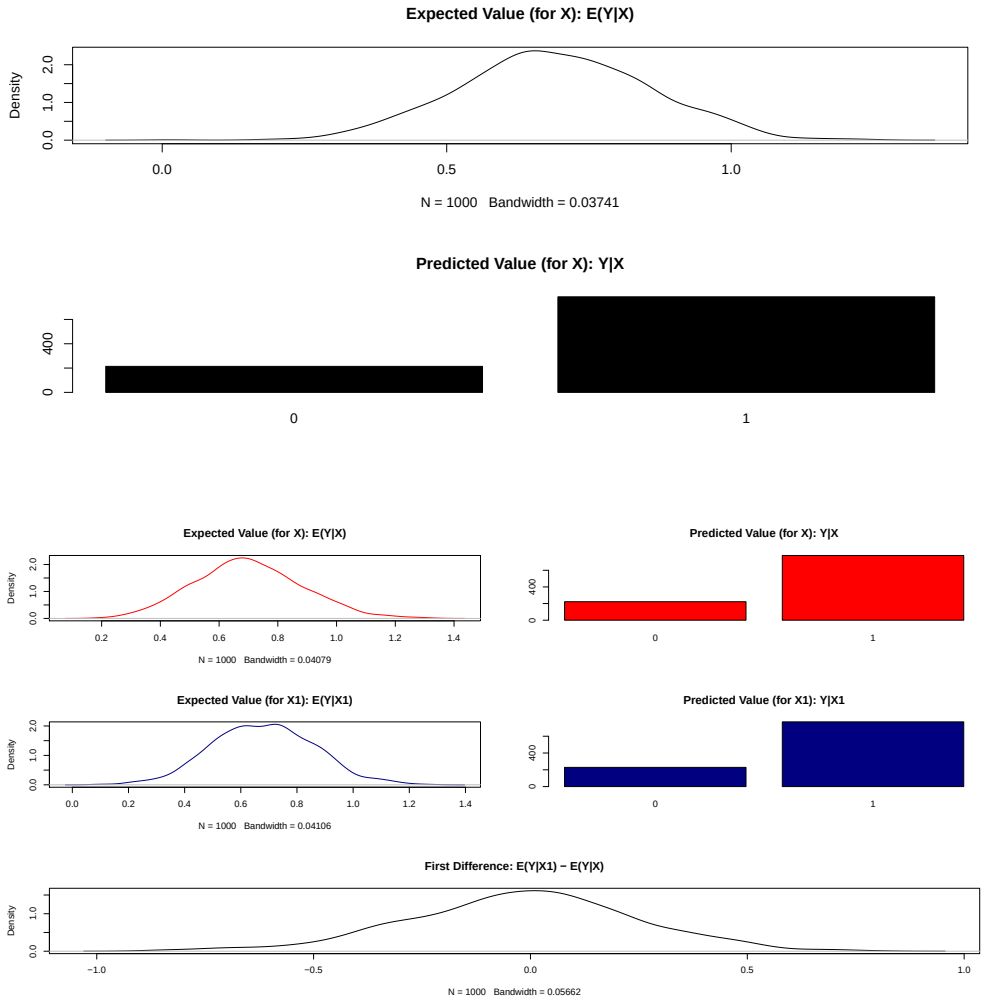
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```



Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `probit.gam` a normal link function is used. Specifically, let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The normal distribution has *stochastic component*

$$Y_i \sim \text{Bernoulli}(\pi_i)$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \mathbf{\Phi}\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right),$$

  where $\mathbf{\Phi}(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n-DF)^2}, \tag{22.1}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n}+2s\frac{DF}{n-s}, \tag{22.2}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `probit.gam` model are the same as those for the standard normal regression.

- The expected value (`qi$ev`) for the `probit.gam` model is the mean of simulations from the stochastic component,

$$\pi_i = \mathbf{\Phi}\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right).$$

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the probit.gam model is defined as

$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$

  for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:

- `coefficients`: parameter estimates for the explanatory variables.
- `fitted.values`: the vector of fitted values for the explanatory variables.
- `residuals`: the working residuals in the final iteration of the IRLS fit.
- `linear.predictors`: the vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `method`: the fitting method used.
- `converged`: logical indicating weather the model converged or not.
- `smooth`: information about the smoothed parameters.
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the input data frame.
- `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite the Probit General Addtitive Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood [48]. Advanced users may wish to refer to `help(gam)`, [47], [46], and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

Chapter 23

gamma.gee: Generalized Estimating Equation for Gamma Regression

23.1 gamma.gee: Generalized Estimating Equation for Gamma Regression

The GEE gamma is similar to standard gamma regression (appropriate when you have an uncensored, positive-valued, continuous dependent variable such as the time until a parliamentary cabinet falls). Unlike in gamma regression, GEE gamma allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. GEE models make no distributional assumptions but require three specifications: a mean function, a variance function, and a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See **(author?)** [51]).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:

– Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard gamma regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

  - Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Exchangeable Dependence

   Attaching the sample turnout dataset:

   ```
   > data(coalition)
   ```

   Sorted variable identifying clusters

   ```
   > coalition$cluster <- c(rep(c(1:62), 5), rep(c(63), 4))
   > sorted.coalition <- coalition[order(coalition$cluster), ]
   ```

   Estimating model and presenting summary:

   ```
   > z.out <- zelig(duration ~ fract + numst2, model = "gamma.gee",
   +     id = "cluster", data = sorted.coalition, robust = TRUE, corstr = "exchangeable")
   > summary(z.out)
   ```

   Setting the explanatory variables at their default values (mode for factor variables and mean for non-factor variables), with numst2 set to the vector 0 = no crisis, 1 = crisis.

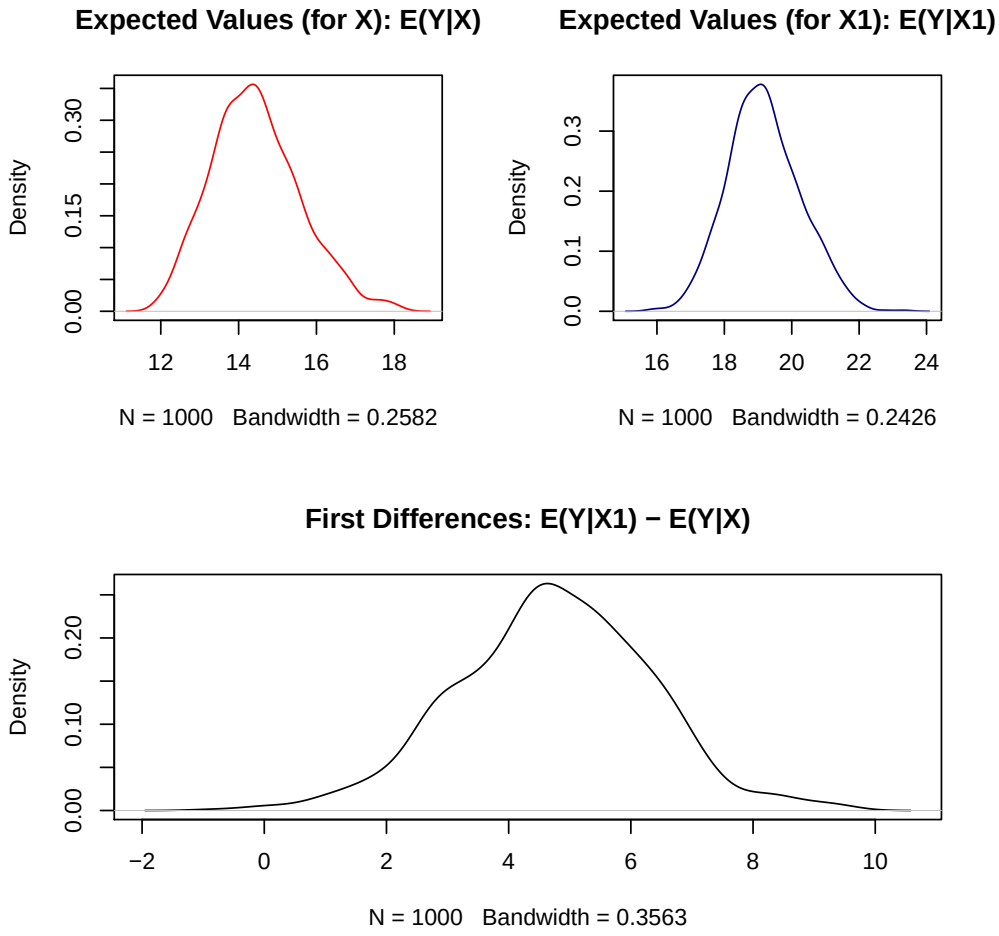
   ```
   > x.low <- setx(z.out, numst2 = 0)
   > x.high <- setx(z.out, numst2 = 1)
   ```

   Simulate quantities of interest

   ```
   > s.out <- sim(z.out, x = x.low, x1 = x.high)
   > summary(s.out)
   ```

   Generate a plot of quantities of interest:

   ```
   > plot(s.out)
   ```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the positive-valued, continuous dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \lambda_i) \\ Y_{it} &\sim g(y_{it} \mid \lambda_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means λ_i and λ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\lambda_{it} = \frac{1}{x_{it}\beta}$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \lambda_{it}^2 = \frac{1}{(x_{it}\beta)^2}$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \lambda_{it}^2$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE gamma model is the mean:

$$E(Y) = \lambda_c = \frac{1}{x_c\beta},$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE gamma model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n}\sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Gamma GEE Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from [22].

Chapter 24

logit.gee: Generalized Estimating Equation for Logit Regression

24.1 `logit.gee`: Generalized Estimating Equation for Logistic Regression

The GEE logit estimates the same model as the standard logistic regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables). Unlike in logistic regression, GEE logit allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See **(author?)** [51]).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:

– Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard logistic regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (Mv=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

  - Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Stationary 3 Dependence

   Attaching the sample turnout dataset:

   ```
   > data(turnout)
   ```

   Variable identifying clusters

   ```
   > turnout$cluster <- rep(c(1:200), 10)
   ```

   Sorting by cluster

   ```
   > sorted.turnout <- turnout[order(turnout$cluster), ]
   ```

   Estimating parameter values for the logistic regression:

   ```
   > z.out1 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
   +     data = sorted.turnout, robust = TRUE, corstr = "stat_M_dep",
   +     Mv = 3)
   ```

   Setting values for the explanatory variables to their default values:

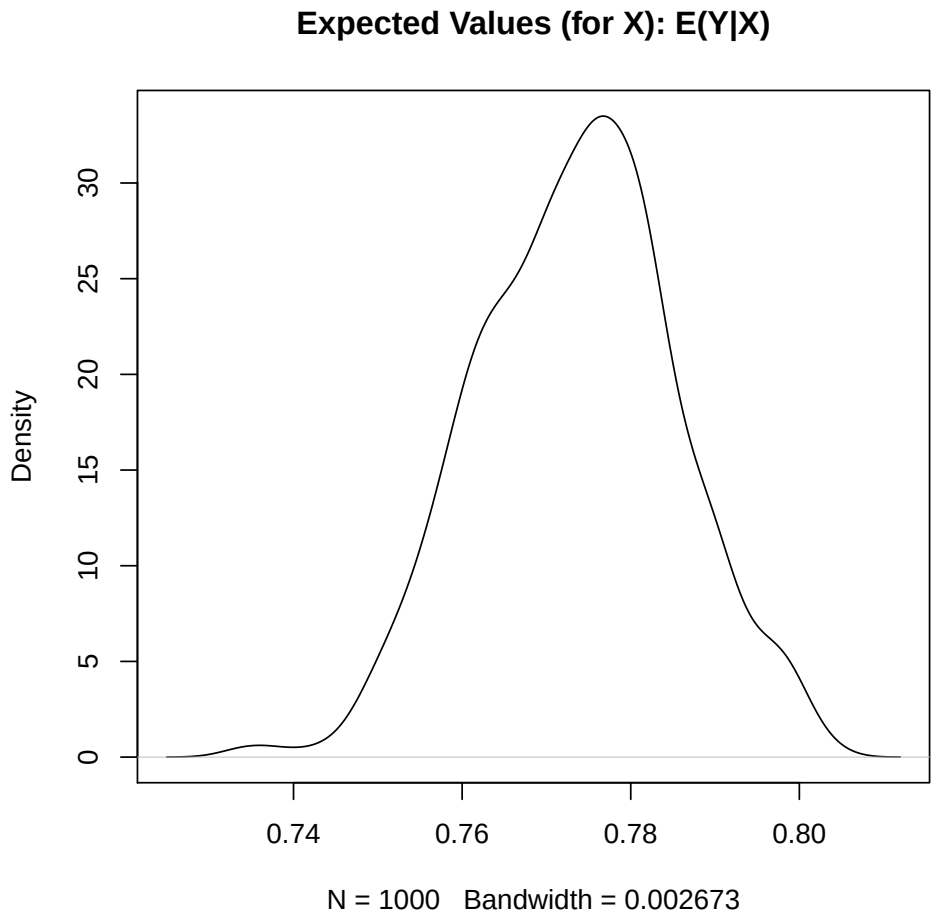
   ```
   > x.out1 <- setx(z.out1)
   ```

   Simulating quantities of interest:

   ```
   > s.out1 <- sim(z.out1, x = x.out1)
   ```

   ```
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Simulating First Differences

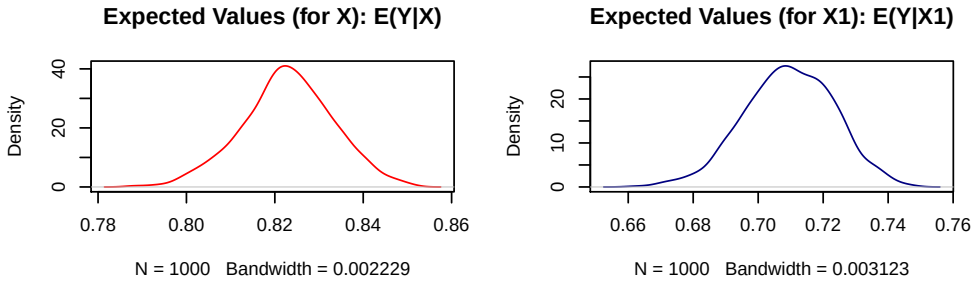
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

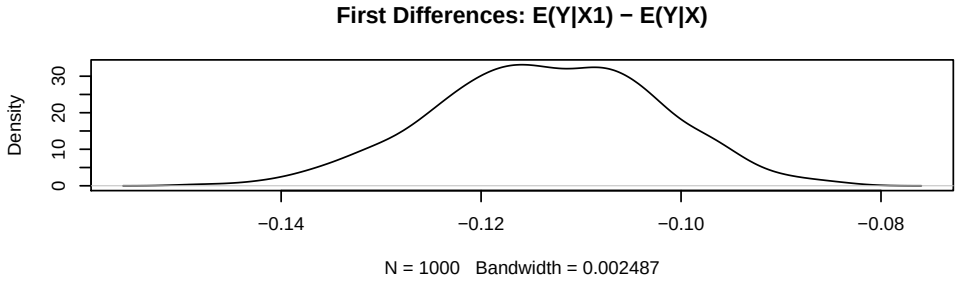
```
> x.high <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.25))
```

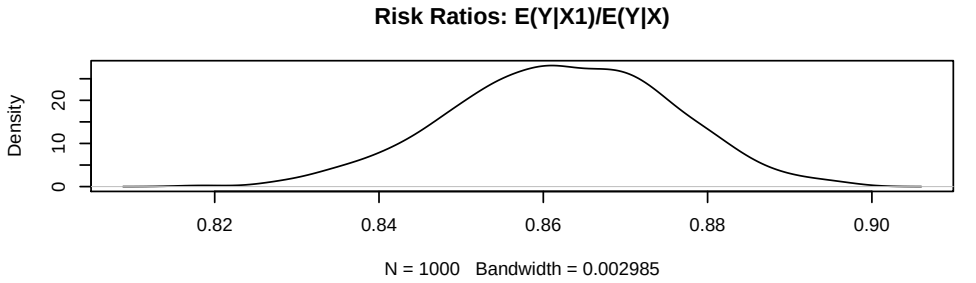
```
> s.out2 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out2)
```

```
> plot(s.out2)
```







3. Example with Fixed Correlation Structure

   User-defined correlation structure

```
> corr.mat <- matrix(rep(0.5, 100), nrow = 10, ncol = 10)
> diag(corr.mat) <- 1
```

   Generating empirical estimates:

```
> z.out2 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
+     data = sorted.turnout, robust = TRUE, corstr = "fixed", R = corr.mat)
```

   Viewing the regression output:

```
> summary(z.out2)
```

The Model

Suppose we have a panel dataset, with Y_{it} denoting the binary dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \pi_i) \\ Y_{it} &\sim g(y_{it} \mid \pi_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means π_i and π_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\pi_{it} = \frac{1}{1 + \exp(-x_{it}\beta)}$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \pi_{it}(1 - \pi_{it})$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \pi_{it}(1 - \pi_{it})$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_c = \frac{1}{1 + \exp(-x_c\beta)},$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \;/\; \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n}\sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it}=1) - E[Y_{it}(tr_{it}=0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Logit GEE Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from [25].

Chapter 25

normal.gee: Generalized Estimating Equation for Normal Regression

25.1 normal.gee: Generalized Estimating Equation for Normal Regression

The GEE normal estimates the same model as the standard normal regression. Unlike in normal regression, GEE normal allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See **(author?)** [51]).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:

– Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard normal regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

  - Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with AR-1 Dependence

   Attaching the sample turnout dataset:

   ```
   > data(macro)
   ```

   Estimating model and presenting summary:

   ```
   > z.out <- zelig(unem ~ gdp + capmob + trade, model = "normal.gee",
   +     id = "country", data = macro, robust = TRUE, corstr = "AR-M",
   +     Mv = 1)
   > summary(z.out)
   ```

   Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values:

   ```
   > x.high <- setx(z.out, trade = quantile(macro$trade, 0.8))
   > x.low <- setx(z.out, trade = quantile(macro$trade, 0.2))
   ```

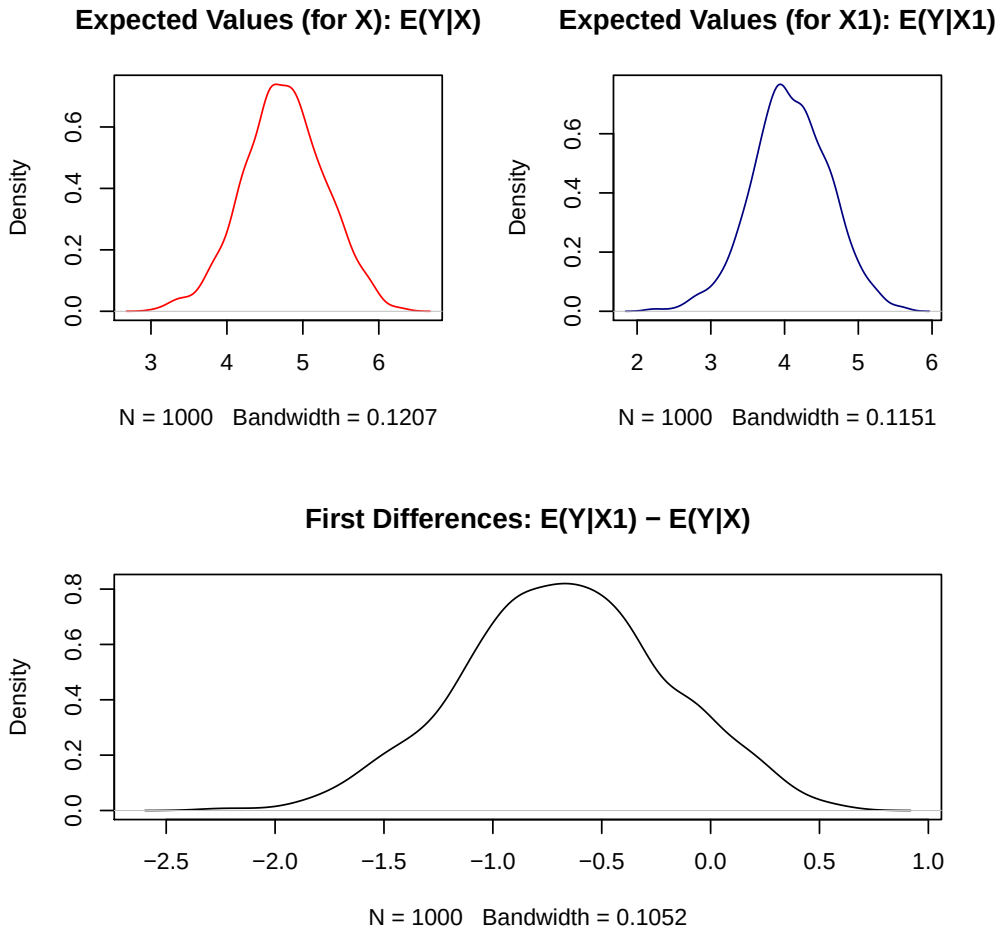
   Generate first differences for the effect of high versus low trade on GDP:

   ```
   > s.out <- sim(z.out, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out)
   ```

   Generate a plot of quantities of interest:

   ```
   > plot(s.out)
   ```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the continuous dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \mu_i) \\ Y_{it} &\sim g(y_{it} \mid \mu_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means μ_i and μ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\mu_{it} = x_{it}\beta$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = 1$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi \, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = 1$ as the tth diagonal element (in the case of GEE normal, A_i is the identity matrix), $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE normal model is the mean of simulations from the stochastic component:

$$E(Y) = \mu_c = x_c \beta,$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE normal model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, μ_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Normal GEE Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from [25].

Chapter 26

poisson.gee: Generalized Estimating Equation for Gamma Regression

26.1 poisson.gee: Generalized Estimating Equation for Poisson Regression

The GEE poisson estimates the same model as the standard poisson regression (appropriate when your dependent variable represents the number of independent events that occur during a fixed period of time). Unlike in poisson regression, GEE poisson allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See **(author?)** [51]).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:

– Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard poisson regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

  - Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Exchangeable Dependence

   Attaching the sample turnout dataset:

   ```
   > data(sanction)
   ```

   Variable identifying clusters

   ```
   > sanction$cluster <- c(rep(c(1:15), 5), rep(c(16), 3))
   ```

   Sorting by cluster

   ```
   > sorted.sanction <- sanction[order(sanction$cluster), ]
   ```

   Estimating model and presenting summary:

   ```
   > z.out <- zelig(num ~ target + coop, model = "poisson.gee", id = "cluster",
   +     data = sorted.sanction, robust = TRUE, corstr = "exchangeable")
   > summary(z.out)
   ```

   Set explanatory variables to their default values:

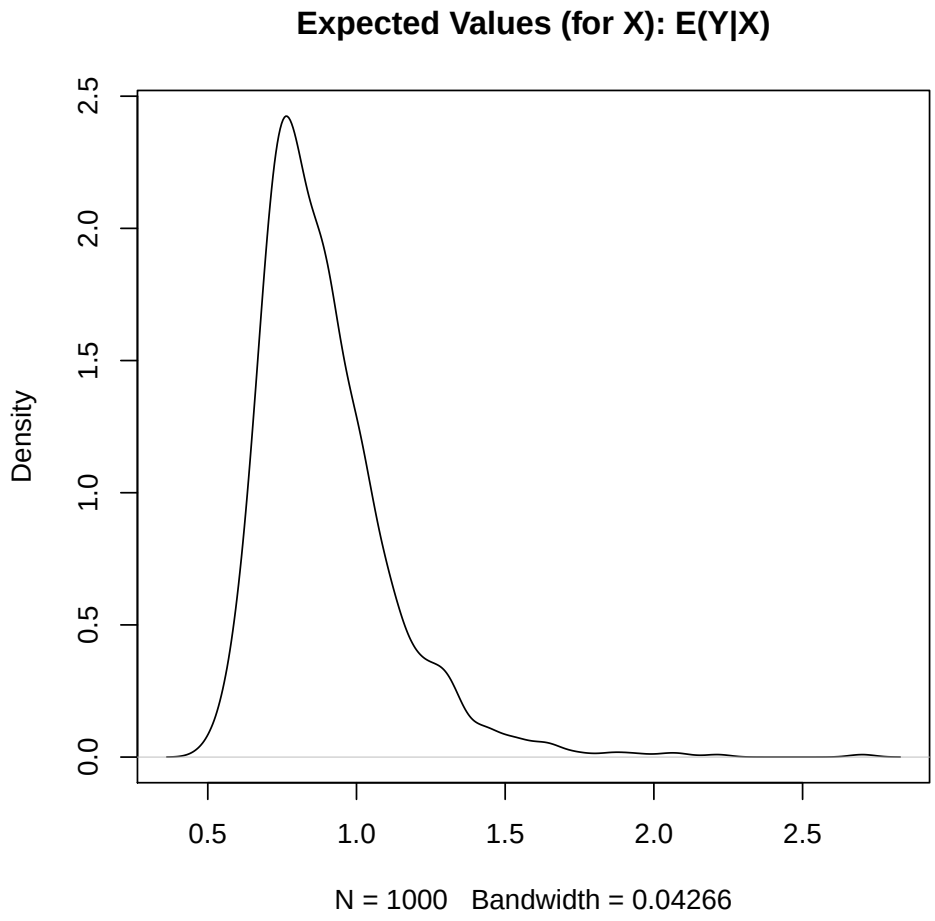
   ```
   > x.out <- setx(z.out)
   ```

   Simulate quantities of interest

   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   ```

   Generate a plot of quantities of interest:

   ```
   > plot(s.out)
   ```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the dependent variable of the number of independent events for a fixed period of time for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \lambda_i) \\ Y_{it} &\sim g(y_{it} \mid \lambda_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means λ_i and λ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\lambda_{it} = \exp(x_{it}\beta)$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \lambda_{it}$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \lambda_{it}$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE poisson model is the mean of simulations from the stochastic component:

$$E(Y) = \lambda_c = \exp(x_c \beta),$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE poisson model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, λ_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Poisson GEE Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from [36]. Please inquire with Lisa Martin before publishing results from these data, as this dataset includes errors that have since been corrected.

Chapter 27

probit.gee: General Estimating Equation for Probit Regression

27.1 probit.gee: Generalized Estimating Equation for Probit Regression

The GEE probit estimates the same model as the standard probit regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables). Unlike in probit regression, GEE probit allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See **(author?)** [51]).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:

– Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard probit regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

  - Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Stationary 3 Dependence

   Attaching the sample turnout dataset:

   ```
   > data(turnout)
   ```

   Variable identifying clusters

   ```
   > turnout$cluster <- rep(c(1:200), 10)
   ```

   Sorting by cluster

   ```
   > sorted.turnout <- turnout[order(turnout$cluster), ]
   ```

   Estimating parameter values:

   ```
   > z.out1 <- zelig(vote ~ race + educate, model = "probit.gee",
   +     id = "cluster", data = sorted.turnout, robust = TRUE, corstr = "stat_M_dep",
   +     Mv = 3)
   ```

   Setting values for the explanatory variables to their default values:

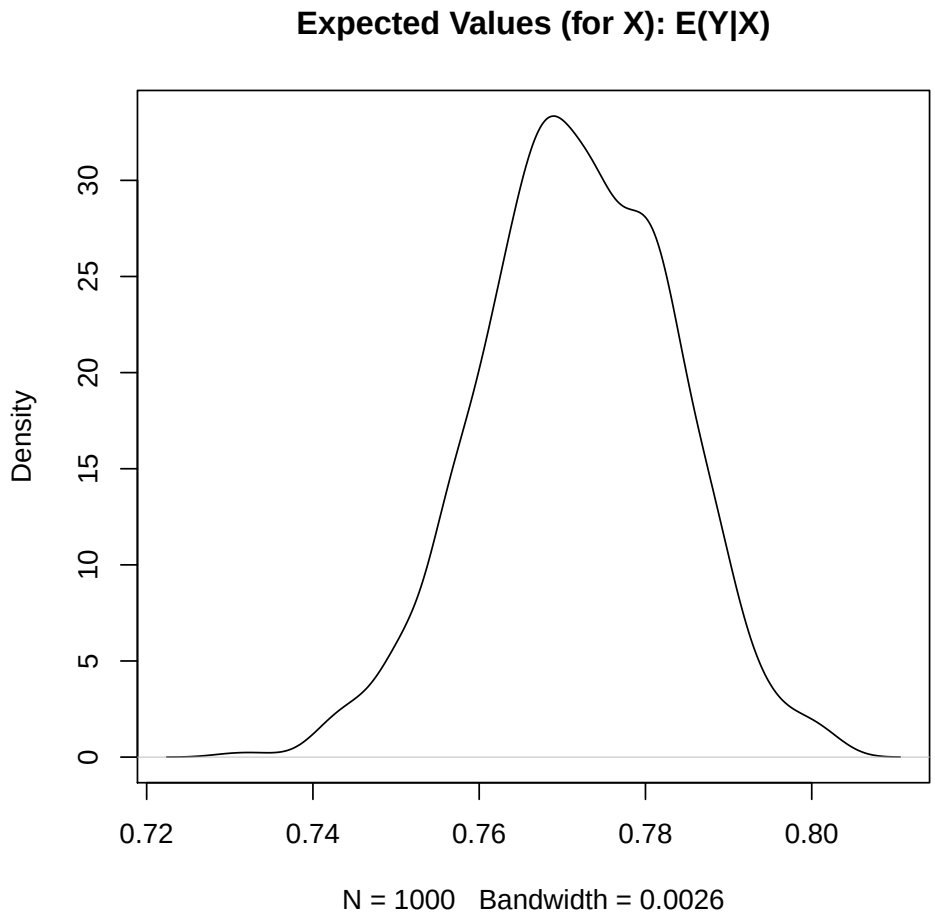
   ```
   > x.out1 <- setx(z.out1)
   ```

   Simulating quantities of interest:

   ```
   > s.out1 <- sim(z.out1, x = x.out1)
   ```

   ```
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Simulating First Differences

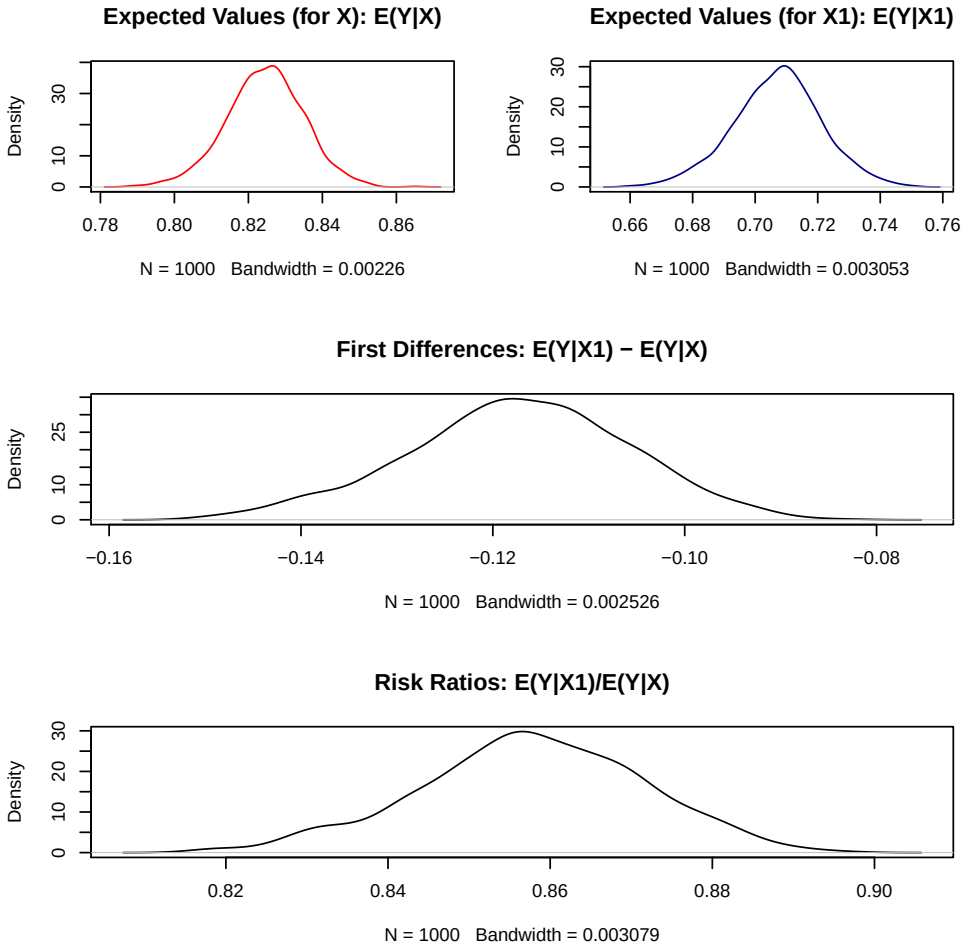
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> x.high <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.25))
```

```
> s.out2 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out2)
```

```
> plot(s.out2)
```



3. Example with Fixed Correlation Structure

   User-defined correlation structure

```
> corr.mat <- matrix(rep(0.5, 100), nrow = 10, ncol = 10)
> diag(corr.mat) <- 1
```

   Generating empirical estimates:

```
> z.out2 <- zelig(vote ~ race + educate, model = "probit.gee",
+     id = "cluster", data = sorted.turnout, robust = TRUE,
+     corstr = "fixed", R = corr.mat)
```

   Viewing the regression output:

```
> summary(z.out2)
```

The Model

Suppose we have a panel dataset, with Y_{it} denoting the binary dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \pi_i) \\ Y_{it} &\sim g(y_{it} \mid \pi_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means π_i and π_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\pi_{it} = \Phi(x_{it}\beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance, x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \pi_{it}(1 - \pi_{it})$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \pi_{it}(1 - \pi_{it})$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE probit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_c = \Phi(x_c\beta),$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE probit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \;/\; \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n}\sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it}=1) - E[Y_{it}(tr_{it}=0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite the Gamma GEE Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from [25].

Chapter 28

bprobit: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

28.1 bprobit: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

Use the bivariate probit regression model if you have two binaryrun dependent variables (Y_1, Y_2), and wish to model them jointly as a function of some explanatory variables. Each pair of dependent variables (Y_{i1}, Y_{i2}) has four potential outcomes, $(Y_{i1} = 1, Y_{i2} = 1)$, $(Y_{i1} = 1, Y_{i2} = 0)$, $(Y_{i1} = 0, Y_{i2} = 1)$, and $(Y_{i1} = 0, Y_{i2} = 0)$. The joint probability for each of these four outcomes is modeled with three systematic components: the marginal $\Pr(Y_{i1} = 1)$ and $\Pr(Y_{i2} = 1)$, and the correlation parameter ρ for the two marginal distributions. Each of these systematic components may be modeled as functions of (possibly different) sets of explanatory variables.

Syntax

```
> z.out <- zelig(list(mu1 = Y1 ~ X1 + X2,
                      mu2 = Y2 ~ X1 + X3,
                      rho = ~ 1),
                model = "bprobit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Input Values

In every bivariate probit specification, there are three equations which correspond to each dependent variable (Y_1, Y_2), and the correlation parameter ρ. Since the correlation parameter does not correspond to one of the dependent variables, the model estimates ρ as a constant by default. Hence, only two formulas (for μ_1 and μ_2) are required. If the explanatory variables for μ_1 and μ_2 are the same and effects are estimated separately for each parameter, you may use the following short hand:

```
> fml <- list(cbind(Y1, Y2) ~ X1 + X2)
```

which has the same meaning as:

```
> fml <- list(mu1 = Y1 ~ X1 + X2, mu2 = Y2 ~ X1 + X2, rho = ~1)
```

You may use the function `tag()` to constrain variables across equations. The `tag()` function takes a variable and a label for the effect parameter. Below, the constrained effect of `x3` in both equations is called the `age` parameter:

```
> fml <- list(mu1 = y1 ~ x1 + tag(x3, "age"), mu2 = y2 ~ x2 + tag(x3,
+     "age"))
```

You may also constrain different variables across different equations to have the same effect.

Examples

1. Basic Example

   Load the data and estimate the model:

   ```
   > data(sanction)
   ```

   ```
   > z.out1 <- zelig(cbind(import, export) ~ coop + cost + target,
   +     model = "bprobit", data = sanction)
   ```

   By default, `zelig()` estimates two effect parameters for each explanatory variable in addition to the correlation coefficient; this formulation is parametrically independent (estimating unconstrained effects for each explanatory variable), but stochastically dependent because the models share a correlation parameter.

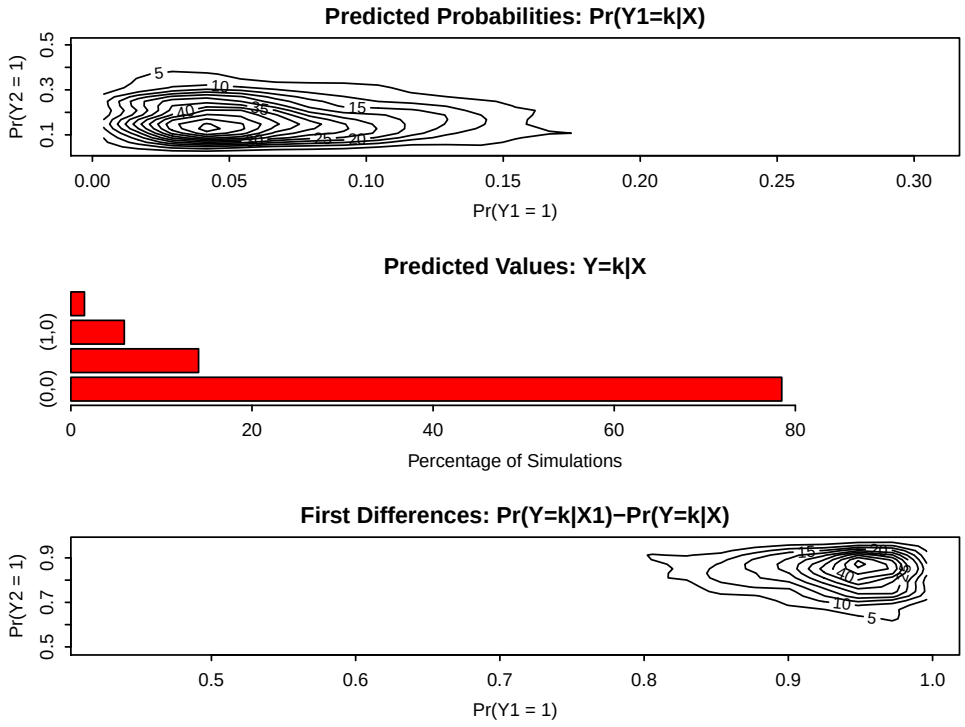
   Generate baseline values for the explanatory variables (with cost set to 1, net gain to sender) and alternative values (with cost set to 4, major loss to sender):

   ```
   > x.low <- setx(z.out1, cost = 1)
   > x.high <- setx(z.out1, cost = 4)
   ```

   Simulate fitted values and first differences:

   ```
   > s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Joint Estimation of a Model with Different Sets of Explanatory Variables

   Using the sample data `sanction`, estimate the statistical model, with `import` a function of `coop` in the first equation and `export` a function of `cost` and `target` in the second equation:

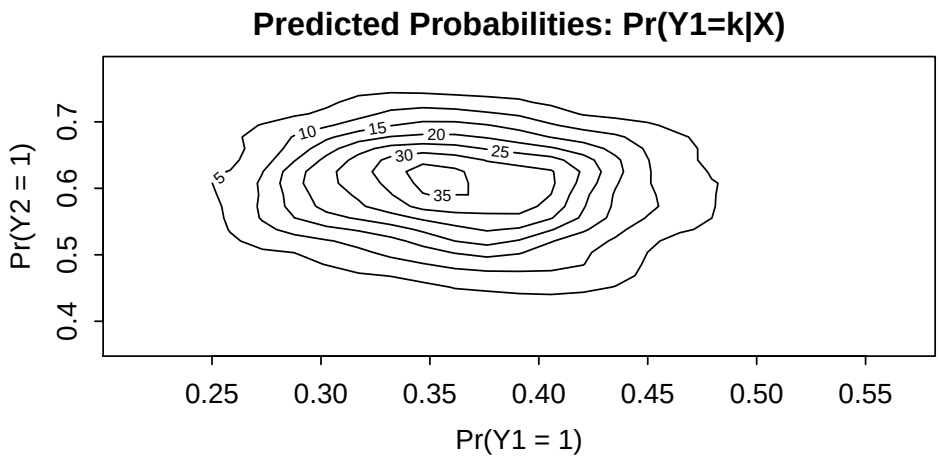
```
> fml2 <- list(mu1 = import ~ coop, mu2 = export ~ cost + target)

> z.out2 <- zelig(fml2, model = "bprobit", data = sanction)
> summary(z.out2)
```

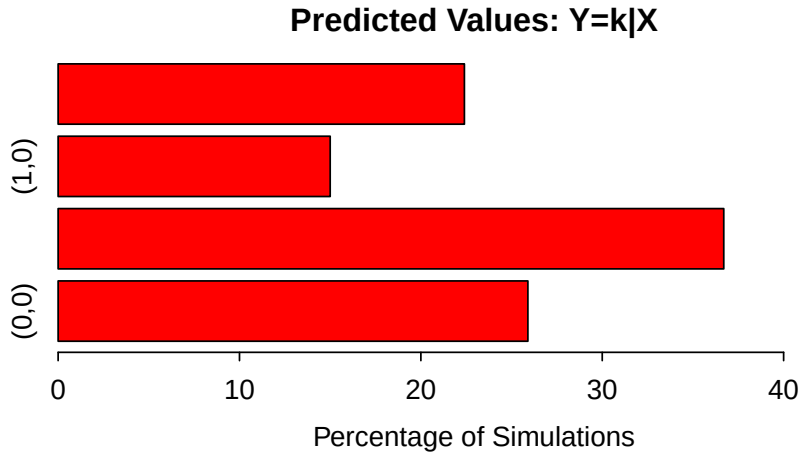
   Set the explanatory variables to their means:

```
> x.out2 <- setx(z.out2)
```

   Simulate draws from the posterior distribution:

```
> s.out2 <- sim(z.out2, x = x.out2)
> summary(s.out2)


> plot(s.out2)
```





Model

For each observation, define two binary dependent variables, Y_1 and Y_2, each of which take the value of either 0 or 1 (in the following, we suppress the observation index i). We model the joint outcome (Y_1, Y_2) using two marginal probabilities for each dependent variable, and the correlation parameter, which describes how the two dependent variables are related.

- The *stochastic component* is described by two latent (unobserved) continuous variables which follow the bivariate Normal distribution:

$$\begin{pmatrix} Y_1^* \\ Y_2^* \end{pmatrix} \sim N_2 \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right\},$$

  where μ_j is a mean for Y_j^* and ρ is a scalar correlation parameter. The following observation mechanism links the observed dependent variables, Y_j, with these latent variables

$$Y_j = \begin{cases} 1 & \text{if } Y_j^* \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

- The *systemic components* for each observation are

$$\begin{aligned} \mu_j &= x_j\beta_j \quad \text{for} \quad j = 1, 2, \\ \rho &= \frac{\exp(x_3\beta_3) - 1}{\exp(x_3\beta_3) + 1}. \end{aligned}$$

Quantities of Interest

For n simulations, expected values form an $n \times 4$ matrix.

- The expected values (`qi$ev`) for the binomial probit model are the predicted joint probabilities. Simulations of β_1, β_2, and β_3 (drawn form their sampling distributions) are substituted into the systematic components, to find simulations of the predicted joint probabilities $\pi_{rs} = \Pr(Y_1 = r, Y_2 = s)$:

$$\begin{aligned} \pi_{11} &= \Pr(Y_1^* \geq 0, Y_2^* \geq 0) &&= \int_0^\infty \int_0^\infty \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\ \pi_{10} &= \Pr(Y_1^* \geq 0, Y_2^* < 0) &&= \int_0^\infty \int_{-\infty}^0 \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\ \pi_{01} &= \Pr(Y_1^* < 0, Y_2^* \geq 0) &&= \int_{-\infty}^0 \int_0^\infty \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\ \pi_{11} &= \Pr(Y_1^* < 0, Y_2^* < 0) &&= \int_{-\infty}^0 \int_{-\infty}^0 \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \end{aligned}$$

  where r and s may take a value of either 0 or 1, ϕ_2 is the bivariate Normal density.

- The predicted values (`qi$pr`) are draws from the multinomial distribution given the expected joint probabilities.

- The first difference (`qi$fd`) in each of the predicted joint probabilities are given by

$$\text{FD}_{rs} = \Pr(Y_1 = r, Y_2 = s \mid x_1) - \Pr(Y_1 = r, Y_2 = s \mid x).$$

- The risk ratio (`qi$rr`) for each of the predicted joint probabilities are given by

$$\text{RR}_{rs} = \frac{\Pr(Y_1 = r, Y_2 = s \mid x_1)}{\Pr(Y_1 = r, Y_2 = s \mid x)}.$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \{Y_{ij}(t_i = 1) - E[Y_{ij}(t_i = 0)]\} \text{ for } j = 1, 2,$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{ij}(t_i = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{Y_{ij}(t_i = 1) - \widehat{Y_{ij}(t_i = 0)}\right\} \text{ for } j = 1, 2,$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_{ij}(t_i = 0)}$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "bprobit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and obtain a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times 4$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times 3$ matrix of the linear predictors $x_j\beta_j$.
  - `residuals`: an $n \times 3$ matrix of the residuals.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times 2$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times 3$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation; otherwise the quantities are matrices). Available quantities are:
  - `qi$ev`: the simulated expected values (joint predicted probabilities) for the specified values of `x`.
  - `qi$pr`: the simulated predicted outcomes drawn from a distribution defined by the joint predicted probabilities.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio in the predicted probabilities for given `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Bivariate Probit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The bivariate probit function is part of the VGAM package by Thomas Yee [49]. In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/~ yee.Sample data are from [36]

Chapter 29

blogit: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

29.1 blogit: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

Use the bivariate logistic regression model if you have two binary dependent variables (Y_1, Y_2), and wish to model them jointly as a function of some explanatory variables. Each pair of dependent variables (Y_{i1}, Y_{i2}) has four potential outcomes, $(Y_{i1} = 1, Y_{i2} = 1)$, $(Y_{i1} = 1, Y_{i2} = 0)$, $(Y_{i1} = 0, Y_{i2} = 1)$, and $(Y_{i1} = 0, Y_{i2} = 0)$. The joint probability for each of these four outcomes is modeled with three systematic components: the marginal $\Pr(Y_{i1} = 1)$ and $\Pr(Y_{i2} = 1)$, and the odds ratio ψ, which describes the dependence of one marginal on the other. Each of these systematic components may be modeled as functions of (possibly different) sets of explanatory variables.

Syntax

```
> z.out <- zelig(list(mu1 = Y1 ~ X1 + X2 ,
                      mu2 = Y2 ~ X1 + X3),
                 model = "blogit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Input Values

In every bivariate logit specification, there are three equations which correspond to each dependent variable (Y_1, Y_2), and ψ, the odds ratio. You should provide a list of formulas for each equation or, you may use `cbind()` if the right hand side is the same for both equations

```
> formulae <- list(cbind(Y1, Y2) ~ X1 + X2)
```

which means that all the explanatory variables in equations 1 and 2 (corresponding to Y_1 and Y_2) are included, but only an intercept is estimated (all explanatory variables are omitted) for equation 3 (ψ).

You may use the function `tag()` to constrain variables across equations:

```
> formulae <- list(mu1 = y1 ~ x1 + tag(x3, "x3"), mu2 = y2 ~ x2 +
+     tag(x3, "x3"))
```

where `tag()` is a special function that constrains variables to have the same effect across equations. Thus, the coefficient for `x3` in equation `mu1` is constrained to be equal to the coefficient for `x3` in equation `mu2`.

Examples

1. Basic Example

   Load the data and estimate the model:

```
> data(sanction)
```

```
> z.out1 <- zelig(cbind(import, export) ~ coop + cost + target,
+     model = "blogit", data = sanction)
```

   By default, `zelig()` estimates two effect parameters for each explanatory variable in addition to the odds ratio parameter; this formulation is parametrically independent (estimating unconstrained effects for each explanatory variable), but stochastically dependent because the models share an odds ratio.

   Generate baseline values for the explanatory variables (with cost set to 1, net gain to sender) and alternative values (with cost set to 4, major loss to sender):

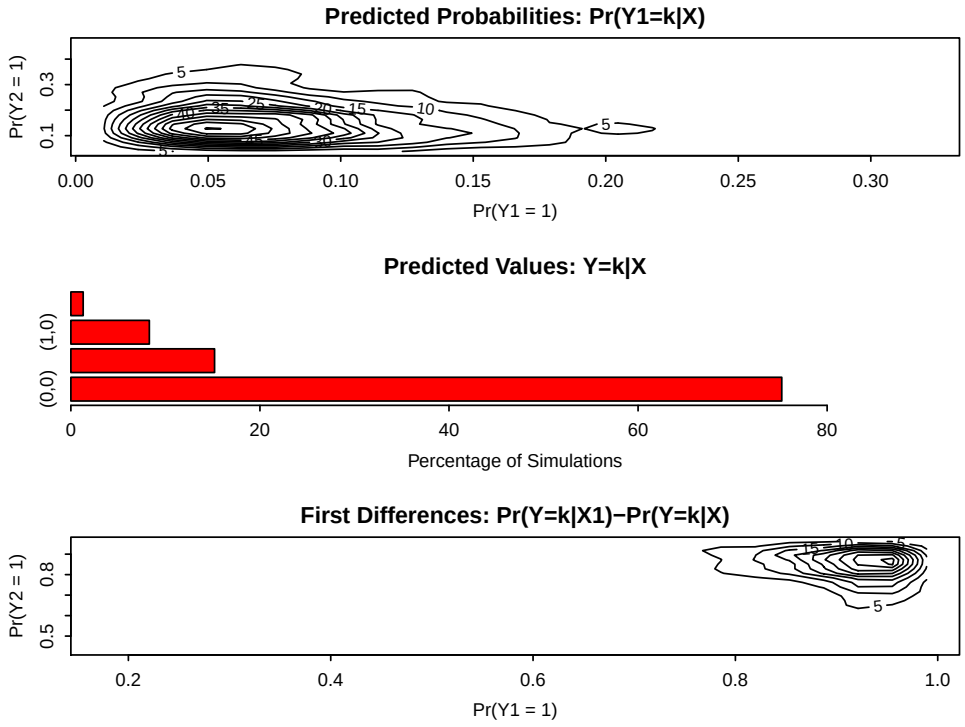
```
> x.low <- setx(z.out1, cost = 1)
```

```
> x.high <- setx(z.out1, cost = 4)
```

   Simulate fitted values and first differences:

```
> s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
> summary(s.out1)
```

```
> plot(s.out1)
```



2. Joint Estimation of a Model with Different Sets of Explanatory Variables

   Using sample data `sanction`, estimate the statistical model, with `import` a function of `coop` in the first equation and `export` a function of `cost` and `target` in the second equation:

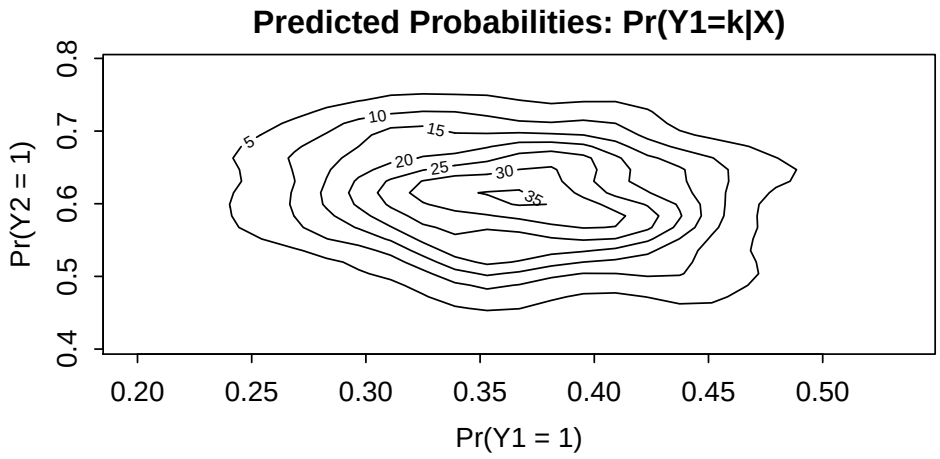
```
> z.out2 <- zelig(list(import ~ coop, export ~ cost + target),
+     model = "blogit", data = sanction)
> summary(z.out2)
```

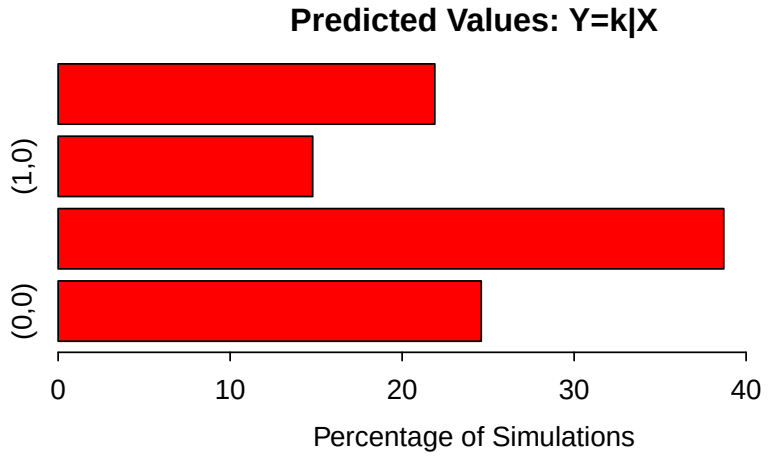
   Set the explanatory variables to their means:

```
> x.out2 <- setx(z.out2)
```

   Simulate draws from the posterior distribution:

```
> s.out2 <- sim(z.out2, x = x.out2)
> summary(s.out2)


> plot(s.out2)
```





Model

For each observation, define two binary dependent variables, Y_1 and Y_2, each of which take the value of either 0 or 1 (in the following, we suppress the observation index). We model the joint outcome (Y_1, Y_2) using a marginal probability for each dependent variable, and the odds ratio, which parameterizes the relationship between the two dependent variables. Define Y_{rs} such that it is equal to 1 when $Y_1 = r$ and $Y_2 = s$ and is 0 otherwise, where r and s take a value of either 0 or 1. Then, the model is defined as follows,

- The *stochastic component* is

$$\begin{aligned} Y_{11} &\sim \text{Bernoulli}(y_{11} \mid \pi_{11}) \\ Y_{10} &\sim \text{Bernoulli}(y_{10} \mid \pi_{10}) \\ Y_{01} &\sim \text{Bernoulli}(y_{01} \mid \pi_{01}) \end{aligned}$$

  where $\pi_{rs} = \Pr(Y_1 = r, Y_2 = s)$ is the joint probability, and $\pi_{00} = 1 - \pi_{11} - \pi_{10} - \pi_{01}$.

- The *systematic components* model the marginal probabilities, $\pi_j = \Pr(Y_j = 1)$, as well as the odds ratio. The odds ratio is defined as $\psi = \pi_{00}\pi_{01}/\pi_{10}\pi_{11}$ and describes the relationship between the two

outcomes. Thus, for each observation we have

$$\begin{aligned} \pi_j &= \frac{1}{1+\exp(-x_j\beta_j)} \quad \text{for} \quad j = 1, 2, \\ \psi &= \exp(x_3\beta_3). \end{aligned}$$

Quantities of Interest

- The expected values (`qi$ev`) for the bivariate logit model are the predicted joint probabilities. Simulations of β_1, β_2, and β_3 (drawn from their sampling distributions) are substituted into the systematic components (π_1, π_2, ψ) to find simulations of the predicted joint probabilities:

$$\begin{aligned} \pi_{11} &= \begin{cases} \frac{1}{2}(\psi-1)^{-1} - a - \sqrt{a^2+b} & \text{for } \psi \neq 1 \\ \pi_1\pi_2 & \text{for } \psi = 1 \end{cases}, \\ \pi_{10} &= \pi_1 - \pi_{11}, \\ \pi_{01} &= \pi_2 - \pi_{11}, \\ \pi_{00} &= 1 - \pi_{10} - \pi_{01} - \pi_{11}, \end{aligned}$$

  where $a = 1 + (\pi_1 + \pi_2)(\psi - 1)$, $b = -4\psi(\psi-1)\pi_1\pi_2$, and the joint probabilities for each observation must sum to one. For n simulations, the expected values form an $n \times 4$ matrix for each observation in `x`.

- The predicted values (`qi$pr`) are draws from the multinomial distribution given the expected joint probabilities.

- The first differences (`qi$fd`) for each of the predicted joint probabilities are given by

$$\text{FD}_{rs} = \Pr(Y_1 = r, Y_2 = s \mid x_1) - \Pr(Y_1 = r, Y_2 = s \mid x).$$

- The risk ratio (`qi$rr`) for each of the predicted joint probabilities are given by

$$\text{RR}_{rs} = \frac{\Pr(Y_1 = r, Y_2 = s \mid x_1)}{\Pr(Y_1 = r, Y_2 = s \mid x)}$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \{Y_{ij}(t_i = 1) - E[Y_{ij}(t_i = 0)]\} \text{ for } j = 1, 2,$$

  where t_i is a binary explanatory variable defining the treatment $(t_i = 1)$ and control $(t_i = 0)$ groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{ij}(t_i = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{Y_{ij}(t_i = 1) - \widehat{Y_{ij}(t_i = 0)}\right\} \text{ for } j = 1, 2,$$

  where t_i is a binary explanatory variable defining the treatment $(t_i = 1)$ and control $(t_i = 0)$ groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_{ij}(t_i = 0)}$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "blogit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and obtain a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times 4$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times 3$ matrix of the linear predictors $x_j\beta_j$.
  - `residuals`: an $n \times 3$ matrix of the residuals.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times 2$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times 3$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation; otherwise the quantities are matrices). Available quantities are:
  - `qi$ev`: the simulated expected joint probabilities (or expected values) for the specified values of `x`.
  - `qi$pr`: the simulated predicted outcomes drawn from a distribution defined by the expected joint probabilities.
  - `qi$fd`: the simulated first difference in the expected joint probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio in the predicted probabilities for given `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Bivariate Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The bivariate logit function is part of the VGAM package by Thomas Yee [49]. In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/~ yee.Sample data are from [36]

Chapter 30

ologit: Ordinal Logistic Regression for Ordered Categorical Dependent Variablest

30.1 `ologit`: Ordinal Logistic Regression for Ordered Categorical Dependent Variables

Use the ordinal logit regression model if your dependent variable is ordered and categorical, either in the form of integer values or character strings.

Syntax

```
> z.out <- zelig(as.factor(Y) ~ X1 + X2, model = "ologit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

If `Y` takes discrete integer values, the `as.factor()` command will order automatically order the values. If `Y` takes on values composed of character strings, such as "strongly agree", "agree", and "disagree", `as.factor()` will order the values in the order in which they appear in `Y`. You will need to replace your dependent variable with a factored variable prior to estimating the model through `zelig()`. See Example 1 for more information on creating ordered factors.

Example

1. Creating An Ordered Dependent Variable

   Load the sample data:

   ```
   > data(sanction)
   ```

   Create an ordered dependent variable:

   ```
   > sanction$ncost <- factor(sanction$ncost, ordered = TRUE,
   +     levels = c("net gain", "little effect", "modest loss",
   +         "major loss"))
   ```

   Estimate the model:

   ```
   > z.out <- zelig(ncost ~ mil + coop, model = "ologit", data = sanction)
   ```

Set the explanatory variables to their observed values:

```
> x.out <- setx(z.out, fn = NULL)
```

Simulate fitted values given `x.out` and view the results:

```
> s.out <- sim(z.out, x = x.out)

> summary(s.out)
```

2. First Differences

   Using the sample data `sanction`, estimate the empirical model and returning the coefficients:

```
> z.out <- zelig(as.factor(cost) ~ mil + coop, model = "ologit",
+     data = sanction)

> summary(z.out)
```

   Set the explanatory variables to their means, with `mil` set to 0 (no military action in addition to sanctions) in the baseline case and set to 1 (military action in addition to sanctions) in the alternative case:

```
> x.low <- setx(z.out, mil = 0)
> x.high <- setx(z.out, mil = 1)
```

   Generate simulated fitted values and first differences, and view the results:

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out)
```

Model

Let Y_i be the ordered categorical dependent variable for observation i that takes one of the integer values from 1 to J where J is the total number of categories.

- The *stochastic component* begins with an unobserved continuous variable, Y_i^*, which follows the standard logistic distribution with a parameter μ_i,

$$Y_i^* \ \sim \ \text{Logit}(y_i^* \mid \mu_i),$$

  to which we add an observation mechanism

$$Y_i \ = \ j \quad \text{if} \quad \tau_{j-1} \leq Y_i^* \leq \tau_j \quad \text{for} \quad j = 1, \ldots, J.$$

  where τ_l (for $l = 0, \ldots, J$) are the threshold parameters with $\tau_l < \tau_m$ for all $l < m$ and $\tau_0 = -\infty$ and $\tau_J = \infty$.

- The *systematic component* has the following form, given the parameters τ_j and β, and the explanatory variables x_i:

$$\Pr(Y \leq j) \ = \ \Pr(Y^* \leq \tau_j) \ = \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)},$$

  which implies:

$$\pi_j \ = \ \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)} - \frac{\exp(\tau_{j-1} - x_i\beta)}{1 + \exp(\tau_{j-1} - x_i\beta)}.$$

Quantities of Interest

- The expected values (`qi$ev`) for the ordinal logit model are simulations of the predicted probabilities for each category:

$$E(Y = j) \;=\; \pi_j \;=\; \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)} - \frac{\exp(\tau_{j-1} - x_i\beta)}{1 + \exp(\tau_{j-1} - x_i\beta)},$$

  given a draw of β from its sampling distribution.

- The predicted value (`qi$pr`) is drawn from the logit distribution described by μ_i, and observed as one of J discrete outcomes.

- The difference in each of the predicted probabilities (`qi$fd`) is given by

$$\Pr(Y = j \mid x_1) \;-\; \Pr(Y = j \mid x) \quad \text{for} \quad j = 1, \ldots, J.$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "ologit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `zeta`: a vector containing the estimated class boundaries τ_j.
  - `deviance`: the residual deviance.
  - `fitted.values`: the $n \times J$ matrix of in-sample fitted values.
  - `df.residual`: the residual degrees of freedom.
  - `edf`: the effective degrees of freedom.
  - `Hessian`: the Hessian matrix.
  - `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, and t-statistics.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`, indexed by simulation × quantity × `x`-observation (for more than one `x`-observation).
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by the expected probabilities, indexed by simulation × `x`-observation.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`, indexed by simulation × quantity × `x`-observation (for more than one `x`-observation).
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Ordinal Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The ordinal logit model is part of the MASS package by William N. Venable and Brian D. Ripley [44]. Advanced users may wish to refer to `help(polr)` as well as [37]. Sample data are from [36].

Chapter 31

oprobit: Ordinal Probit Regression for Ordered Categorical Dependent Variables

31.1 `oprobit`: Ordinal Probit Regression for Ordered Categorical Dependent Variables

Use the ordinal probit regression model if your dependent variables are ordered and categorical. They may take on either integer values or character strings.

Syntax

```
> z.out <- zelig(as.factor(Y) ~ X1 + X2, model = "oprobit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

If `Y` takes discrete integer values, the `as.factor()` command will order it automatically. If `Y` takes on values composed of character strings, such as "strongly agree", "agree", and "disagree", `as.factor()` will order the values in the order in which they appear in `Y`. You will need to replace your dependent variable with a factored variable prior to estimating the model through `zelig()`.

Example

1. Creating An Ordered Dependent Variable

   Load the sample data:

   ```
   > data(sanction)
   ```

   Create an ordered dependent variable:

   ```
   > sanction$ncost <- factor(sanction$ncost, ordered = TRUE,
   +     levels = c("net gain", "little effect", "modest loss",
   +         "major loss"))
   ```

   Estimate the model:

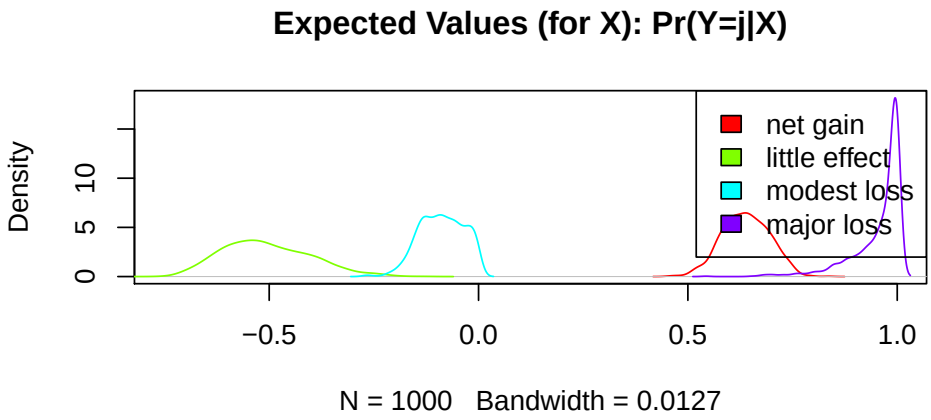
   ```
   > z.out <- zelig(ncost ~ mil + coop, model = "oprobit", data = sanction)
   > summary(z.out)
   ```

Set the explanatory variables to their observed values:

```
> x.out <- setx(z.out, fn = NULL)
```

Simulate fitted values given `x.out` and view the results:

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```



2. First Differences

   Using the sample data `sanction`, let us estimate the empirical model and return the coefficients:

   ```
   > z.out <- zelig(as.factor(cost) ~ mil + coop, model = "oprobit",
   +     data = sanction)

   > summary(z.out)
   ```

   Set the explanatory variables to their means, with `mil` set to 0 (no military action in addition to sanctions) in the baseline case and set to 1 (military action in addition to sanctions) in the alternative case:

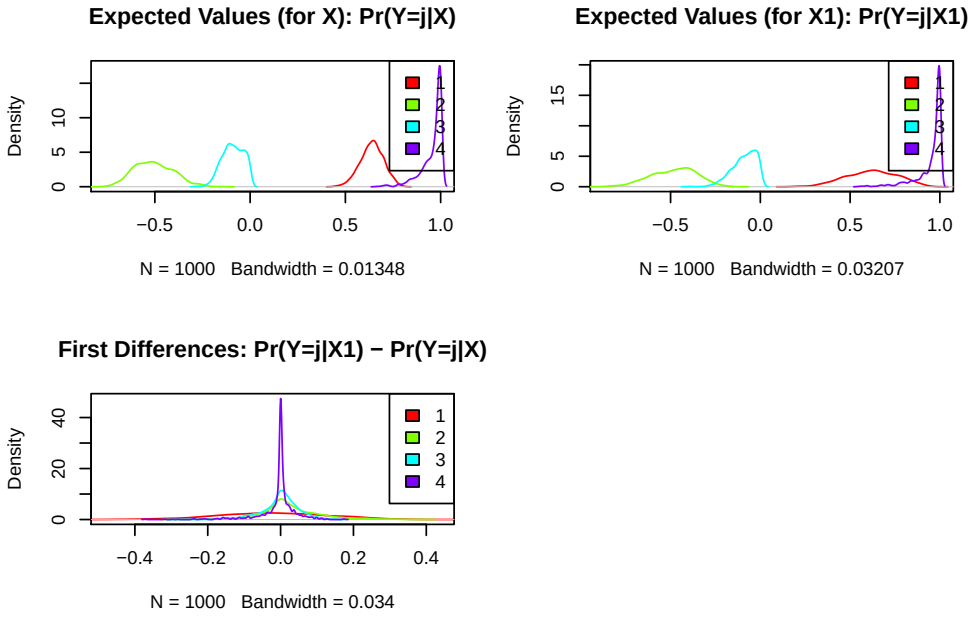
```
> x.low <- setx(z.out, mil = 0)
> x.high <- setx(z.out, mil = 1)
```

Generate simulated fitted values and first differences, and view the results:

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i be the ordered categorical dependent variable for observation i that takes one of the integer values from 1 to J where J is the total number of categories.

- The *stochastic component* is described by an unobserved continuous variable, Y_i^*, which follows the normal distribution with mean μ_i and unit variance

$$Y_i^* \sim N(\mu_i, 1).$$

The observation mechanism is

$$Y_i \;=\; j \quad \text{if} \quad \tau_{j-1} \leq Y_i^* \leq \tau_j \quad \text{for} \quad j = 1, \ldots, J.$$

where τ_k for $k = 0, \ldots, J$ is the threshold parameter with the following constraints; $\tau_l < \tau_m$ for all $l < m$ and $\tau_0 = -\infty$ and $\tau_J = \infty$.

Given this observation mechanism, the probability for each category, is given by

$$\Pr(Y_i = j) \;=\; \Phi(\tau_j \mid \mu_i) - \Phi(\tau_{j-1} \mid \mu_i) \quad \text{for} \quad j = 1, \ldots, J$$

where $\Phi(\mu_i)$ is the cumulative distribution function for the Normal distribution with mean μ_i and unit variance.

- The *systematic component* is given by

$$\mu_i \;=\; x_i\beta$$

where x_i is the vector of explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the ordinal probit model are simulations of the predicted probabilities for each category:

$$E(Y_i = j) \;=\; \Pr(Y_i = j) \;=\; \Phi(\tau_j \mid \mu_i) - \Phi(\tau_{j-1} \mid \mu_i) \quad \text{for} \quad j = 1, \ldots, J,$$

given draws of β from its posterior.

- The predicted value (`qi$pr`) is the observed value of Y_i given the underlying standard normal distribution described by μ_i.

- The difference in each of the predicted probabilities (`qi$fd`) is given by

$$\Pr(Y = j \mid x_1) \;-\; \Pr(Y = j \mid x) \quad \text{for} \quad j = 1, \ldots, J.$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "oprobit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times J$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times (J-1)$ matrix of the linear predictors $x_i\beta_j$.
  - `residuals`: an $n \times (J-1)$ matrix of the residuals.
  - `zeta`: a vector containing the estimated class boundaries.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times J$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times (m-1)$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by the expected probabilities, indexed by simulation $\times$ `x`-observation.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite the Ordinal Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

The ordinal probit function is part of the VGAM package by Thomas Yee [49]. In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at `http://www.stat.auckland.ac.nz/~\protect\kern+.1667em\relaxyee`http://www.stat.auckland.ac.nz/ yee.Sample data are from [36]

Chapter 32

gamma.survey: Survey-Weighted Gamma Regression for Continuous, Positive Dependent Variables

32.1 `gamma.survey`: Survey-Weighted Gamma Regression for Continuous, Positive Dependent Variables

The survey-weighted Gamma regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional Gamma regression models (see Section 12.1), survey-weighted Gamma regression specifies a continuous, positive dependent variable as function of a set of explanatory variables. The survey-weighted Gamma model reports estimates of model parameters identical to conventional Gamma estimates, but uses information from the survey design to correct variance estimates.

The `gamma.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section 6.1), survey-weighted Gamma models accept the following optional inputs:

1. Datasets that include survey weights:.

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
   - `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.

- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a positive and continuous measure of public schools' performance (`api00`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), an indicator for whether each school holds classes year round (`year.rnd`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses school performance on the `meals` and `year.rnd` variables:

   ```
   > z.out1 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
   +     weights = ~pw, data = apistrat)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

   ```
   > x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
   > x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
   ```

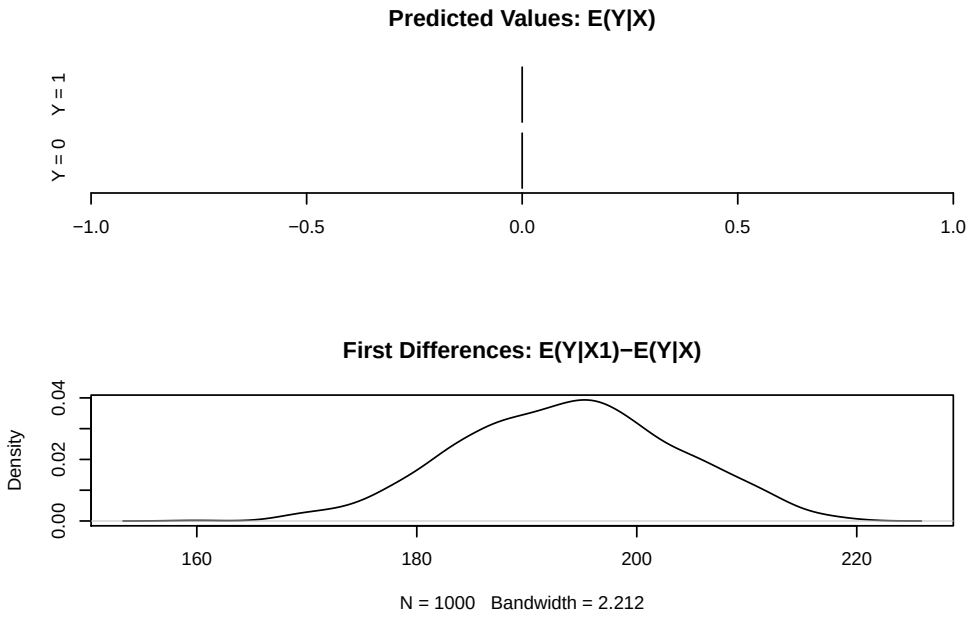
   Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

   ```
   > s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out1)
   ```

   Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

   Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"gamma.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). Suppose that the survey house that published these school data withheld strata/cluster identifiers and instead published replication weights. For the sake of illustrating how replicate weights can be used as inputs in `gamma.survey` models, create a set of jack-knife (JK1) replicate weights:

   ```
   > jk1reps <- jk1weights(psu = apistrat$dnum)
   ```

   Again, estimate a model that regresses school performance on the `meals` and `year.rnd` variables, using the JK1 replicate weights in `jk1reps` to compute standard errors:

   ```
   > z.out3 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
   +     data = apistrat, repweights = jk1reps$weights, type = "JK1")
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set the explanatory variable `meals` at its 20th and 80th quantiles:

   ```
   > x.low <- setx(z.out3, meals = quantile(apistrat$meals, 0.2))
   > x.high <- setx(z.out3, meals = quantile(apistrat$meals, 0.8))
   ```

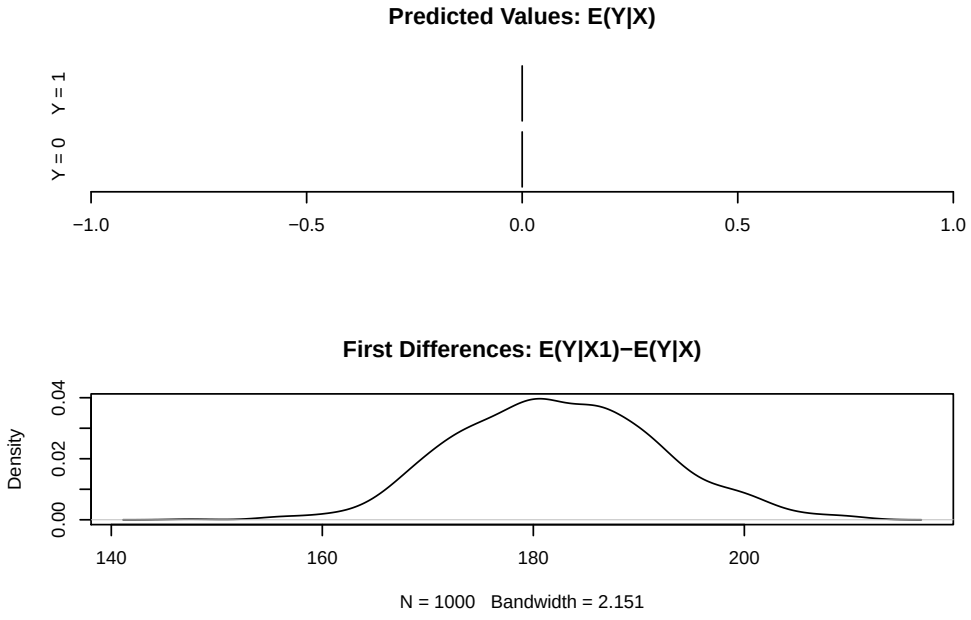
   Generate first differences for the effect of high versus low concentrations of poverty on school performance:

   ```
   > s.out3 <- sim(z.out3, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out3)
   ```

   Generate a visual summary of quantities of interest:

   ```
   > plot(s.out3)
   ```



Model

- The Gamma distribution with scale parameter α has a *stochastic component*:

$$\begin{aligned} Y &\sim \text{Gamma}(y_i \mid \lambda_i, \alpha) \\ f(y) &= \frac{1}{\alpha^{\lambda_i}\,\Gamma\lambda_i}\, y_i^{\lambda_i - 1} \exp - \left\{ \frac{y_i}{\alpha} \right\} \end{aligned}$$

  for $\alpha, \lambda_i, y_i > 0$.

- The *systematic component* is given by

$$\lambda_i = \frac{1}{x_i \beta}$$

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\boldsymbol{\Sigma}} \left[\sum_{i=1}^{n} \frac{(1-\pi_i)}{\pi_i^2} (\mathbf{X}_i (Y_i - \mu_i))' (\mathbf{X}_i (Y_i - \mu_i)) + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\pi_{ij} - \pi_i \pi_j)}{\pi_i \pi_j \pi_{ij}} (\mathbf{X}_i (Y_i - \mu_i))' (\mathbf{X}_j (Y_j - \mu_j)) \right] \hat{\boldsymbol{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in [3] and the Horvitz-Thompson estimator of the variance of a sum described in [6].

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component given draws of α and β from their posteriors:
$$E(Y) = \alpha\lambda_i.$$
- The predicted values (`qi$pr`) are draws from the gamma distribution for each given set of parameters (α, λ_i).
- If `x1` is specified, `sim()` also returns the differences in the expected values (`qi$fd`),
$$E(Y \mid x_1) - E(Y \mid x)$$
.
- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.
- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is
$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.

- `residuals`: the working residuals in the final iteration of the IWLS fit.
- `fitted.values`: the vector of fitted values.
- `linear.predictors`: the vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from a distribution defined by (α, λ_i).
  - `qi$fd`: the simulated first difference in the expected values for the specified values in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `gamma.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `gamma.survey` models in `Zelig`.

How to Cite the Gamma Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37].

Chapter 33

logit.survey: Survey-Weighted Logistic Regression for Dichotomous Dependent Variables

33.1 `logit.survey`: Survey-Weighted Logistic Regression for Dichotomous Dependent Variables

The survey-weighted logistic regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional logistic regression models (see Section 13.1), survey-weighted logistic regression specifies a dichotomous dependent variable as function of a set of explanatory variables. The survey-weighted logit model reports estimates of model parameters identical to conventional logit estimates, but uses information from the survey design to correct variance estimates.

The `logit.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section 6.1), survey-weighted logistic models accept the following optional inputs:

1. Datasets that include survey weights:.

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
   - `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.

- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a dichotomous indicator for whether each public school attends classes year round (`yr.rnd`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), a measure of the percentage of students at each school who are new to to the school (`mobility`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses the year-round schooling indicator on the `meals` and `mobility` variables:

   ```
   > z.out1 <- zelig(yr.rnd ~ meals + mobility, model = "logit.survey",
   +     weights = ~pw, data = apistrat)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

   ```
   > x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
   > x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
   ```

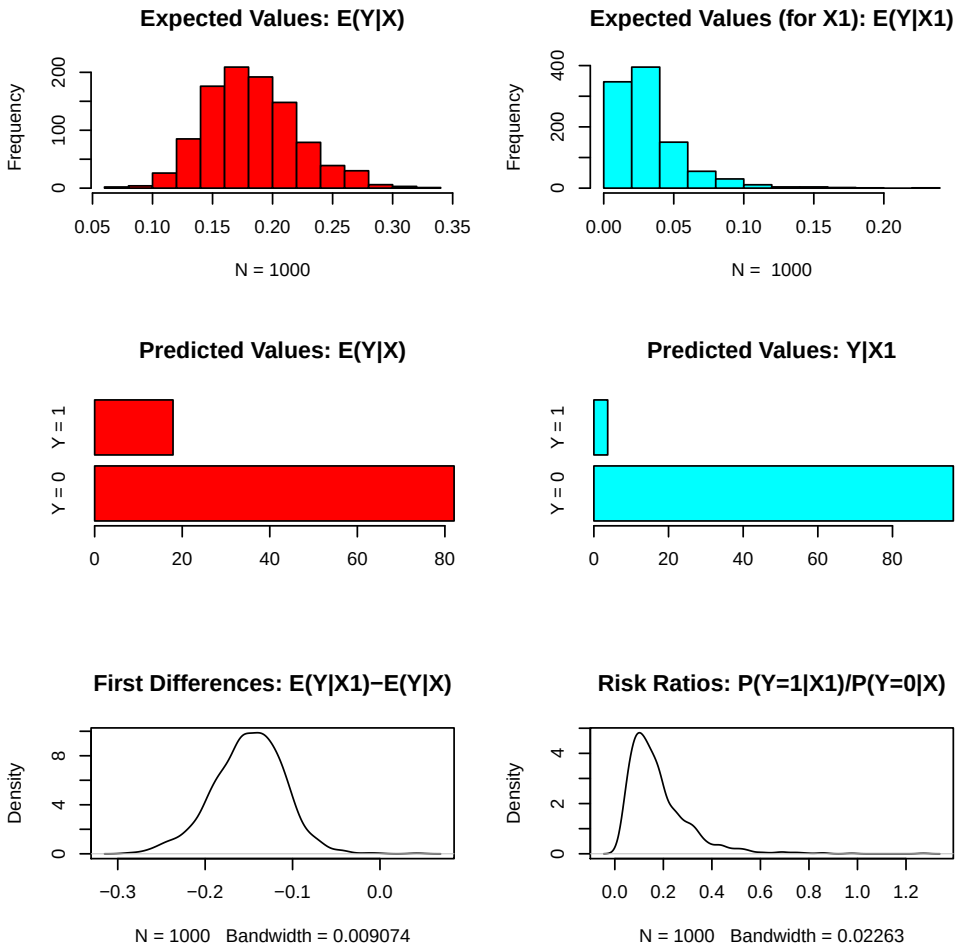
   Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

   ```
   > s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out1)
   ```

   Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(yr.rnd ~ meals + mobility, model = "logit.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

   Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"logit.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in `logit.survey` models, create a set of balanced repeated replicate (BRR) weights and an (artificial) dependent variable to simulate an indicator for whether each hospital was sued:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0,
   +     1), c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   > scd$sued <- as.vector(c(0, 0, 0, 1, 1, 1))
   ```

   Estimate a model that regresses the indicator for hospitals that were sued on the number of patients who arrive alive in each hospital and the number of cardiac arrests that each hospital treats, using the BRR replicate weights in `BRRrep` to compute standard errors.

   ```
   > z.out3 <- zelig(formula = sued ~ arrests + alive, model = "logit.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set `alive` at its mean and set `arrests` at its 20th and 80th quantiles:

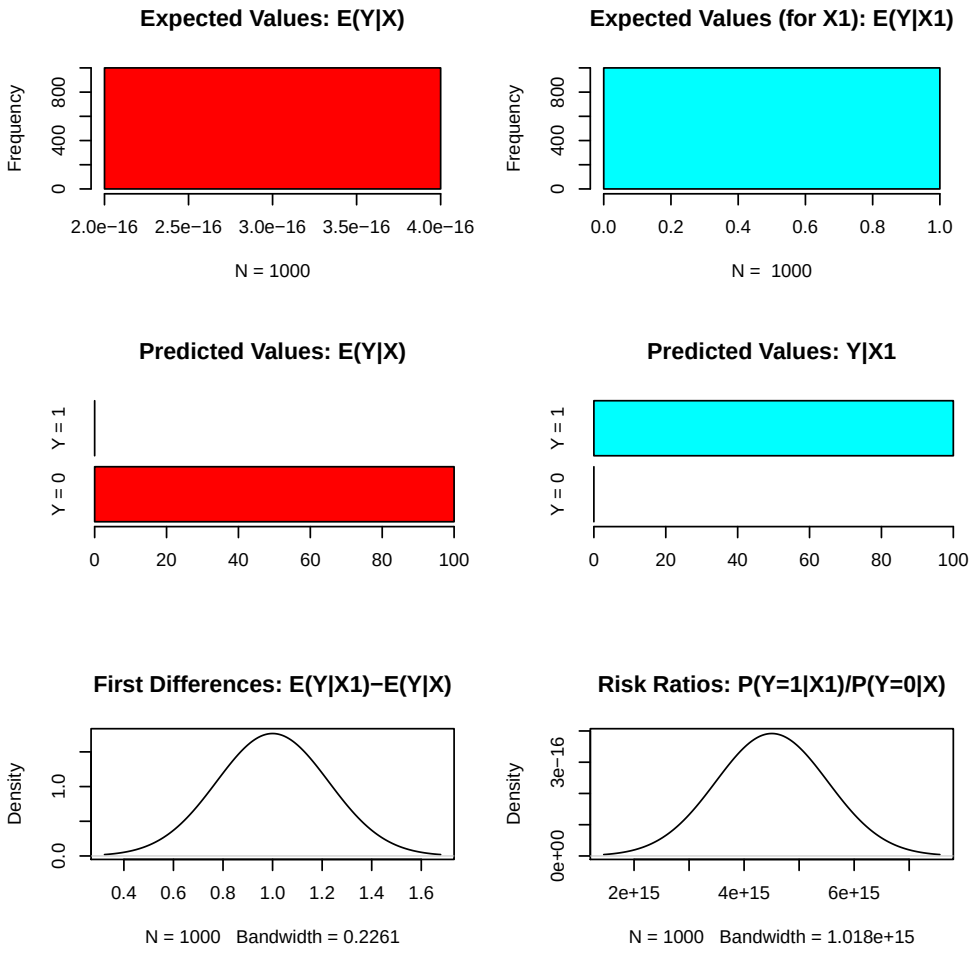
   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

   Generate first differences for the effect of high versus low cardiac arrests on the probability that a hospital will be sued:

   ```
   > s.out3 <- sim(z.out3, x = x.high, x1 = x.low)

   > summary(s.out3)
   ```

   Generate a visual summary of quantities of interest:

   ```
   > plot(s.out3)
   ```



Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i \mid \pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp(-x_i\beta)}.$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i))+2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in [3] and the Horvitz-Thompson estimator of the variance of a sum described in [6].

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected values (`qi$ev`) for the survey-weighted logit model are simulations of the predicted probability of a success:
$$E(Y)=\pi_i=\frac{1}{1+\exp(-x_i\beta)},$$
given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the survey-weighted logit model is defined as
$$\text{FD}=\Pr(Y=1\mid x_1)-\Pr(Y=1\mid x).$$

- The risk ratio (`qi$rr`) is defined as
$$\text{RR}=\Pr(Y=1\mid x_1)\ /\ \Pr(Y=1\mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^{n}t_i}\sum_{i:t_i=1}^{n}\left\{Y_i(t_i=1)-E[Y_i(t_i=0)]\right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i=1$) and control ($t_i=0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i=0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i=0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is
$$\frac{1}{\sum_{i=1}^{n}t_i}\sum_{i:t_i=1}^{n}\left\{Y_i(t_i=1)-\widehat{Y_i(t_i=0)}\right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i=1$) and control ($t_i=0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i=0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i=0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i\beta$
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the name of the input data frame.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `logit.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `logit.survey` models in `Zelig`.

How to Cite the Logit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.
>
> Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37].

Chapter 34

normal.survey: Survey-Weighted Normal Regression for Continuous Dependent Variables

34.1 `normal.survey`: Survey-Weighted Normal Regression for Continuous Dependent Variables

The survey-weighted Normal regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the least squares and Normal regression models (see Section 14.1 and Section 16.1), survey-weighted Normal regression specifies a continuous dependent variable as a linear function of a set of explanatory variables. The survey-weighted normal model reports estimates of model parameters identical to least squares or Normal regression estimates, but uses information from the survey design to correct variance estimates.

The `normal.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section 6.1), survey-weighted Normal models accept the following optional inputs:

1. Datasets that include survey weights:

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
   - `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.

- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a continuous measure of public schools' performance (`api00`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), an indicator for whether each school holds classes year round (`year.rnd`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses school performance on the `meals` and `year.rnd` variables:

   ```
   > z.out1 <- zelig(api00 ~ meals + yr.rnd, model = "normal.survey",
   +     weights = ~pw, data = apistrat)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

   ```
   > x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
   > x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
   ```

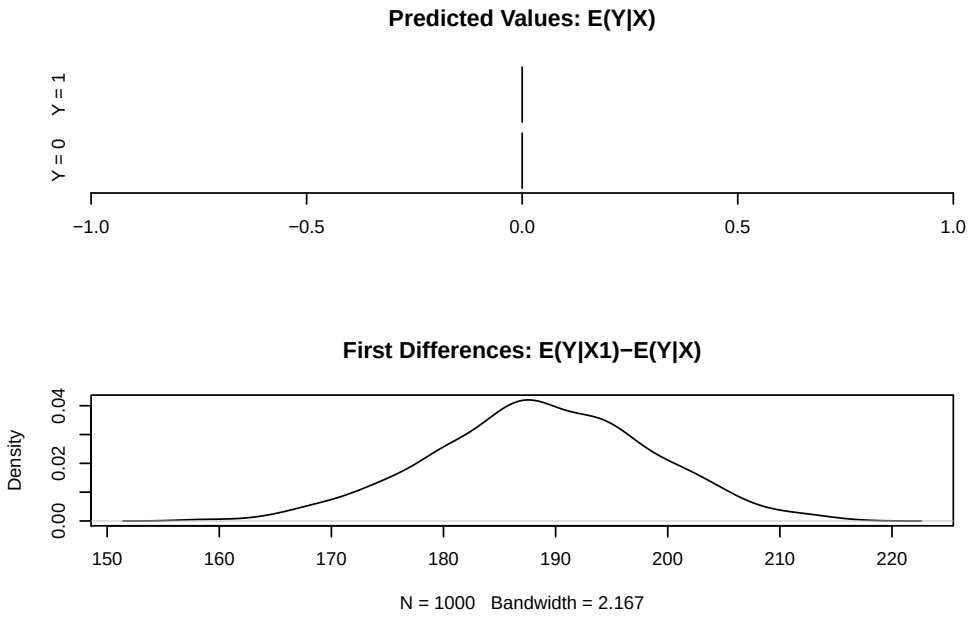
   Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on academic performance:

   ```
   > s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out1)
   ```

   Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(api00 ~ meals + yr.rnd, model = "normal.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

   Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"normal.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals that includes counts of the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in `normal.survey` models, create a set of balanced repeated replicate (BRR) weights:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0,
   +     1), c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   ```

   Estimate a model that regresses counts of patients who arrive alive in each hospital on the number of cardiac arrests that each hospital treats, using the BRR replicate weights in `BRRrep` to compute standard errors.

   ```
   > z.out3 <- zelig(alive ~ arrests, model = "poisson.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set `arrests` at its 20th and 80th quantiles:

   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

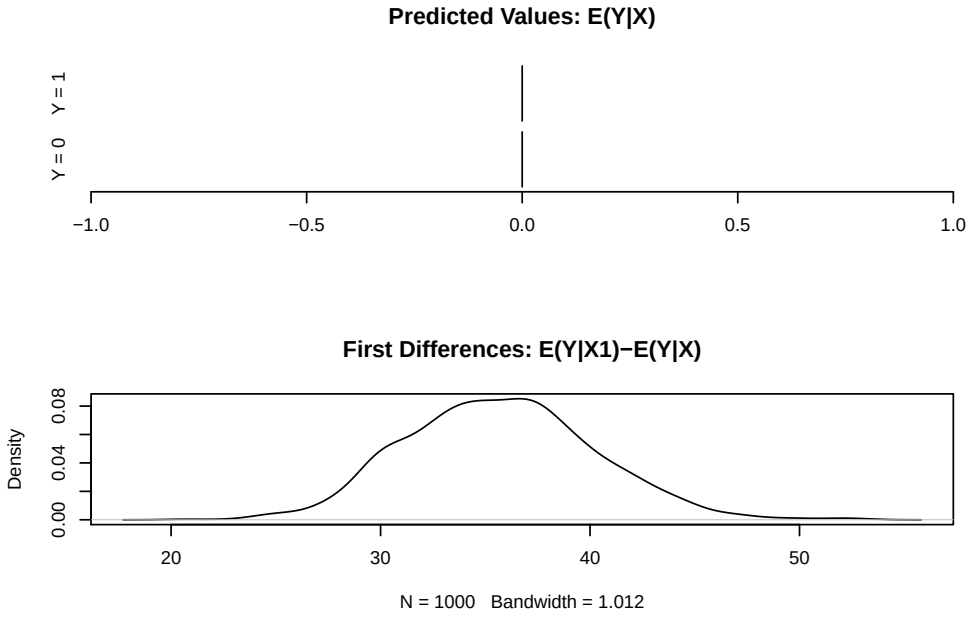
   Generate first differences for the effect of minimal versus maximal cardiac arrests on numbers of patients who arrive alive:

   ```
   > s.out3 <- sim(z.out3, x = x.low, x1 = x.high)
   ```

   ```
   > summary(s.out3)
   ```

   Generate a visual summary of quantities of interest:

   ```
   > plot(s.out3)
   ```



Model

Let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is

$$\mu_i = x_i\beta,$$

where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i))+2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in [3] and the Horvitz-Thompson estimator of the variance of a sum described in [6].

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the the stochastic component,

$$E(Y) = \mu_i = x_i\beta,$$

  given a draw of β from its posterior.

- The predicted value (`qi$pr`) is drawn from the distribution defined by the set of parameters (μ_i, σ).

- The first difference (`qi$fd`) is:

$$\text{FD} \;=\; E(Y \mid x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.

- `fitted.values`: fitted values. For the survey-weighted normal model, these are identical to the `linear predictors`.
- `linear.predictors`: fitted values. For the survey-weighted normal model, these are identical to `fitted.values`.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, σ).
  - `qi$fd`: the simulated first difference in the simulated expected values for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `normal.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `normal.survey` models in `Zelig`.

How to Cite the Normal Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37].

Chapter 35

poisson.survey: Survey-Weighted Poisson Regression for Event Count Dependent Variables

poisson.survey: Survey-Weighted Poisson Regression for Event Count Dependent Variables Kosuke Imai, Olivia Lau, and Gary King

35.1 `poisson.survey`: Survey-Weighted Poisson Regression for Event Count Dependent Variables

The survey-weighted poisson regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional poisson regression model (see Section 17.1), survey-weighted poisson regression specifies a dependent variable representing the number of independent events that occur during a fixed period of time as function of a set of explanatory variables. The survey-weighted poisson model reports estimates of model parameters identical to conventional poisson estimates, but uses information from the survey design to correct variance estimates.

The `poisson.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

(b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section 6.1), survey-weighted poisson models accept the following optional inputs:

1. Datasets that include survey weights:.

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.

- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a variable reporting the number of times a new student enrolled during the previous school year (`enroll`), a measure of each school's academic performance (`api99`), an indicator for whether each school holds classes year round (`year.rnd`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses `enroll` on `api99` and `year.rnd`:

   ```
   > z.out1 <- zelig(enroll ~ api99 + yr.rnd, model = "poisson.survey",
   +     weights = ~pw, data = apistrat)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for the measure of academic performance:

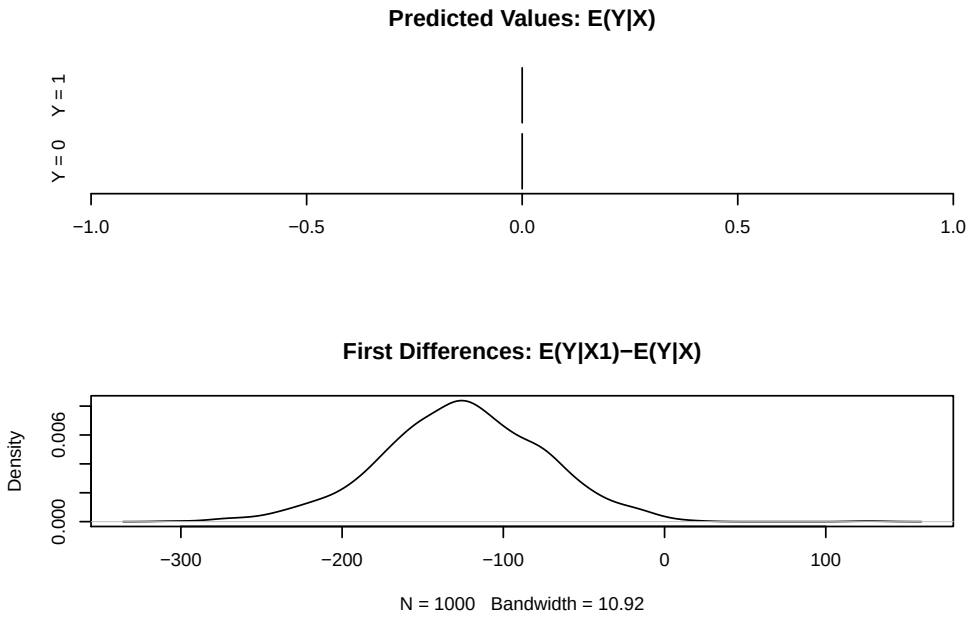
   ```
   > x.low <- setx(z.out1, api99 = quantile(apistrat$api99, 0.2))
   > x.high <- setx(z.out1, api99 = quantile(apistrat$api99, 0.8))
   ```

   Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

```
> s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(enroll ~ api99 + yr.rnd, model = "poisson.survey",
+     data = apistrat, strata = ~stype, fpc = ~fpc)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"poisson.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in `normal.survey` models, create a set of balanced repeated replicate (BRR) weights:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0,
   +     1), c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   ```

   Estimate a model that regresses the count of patients who arrived alive at the hospital last year on the number of patients treated for cardiac arrests, using the BRR replicate weights in `BRRrep` to compute standard errors:

   ```
   > z.out3 <- zelig(alive ~ arrests, model = "poisson.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   > summary(z.out3)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set the explanatory variable `arrests` at its 20th and 80th quantiles:

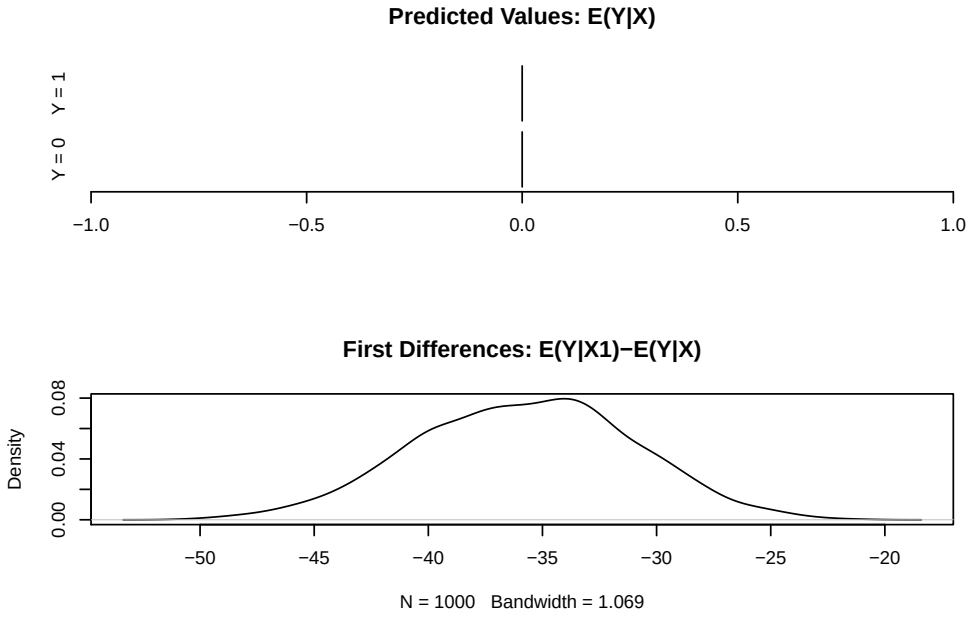
   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

   Generate first differences for the effect of high versus low cardiac arrests on the count of patients who arrive alive:

   ```
   > s.out3 <- sim(z.out3, x = x.high, x1 = x.low)

   > summary(s.out3)
   ```

   Generate a visual summary of quantities of interest:

   ```
   > plot(s.out3)
   ```



Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is

$$\lambda_i = \exp(x_i\beta),$$

  where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i))+2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in [3] and the Horvitz-Thompson estimator of the variance of a sum described in [6].

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp(x_i\beta),$$

  given draws of β from its sampling distribution.

- The predicted value (`qi$pr`) is a random draw from the poisson distribution defined by mean λ_i.

- The first difference in the expected values (`qi$fd`) is given by:

$$\text{FD} \;=\; E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.

- `residuals`: the working residuals in the final iteration of the IWLS fit.
- `fitted.values`: a vector of the fitted values for the systemic component λ.
- `linear.predictors`: a vector of $x_i\beta$.
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by λ_i.
  - `qi$fd`: the simulated first differences in the expected values given the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `poisson.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `poisson.survey` models in `Zelig`.

How to Cite the Gamma Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37].

Chapter 36

probit.survey: Survey-Weighted Probit Regression for Dichotomous Dependent Variables

36.1 `probit.survey`: Survey-Weighted Probit Regression for Dichotomous Dependent Variables

The survey-weighted probit regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional probit regression models (see Section 18.1), survey-weighted probit regression specifies a dichotomous dependent variable as function of a set of explanatory variables. The survey-weighted probit model reports estimates of model parameters identical to conventional probit estimates, but uses information from the survey design to correct variance estimates.

The `probit.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section 6.1), survey-weighted probit models accept the following optional inputs:

1. Datasets that include survey weights:.

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
   - `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.

- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a dichotomous indicator for whether each public school attends classes year round (`yr.rnd`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), a measure of the percentage of students at each school who are new to to the school (`mobility`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses the year-round schooling indicator on the `meals` and `mobility` variables:

   ```
   > z.out1 <- zelig(yr.rnd ~ meals + mobility, model = "probit.survey",
   +     weights = ~pw, data = apistrat)
   ```

   Summarize regression coefficients:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

   ```
   > x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
   > x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
   ```

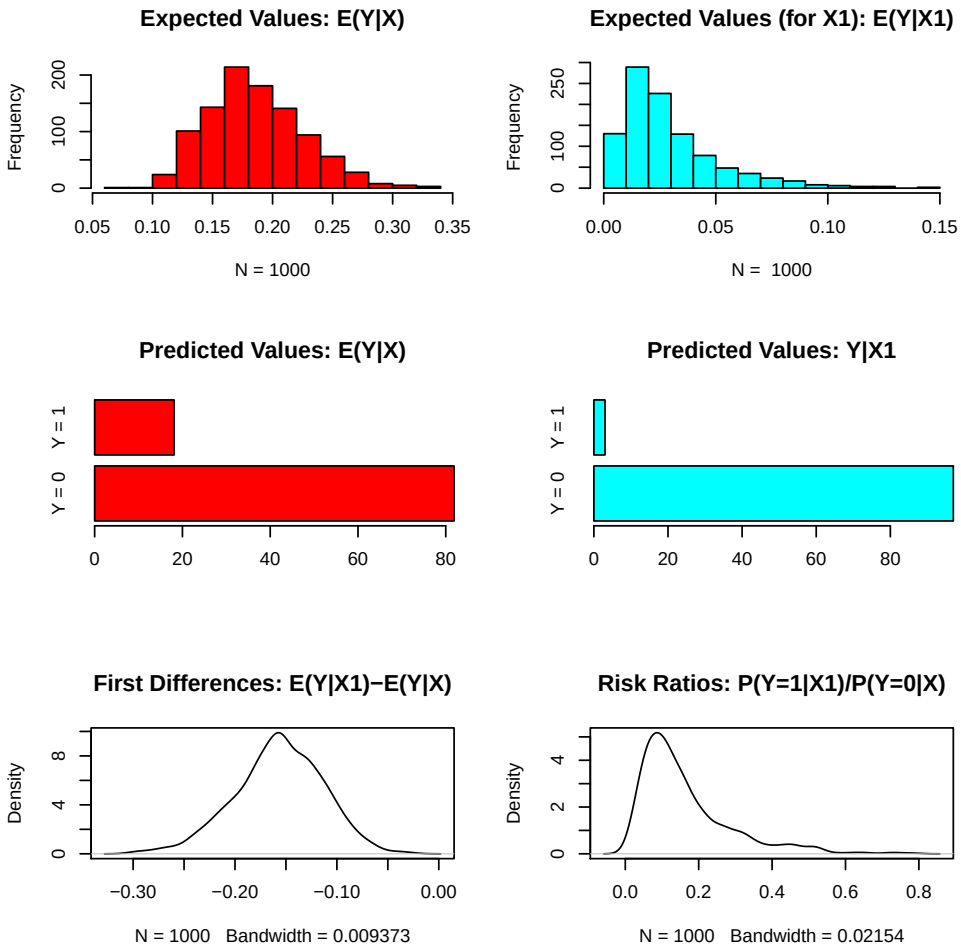
   Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

   ```
   > s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out1)
   ```

   Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(yr.rnd ~ meals + mobility, model = "probit.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

   Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"probit.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in `probit.survey` models, create a set of balanced repeated replicate (BRR) weights and an (artificial) dependent variable to simulate an indicator for whether each hospital was sued:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0,
   +     1), c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   > scd$sued <- as.vector(c(0, 0, 0, 1, 1, 1))
   ```

   Estimate a model that regresses the indicator for hospitals that were sued on the number of patients who arrive alive in each hospital and the number of cardiac arrests that each hospital treats, using the BRR replicate weights in `BRRrep` to compute standard errors.

   ```
   > z.out3 <- zelig(formula = sued ~ arrests + alive, model = "probit.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set `alive` at its mean and set `arrests` at its 20th and 80th quantiles:

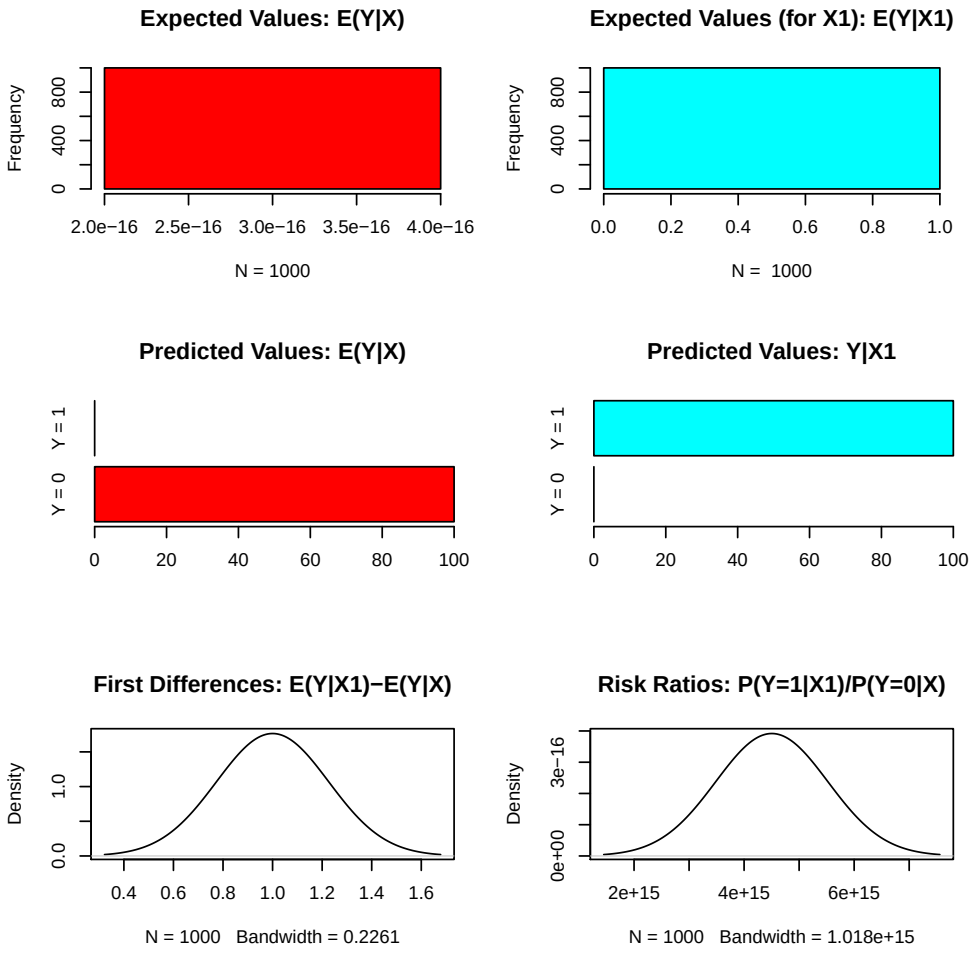
   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

   Generate first differences for the effect of high versus low cardiac arrests on the probability that a hospital will be sued:

   ```
   > s.out3 <- sim(z.out3, x = x.high, x1 = x.low)

   > summary(s.out3)
   ```

   Generate a visual summary of quantities of interest:

   ```
   > plot(s.out3)
   ```



Model

Let Y_i be the observed binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i),$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is

$$\pi_i = \Phi(x_i\beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i)) + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in [3] and the Horvitz-Thompson estimator of the variance of a sum described in [6].

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is a simulation of predicted probability of success

$$E(Y) = \pi_i = \Phi(x_i\beta),$$

  given a draw of β from its sampling distribution.

- The predicted value (`qi$pr`) is a draw from a Bernoulli distribution with mean π_i.

- The first difference (`qi$fd`) in expected values is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) / \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i\beta$
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the name of the input data frame.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `probit.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `probit.survey` models in `Zelig`.

How to Cite the Probit Model

How to Cite the Zelig Software Package

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by **(author?)** [44]. Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as [37].

Part VI

Appendix

Chapter 37

Frequently Asked Questions

37.1 For All Zelig Users

How do I cite Zelig?

We would appreciate if you would cite Zelig as:

> Imai, Kosuke, Gary King and Olivia Lau. 2006. "Zelig: Everyone's Statistical Software," `http://GKing.Harvard.Edu/zelig`.

Please also cite the contributors for the models or methods you are using. These citations can be found in the contributors section of each model or command page.

Why can't I install Zelig?

You must be connected to the internet to install packages from web depositories. In addition, there are a few platform-specific reasons why you may have installation problems:

- **On Windows**: If you are using the very latest version of R, you may not be able to install Zelig until we update Zelig to work on the latest release of R. If you wish to install Zelig in the interim, check the Zelig release notes (Section 38.1) and download the appropriate version of R to work with the last release of Zelig. You may have to manually download and install Zelig.
- **On Mac**: If the latest version of Zelig is not yet available at CRAN but you would like to install it on your Mac, try typing the following at your R prompt:

  ```
  install.packages("Zelig", repos = "http://gking.harvard.edu", type = "source")
  ```

- **On Mac or Linux systems**: If you get the following warning message at the end of your installation:

  ```
  Installation of package VGAM had non-zero exit status in ...
  ```

  this means that you were not able to install VGAM properly. Make sure that you have the g77 Fortran compiler. For PowerPC Macs, download g77 from http://hpc.sourceforge.net). For Intel Macs, download the xcode Apple developer tools. After installation, try to install Zelig again.

Why can't I install R?

If you have problems installing R (rather than Zelig), you should check the R FAQs for your platform. If you still have problems, you can search the archives for the R help mailing list, or email the list directly at r-help@stat.math.ethz.ch.

Why can't I load data?

When you start R, you need to specify your working directory. In linux R, this is done pretty much automatically when you start R, whether within ESS or in a terminal window. In Windows R, you may wish to specify a working directory so that you may load data without typing in long directory paths to your data files, and it is important to remember that *Windows* R uses the *Linux* directory delimiter. That is, if you right click and select the "Properties" link on a Windows file, the slashes are backslashes (\), but Windows R uses forward slashes (/) in directory paths. Thus, the Windows link may be `C:\Program Files\R\R-2.5.1\`, but you would type `C:/Program Files/R/R-2.5.1/` in Windows R.

When you start R in Windows, the working directory is by default the directory in which the R executible is located.

```
# Print your current working directory.
> getwd()

# To read data not located in your working directory.
> data <- read.table("C:/Program Files/R/newwork/mydata.tab")

# To change your working directory.
> setwd("C:/Program Files/R/newwork")

# Reading data in your working directory.
> data <- read.data("mydata.tab")
```

Once you have set the working directory, you no longer need to type the entire directory path.

Where can I find old versions of Zelig?

For some replications, you may require older versions of Zelig.

- **Windows** users may find old binaries at `http://gking.harvard.edu/bin/windows/contrib/` and selecting the appropriate version of R.
- **Linux** and **MacOSX** users may find source files at `http://gking.harvard.edu/src/contrib/`

If you want an older version of Zelig because you are using an older version of R, we strongly suggest that you update R and install the latest version of Zelig.

Some Zelig functions don't work for me!

If this is a new phenomenon, there may be functions in your namespace that are overwriting Zelig functions. In particular, if you have a function called `zelig`, `setx`, or `sim` in your workspace, the corresponding functions in Zelig will not work. Rather than deleting things that you need, R will tell you the following when you load the Zelig library:

```
Attaching package: 'Zelig'
        The following object(s) are masked _by_ .GlobalEnv :
         sim
```

In this case, simply rename your `sim` function to something else and load Zelig again:

```
> mysim <- sim
> detach(package:Zelig)
> library(Zelig)
```

Who can I ask for help? How do I report bugs?

If you find a bug, or cannot figure something out, please follow these steps: (1) Reread the relevant section of the documentation. (2) Update Zelig if you don't have the current version. (3) Rerun the same code and see if the bug has been fixed. (4) Check our list of frequently asked questions. (5) Search or browse messages to find a discussion of your issue on the zelig listserv.

If none of these work, then if you haven't already, please (6) subscribe to the Zelig listserv and (7) send your question to the listserv at `zelig@lists.gking.harvard.edu`. Please explain exactly what you did and include the full error message, including the `traceback()`. You should get an answer from the developers or another user in short order.

How do I increase the memory for R?

Windows users may get the error that R has run out of memory.

If you have R already installed and subsequently install more RAM, you may have to reinstall R in order to take advantage of the additional capacity.

You may also set the amount of available memory manually. Close R, then right-click on your R program icon (the icon on your desktop or in your programs directory). Select "Properties", and then select the "Shortcut" tab. Look for the "Target" field and after the closing quotes around the location of the R executible, add

```
--max-mem-size=500M
```

as shown in the figure below. You may increase this value up to 2GB or the maximum amount of physical RAM you have installed.

If you get the error that R cannot allocate a vector of length x, close out of R and add the following line to the "Target" field:

```
--max-vsize=500M
```

or as appropriate.

You can always check to see how much memory R has available by typing at the R prompt

```
> round(memory.limit()/2^20, 2)
```

which gives you the amount of available memory in MB.

Why doesn't the pdf print properly?

Zelig uses several special LaTeX environments. If the pdf looks right on the screen, there are two possible reasons why it's not printing properly:

- Adobe Acrobat isn't cleaning up the document. Updating to Acrobat Reader 6.0.1 or higher should solve this problem.
- Your printer doesn't support PostScript Type 3 fonts. Updating your print driver should take care of this problem.

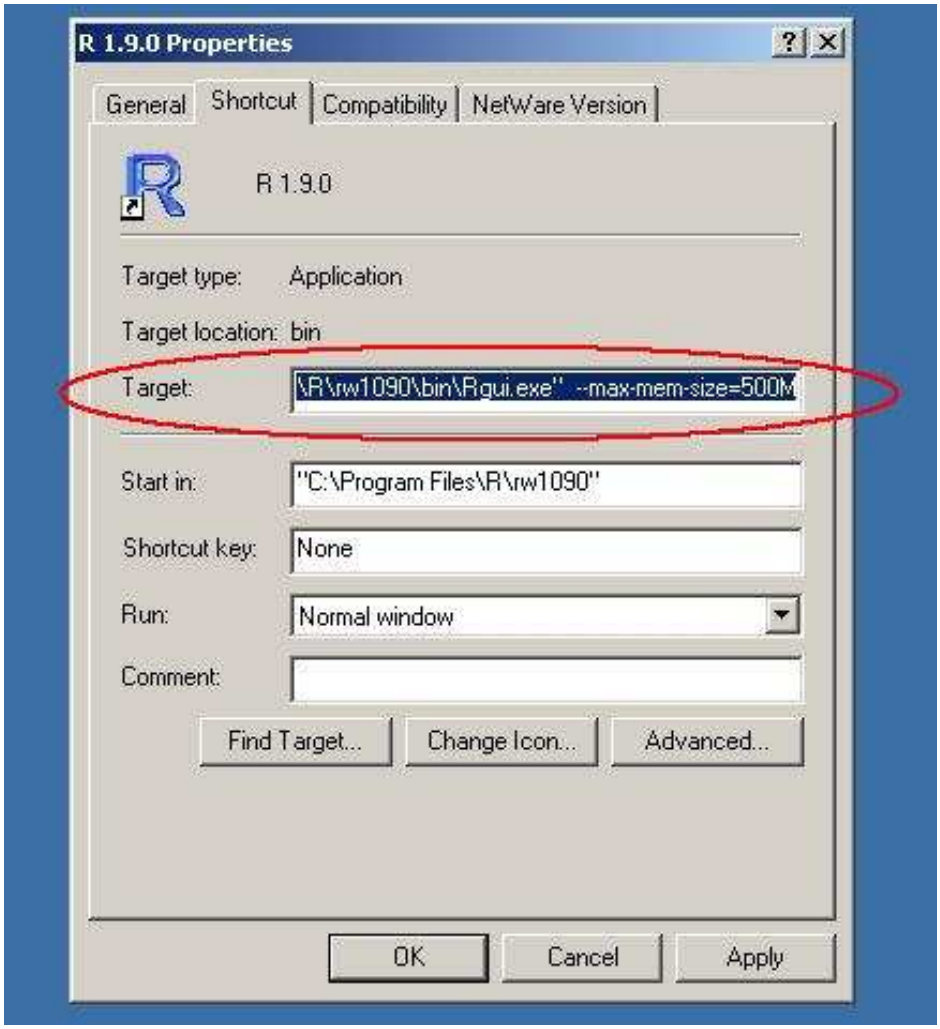
R is neat. How can I find out more?

R is a collective project with contributors from all over the world. Their website (`http://www.r-project.org`) has more information on the R project, R packages, conferences, and other learning material.

In addition, there are several canonical references which you may wish to peruse:

Venables, W.N. and B.D. Ripley. 2002. *Modern Applied Statistics with S.* 4th Ed. Springer-Verlag.

Venables, W.N. and B.D. Ripley. 2000. *S Programming.* Springer-Verlag.



37.2 For Zelig Contributors

Where can I find the source code for Zelig?

Zelig is distributed under the GNU General Public License, Version 2. After installation, the source code is located in your R library directory. For Linux users who have followed our installation example, this is `~/.R/library/Zelig/`. For Windows users under R 2.5.1, this is by default `C:\Program Files\R\R-2.5.1\library\Zelig\`. For Macintosh users, this is `~/Library/R/library/Zelig/`.

In addition, you may download the latest Zelig source code as a tarball'ed directory from `http://gking.harvard.edu/src/` (This makes it easier to distinguish functions which are run together during installation.)

How can I make my R programs run faster?

Unlike most commercial statistics programs which rely on precompiled and pre-packaged routines, R allows users to program functions and run them in the same environment. If you notice a perceptible lag when

running your R code, you may improve the performance of your programs by taking the following steps:

- Reduce the number of loops. If it is absolutely necessary to run loops in loops, the inside loop should have the most number of cycles because it runs faster than the outside loop. Frequently, you can eliminate loops by using vectors rather than scalars. Most R functions deal with vectors in an efficient and mathematically intuitive manner.
- Do away with loops altogether. You can vectorize functions using the `apply`, `mapply()`, `sapply()`, `lapply()`, and `replicate()` functions. If you specify the function passed to the above `*apply()` functions properly, the R consensus is that they should run significantly faster than loops in general.
- You can compile your code using C or Fortran. R is not compiled, but can use bits of precompiled code in C or Fortran, and calls that code seamlessly from within R wrapper functions (which pass input from the R function to the C code and back to R). Thus, almost every regression package includes C or Fortran algorithms, which are locally compiled in the case of Linux systems or precompiled in the case of Windows distributions. The recommended Linux compilers are gcc for C and g77 for Fortran, so you should make sure that your code is compatible with those standards to achieve the widest possible distribution.

Which compilers can I use with R and Zelig?

In general, the C or Fortran algorithms in your package should compile for any platform. While Windows R packages are distributed as compiled binaries, Linux R compiles packages locally during installation. Thus, to ensure the widest possible audience for your package, you should make sure that your code will compile on gcc (for C and C++), or on g77 (for Fortran).

Chapter 38

What's New?

38.1 What's New: Zelig Release Notes

Releases listed as "stable releases" have been tested against prior versions of Zelig for consistency and accuracy. Testing distributions may contain bugs, but are usually replaced by stable releases within a few days.

- 4.0-3 (May 25, 2010): Testing Release for R 2.12

  Extensive changes including:

  - Added Roxygen-compliant in-source documentation
  - `demo(HelloWorld)` now produces an informative demo on creating a simple statistical package
  - Removed the approximately half the models from Zelig
  - Moved 22 models from the "core" package to more focused packages
  - Added 'zelig.skeleton' method for rapidly creating Zelig packages
  - Resolved error messages, warnings, and notes from R CMD check and R CMD build
  - Moved 500+ page Zelig documentation
  - Temporarily disabling bootstrapping from the `sim` method
  - Temporarily removing `Average Treatment Effects` from the generalized linear models

- 3.4-8 (Jan 1, 2010): Stable release for R 2.10

  Fixed problem with survival regressions and robust standard erros (assuming survival >= 2.35-8)
  Fixed vignette documentation

- 3.4-7 (Oct 23, 2009): Stable release for R 2.9.2

  Fixed .net models

- 3.4-6 (May 22, 2009): Stable release for R 2.9

  Added quantile regression model (quantile)
  Changed the digits option (thanks to Matthias Studer)
  Removed the old mloglm model (Multinomial Log-Linear Regression)

- 3.4-5 (Mar 13, 2009): A bug fixed in `plot.ci()` (thanks to Ken Benoit)

- 3.4-4 (Mar 4, 2009): weights are incorporated into `ologit`, `oprobit`, and `negbin` models (thanks to Casey Klofstad)

- 3.4-2 (Feb 10, 2009): Small fixes in the Rd files as required by new check in CRAN

- 3.4-0 (Jan 2, 2009): Bug-fix release for R 2.8.0 Changed the Zelig citation
  Fixed zelig() signature to ensure that the formals() work properly and all arguments remain documented. "save.data" and "cite" were not documented (thanks to Micah Altman)
  Fixed some typos in model family names (thanks to Kevin Condon)
  Fixed the predicted values in gam.* models
  Fixed the plot functions in gam.* models

- 3.4-0 (Oct 27, 2008): Stable release for R 2.8.0. zelig() now takes a "citation" argument. If "citation" is "true" (default) the model citation is printed in each zelig run
  Introduced two new elemetns on the describe.mymodel function: authors and year
  Fixed the problems with lme4 package. Note that there is still a problem with simulation step of "gamma.mixed" model. We are still working on that.
  Fixed the bug with "gam" models (wrong predicted values)
  Fixed the bug with when zelig model name was provided from a variable (reported from Jeroen)

- **3.3-1** (June 12, 2008): Bug-fix release for R.2.7.0. A bug fix for plot.ci() so that it works with mixed effects models (thanks to Keith Schnakenberg).

- **3.3-0** (June 03, 2008): Stable release for R.2.7.0. Updated coxph so that it handles time-varying covariates (contributed by Patrick Lam). A new plot function for survival models (contributed by John Graves). First version dependencies are as follows:

"MASS"	"7.2-41"
"nlme"	"3.1-87"
"survival"	"2.34"
"coda"	"0.13-1"
"sna"	"1.5"
"boot"	"1.2-31"
"nnet"	"7.2-41"
"zoo"	"1.5-0"
"sandwich"	"2.1-0"
"lme4"	"0.99875-9"
"systemfit"	"1.0-2"
"VGAM"	"0.7-5"
"MCMCpack"	"0.8-2"
"mvtnorm"	"0.8-3"
"gee"	"4.13-13"
"mgcv"	"1.3-29"
"anchors"	"1.9-2"
"survey"	"3.6-13"

- **3.2-1** (April 10, 2008): Bug-fix release for R.2.6.0-2.6.2. Fixed the setx() bug for multiply imputed data sets. (Thanks to Steve Shewfelt and Keith Schnakenberg.)

- **3.2** (April 3, 2008): Stable release for R 2.6.0-2.6.2. Adding models for survey data – `normal.survey`, `logit.survey`, `probit.survey`, `poisson.survey`, `gamma.survey`. First version dependencies are as

follows:

survey	3.6-13
MASS	7.2-34
nlme	3.1-86
survival	2.34
boot	1.2-30
nnet	7.2-34
zoo	1.4-0
sandwich	2.0-2
sna	1.4
lme4	0.99875-9
coda	0.12-1
systemfit	0.8-5
VGAM	0.7-4
MCMCpack	0.8-2
mvtnorm	0.8-1
gee	4.13-13
mgcv	1.3-29
anchors	2.0

- **3.1-1** (January 10, 2008): Bug-fix release for R 2.6.0-2.6.1. Fixed bugs, improved the code and the documentation for mixed effects models. Thanks to Gregor Gorjanc. Fixed systemfit models due to some API changes in systemfit package. Added some other models (including *.mixed models) in plot.ci

- **3.1** (November 30, 2007): Stable release for R 2.6.0-2.6.1. Adding many new models such as `aov`, `chopit`, `coxph`, generalized linear mixed-effects models, and gee models. Also, several bugs are fixed. First version dependencies are as follows:

MASS	7.2-34
nlme	3.1-86
survival	2.34
boot	1.2-30
nnet	7.2-34
zoo	1.4-0
sandwich	2.0-2
sna	1.4
lme4	0.99875-9
coda	0.12-1
systemfit	0.8-5
VGAM	0.7-4
MCMCpack	0.8-2
mvtnorm	0.8-1
gee	4.13-13
mgcv	1.3-29
anchors	2.0

- **3.0-1** – **3.0-6**: Minor bug fixes. Stable release for R 2.5.0-2.5.1.

- **3.0** (July 20, 2007): Stable release for R 2.5.0-2.5.1. Introducing vignettes for each model. Improving documentation in the Zelig web site, improving citation style, improving `help.zelig()` function, adding gam models, social network methods, logit gee model, adding support for cross-validation procedures and diagnostics tools, etc.

- **2.8-3** (May 29, 2007): Stable release for R 2.4.0-2.5.0. Fixed bugs in `help.zelig()`, and summary for multinomial logit, bivariate probit, and bivariate logit with multiple imputation. (Thanks to Brant

Inman and Javier Marquez.) First version dependencies are as follows:

MASS	7.2-34
boot	1.2-27
VGAM	0.7-1
MCMCpack	0.8-2
mvtnorm	0.7-5
survival	2.31
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7
nnet	7.2-34
sna	1.4

- **2.8-2** (March 3, 2007): Stable release for R 2.4.0-2.4.1. Fixed bug in ARIMA simulation process.

- **2.8-1** (February 21, 2007): Stable release for R 2.4.0-2.4.1. Made `setx()` compatible with ordred factor variables (thanks to Mike Ward and Kirill Kalinin). First order dependencies as in version 2.8-1.

- **2.8-0** (February 12, 2007): Stable release for R 2.4.0-2.4.1. Released ARIMA models and network analysis models (least squares and logit) for sociomatrices. First level dependencies are as follows:

MASS	7.2-31
boot	1.2-27
VGAM	0.7-1
MCMCpack	0.7-4
mvtnorm	0.7-5
survival	2.31
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7
nnet	7.2-31
sna	1.4

- **2.7-5** (December 25, 2006): Stable release for R 2.4.0-2.4.1. Fixed bug related to `names.default()`, summary for multiple imputation methods, and prediction for ordinal response models (thanks to Brian Ripley, Chris Lawrence, and Ian Yohai).

- **2.7-4** (November 10, 2006): Stable release for R 2.4.0. Fixed bugs related to R check.

- **2.7-3** (November 9, 2006): Stable release for R 2.4.0. Fixed bugs related to R check.

- **2.7-2** (November 5, 2006): Stable release for R 2.4.0. Temporarily removed ARIMA models.

- **2.7-1** (November 3, 2006): Stable release for R 2.4.0. Made changes regarding the S4 classes in VGAM. The ARIMA (`arima`) model for time series data added by Justin Grimmer. First level dependencies are as follows:

MASS	7.2-29
boot	1.2-26
VGAM	0.7-1
MCMCpack	0.7-4
mvtnorm	0.7-5
survival	2.29
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7

- **2.6-5** (September 14, 2006): Stable release for R 2.3.0-2.3.1. Fixed bugs in bivariate logit, bivariate probit, multinomial logit, and model.matrix.multiple (related to changes in version 2.6-4, but not previous versions, thanks to Chris Lawrence). First level dependencies are as follows:

MASS	7.2-27.1
boot	1.2-26
VGAM	0.6-9
MCMCpack	0.7-1
mvtnorm	0.7-2
survival	2.28
sandwich	1.1-1
zoo	1.0-6
coda	0.10-5

- **2.6-4** (September 8, 2006): Stable release for R 2.3.0-2.3.1. Fixed bugs in `setx()`, and bugs related to `multiple` and the multinomial logit model. Added instructions for installing Fortran tools for Intel macs. Added the R×C ecological inference model. (thanks to Kurt Hornik, Luke Keele, Joerg Mueller-Scheessel, and B. Dan Wood)

- **2.6-3** (June 19, 2006): Stable release for R 2.0.0-2.3.1. Fixed bug in VDC interface functions, and `parse.formula()`. (thanks to Micah Altman, Christopher N. Lawrence, and Eric Kostello)

- **2.6-2** (June 7, 2006): Stable release for R 2.0.0-2.3.1. Removed R × C EI. Changed `data = list()` to `data = mi()` for multiply-imputed data frames. First level version compatibilities are as for version 2.6-1.

- **2.6-1** (April 29, 2006): Stable release for R 2.0.0-2.2.1. Fixed major bug in ordinal logit and ordinal probit expected value simulation procedure (does not affect Bayesian ordinal probit). (reported by Ian Yohai) Added the following ecological inference EI models: Bayesian hierarchical EI, Bayesian dynamic EI, and R × C EI. First level version compatibilities (at time of release) are as follows:

MASS	7.2-24
boot	1.2-24
VGAM	0.6-8
MCMCpack	0.7-1
mvtnorm	0.7-2
survival	2.24
sandwich	1.1-1
zoo	1.0-6
coda	0.10-5

- **2.5-4** (March 16, 2006): Stable release for R 2.0.0-2.2.1. Fixed bug related to windows build. First-level dependencies are the same as in version 2.5-1.

- **2.5-3** (March 9, 2006): Stable release for R 2.0.0-2.2.1. Fixed bugs related to VDC GUI. First level dependencies are the same as in version 2.5-1.

- **2.5-2** (February 3, 2006): Stable release for R 2.0.0-2.2.1. Fixed bugs related to VDC GUI. First level dependencies are the same as in version 2.5-1.

- **2.5-1** (January 31, 2006): Stable release for R 2.0.0-2.2.1. Added methods for multiple equation models. Added tobit regression. Fixed bugs related to robust estimation and upgrade of sandwich and zoo packages. Revised `setx()` to use environments. Added `current.packages()` to retrieve version of packages upon which Zelig depends. First level version compatibilities (at time of release) are as

follows:

MASS	7.2-24
boot	1.2-24
VGAM	0.6-7
mvtnorm	0.7-2
survival	2.20
sandwich	1.1-0
zoo	1.0-4
MCMCpack	0.6-6
coda	0.10-3

- **2.4-7** (December 10, 2005): Stable release for R 2.0.0-2.2.2. Fixed the environment of `eval()` called within `setx.default()` (thanks to Micah Altman).
- **2.4-6** (October 27, 2005): Stable release for R 2.0.0-2.2.2. Fixed bug related to simulation for Bayesian Normal regression.
- **2.4-5** (October 18, 2005): Stable release for R 2.0.0-2.2.0. Fixed installation instructions.
- **2.4-4** (September 29, 2005): Stable release for R 2.0.0-2.2.0. Fixed `help.zelig()` links.
- **2.4-3** (September 29, 2005): Stable release for R 2.0.0-2.2.0. Revised `matchit()` documentation.
- **2.4-2** (August 30, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in `setx()` related to `as.factor()` and `I()`. Streamlined `qi.survreg()`.
- **2.4-1** (August 15, 2005): Stable release for R 2.0.0-2.1.1. Added the following Bayesian models: factor analysis, mixed factor analysis, ordinal factor analysis, unidimensional item response theory, k-dimensional item response theory, logit, multinomial logit, normal, ordinal probit, Poisson, and tobit. Also fixed minor bug in formula (long variable names coerced to list).
- **2.3-2** (August 5, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in simulation procedure for lognormal model.
- **2.3-1** (August 4, 2005): Stable release for R 2.0.0-2.1.1. Fixed documentation errors related to model parameterization and code bugs related to first differences and conditional prediction for exponential, lognormal, and Weibull models. (reported by Alison Post)
- **2.2-4** (July 30, 2005): Stable release for R 2.0.0-2.1.1. Revised relogit, adding option for weighting in addition to prior correction. (reported by Martin Plöderl)
- **2.2-3** (July 24, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug associated with robust standard errors for negative binomial.
- **2.2-2** (July 13, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in setx(). (reported by Ying Lu)
- **2.2-1** (July 11, 2005): Stable release for R 2.0.0-2.1.0. Revised ordinal probit to use MASS library. Added robust standard errors for the following regression models: exponential, gamma, logit, lognormal, least squares, negative binomial, normal (Gaussian), poisson, probit, and weibull.
- **2.1-4** (May 22, 2005): Stable release for R 1.9.1-2.1.0. Revised help.zelig() to deal with CRAN build of Windows version. Added recode of slots to lists in NAMESPACE. Revised `install.R` script to deal with changes to `install.packages()`. (reported by Dan Powers and Ying Lu)
- **2.1-3** (May 9, 2005): Stable release for R 1.9.1-2.1.0. Revised param.lm() function to work with bootstrap simulation. (reported by Jens Hainmueller)
- **2.1-2** (April 14, 2005): Stable release for R 1.9.1-2.1.0. Revised summary.zelig().

- **2.1-1** (April 7, 2005): Stable release for R 1.9.1-2.1.0. Fixed bugs in NAMESPACE and summary.vglm().
- **2.0-14** (April 5, 2005): Stable release for R 1.9.1-2.0.1. Added `summary.vglm()` to ensure the compatibility with VGAM 0.6-2.
- **2.0-13** (March 11, 2005): Stable release for R 1.9.1-2.0.1. Fixed bugs in `NAMESPACE` and R-help file for `rocplot()`.
- **2.0-12** (February 20, 2005): Stable release for R 1.9.1-2.0.1. Added `plot = TRUE` option to `rocplot()`.
- **2.0-11** (January 14, 2005): Stable release for R 1.9.1-2.0.1. Changed class name for subsettted models from `"multiple"` to `"strata"`, and modified affected functions.
- **2.0-10** (January 5, 2005): Stable release for R 1.9.1 and R 2.0.0. Fixed bug in ordinal logit simulation procedure. (reported by Ian Yohai)
- **2.0-9** (October 21, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux and Windows). Fixed bug in NAMESPACE file.
- **2.0-8** (October 18, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux only). Revised for submission to CRAN.
- **2.0-7** (October 14, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux only). Fixed bugs in `summary.zelig()`, NAMESPACE, and assorted bugs related to new R release. Revised syntax for multiple equation models.
- **2.0-6** (October 4, 2004): Stable release for R 1.9.1. Fixed problem with NAMESPACE.
- **2.0-5** (September 25, 2004): Stable release for R 1.9.1. Changed installation procedure to source `install.R` from Zelig website.
- **2.0-4** (September 22, 2004): Stable release for R 1.9.1. Fixed typo in installation directions, implemented NAMESPACE, rationalized `summary.zelig()`, and tweaked documentation for least squares.
- **2.0-3** (September 1, 2004): Stable release for R 1.9.1. Fixed bug in conditional prediction for survival models.
- **2.0-2** (August 25, 2004): Stable release for R 1.9.1. Removed predicted values from `ls`.
- **2.0-1b** (July 16, 2004): Stable release for R 1.9.1. MD5 checksum problem fixed. Revised `plot.zelig()` command to be a generic function with methods assigned by the model. Revised entire architecture to accept multiply imputed data sets with strata. Added functions to simplify adding models. Completely restructured reference manual. Fixed bugs related to conditional prediction in setx and summarizing strata in summary.zelig.
- **1.1-2** (June 24, 2004): Stable release for R 1.9.1 (MD5 checksum problem not fixed, but does not seem to cause problems). Fixed bug in `help.zelig()`. (reported by Michael L. Levitan)
- **1.1-1** (June 14, 2004): Stable release for R 1.9.0. Revised `zelig()` procedure to use `zelig2model()` wrappers, revised `help.zelig()` to use a data file with extension `.url.tab`, and revised `setx()` procedure to take a list of `fn` to apply to variables, and such that `fn = NULL` returns the entire `model.matrix()`.
- **1.0-8** (May 27, 2004): Stable release for R 1.9.0. Fixed bug in simulation procedure for survival models. (reported by Elizabeth Stuart)
- **1.0-7** (May 26, 2004): Stable release for R 1.9.0. Fixed bug in relogit simulation procedure. (reported by Tom Vanwellingham)

- **1.0-6** (May 11, 2004): Stable release for R 1.9.0. Fixed bug in setx.default, which had previously failed to ignore extraneous variables in data frame. (reported by Steve Purpura)
- **1.0-5** (May 7, 2004): Replaced relogit procedure with memory-efficient version. (reported by Tom Vanwellingham)
- **1.0-4** (April 19, 2004): Stable release for R 1.9.0. Added vcov.lm method; changed print for summary.relogit.
- **1.0-2** (April 16, 2004): Testing distribution for R 1.9.0.
- **1.0-1** (March, 23, 2004): Stable release for R 1.8.1.

38.2 What's Next?

We have several plans for expanding and improving Zelig. Major changes slated for Version 3.0 (and beyond) include:

- Hierarchical and multi-level models
- Ecological inference models
- GEE models
- Neural network models
- Average treatment effects for everyone (treated and control units)
- Time-series cross-sectional models (via `nlme`)
- Generalized boosted regression model (via `gbm`)
- Saving random seeds to ensure exact replication

If you have suggestions, or packages that you would like to contribute to Zelig, please email our listserv at zelig@lists.gking.harvard.edu.

Bibliography

[1] Christopher Adolph, with Michael C. Herron Gary King, and Kenneth W. Shotts. A consensus on second stage analyses in ecological inference models. *Political Analysis*, 11(1):86–94, Winter 2003. http://gking.harvard.edu/files/abs/akhs-abs.shtml.

[2] Donald W.K. Andrews. Heteroskedasticity and autocorrelation consistent covariance matrix estimation. *Econometrica*, 59(3):817–858, May 1991.

[3] David A. Binder. On the variance of asymptoticaly normal estimators from complex surveys. *International Statistical Review*, 51(3):279–292, 1983.

[4] Andrew Gelman and Gary King. A unified method of evaluating electoral systems and redistricting plans. *American Journal of Political Science*, 38(2):514–554, May 1994. http://gking.harvard.edu/files/abs/writeit-abs.shtml.

[5] Daniel Ho, Kosuke Imai, Gary King, and Elizabeth Stuart. Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference. *Political Analysis*, 15:199–236, 2007. http://gking.harvard.edu/files/abs/matchp-abs.shtml.

[6] D. G. Horvitz and D.J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–685, 1952.

[7] Peter J. Huber. *Robust Statistics*. Wiley, 1981.

[8] Kosuke Imai, Olivia Lau, and Gary King. *logit: Logistic Regression for Dichotomous Dependent*, 2011.

[9] Kosuke Imai, Olivia Lau, and Gary King. *logit.survey: Survey-Weighted Logistic Regression for Dichotomous Dependent Variables*, 2011.

[10] Kosuke Imai, Olivia Lau, and Gary King. *ls: Least Squares Regression for Continuous Dependent Variables*, 2011.

[11] Kosuke Imai, Olivia Lau, and Gary King. *negbinom: Negative Binomial Regression for Event Count Dependent Variables*, 2011.

[12] Kosuke Imai, Olivia Lau, and Gary King. *normal: Normal Regression for Continuous Dependent Variables*, 2011.

[13] Kosuke Imai, Olivia Lau, and Gary King. *normal.survey: Survey-Weighted Normal Regression for Continuous Dependent Variables*, 2011.

[14] Kosuke Imai, Olivia Lau, and Gary King. *ologit: Ordinal Logistic Regression for Ordered Categorical Dependent Variables*, 2011.

[15] Kosuke Imai, Olivia Lau, and Gary King. *poisson: Poisson Regression for Event Count Dependent Variables*, 2011.

[16] Kosuke Imai, Olivia Lau, and Gary King. *probit: Probit Regression for Dichotomous Dependent Variables*, 2011.

[17] Kosuke Imai, Olivia Lau, and Gary King. *probit.survey: Survey-Weighted Probit Regression for Dichotomous Dependent Variables*, 2011.

[18] Kosuke Imai, Olivia Lau, and Gary King. *probit.survey: Survey-Weighted Probit Regression for Dichotomous Dependent Variables*, 2011.

[19] Jonathan Katz and Gary King. A statistical model for multiparty electoral data. *American Political Science Review*, 93(1):15–32, March 1999. http://gking.harvard.edu/files/abs/multiparty-abs.shtml.

[20] Gary King. Replication, replication. *PS: Political Science and Politics*, 28(3):443–499, September 1995. http://gking.harvard.edu/files/abs/replication-abs.shtml.

[21] Gary King. *A Solution to the Ecological Inference Problem: Reconstructing Individual Behavior from Aggregate Data*. Princeton University Press, Princeton, 1997. http://gking.harvard.edu/eicamera/kinroot.html.

[22] Gary King, James Alt, Nancy Burns, and Michael Laver. A unified model of cabinet dissolution in parliamentary democracies. *American Journal of Political Science*, 34(3):846–871, August 1990. http://gking.harvard.edu/files/abs/coal-abs.shtml.

[23] Gary King, James Honaker, Anne Joseph, and Kenneth Scheve. Analyzing incomplete political science data: An alternative algorithm for multiple imputation. *American Political Science Review*, 95(1):49–69, March 2001. http://gking.harvard.edu/files/abs/evil-abs.shtml.

[24] Gary King, Christopher J.L. Murray, Joshua A. Salomon, and Ajay Tandon. Enhancing the validity and cross-cultural comparability of measurement in survey research. *American Political Science Review*, 98(1):191–207, February 2004. http://gking.harvard.edu/files/abs/vign-abs.shtml.

[25] Gary King, Michael Tomz, and Jason Wittenberg. Making the most of statistical analyses: Improving interpretation and presentation. *American Journal of Political Science*, 44(2):341–355, April 2000. http://gking.harvard.edu/files/abs/making-abs.shtml.

[26] Gary King and Langche Zeng. Improving forecasts of state failure. *World Politics*, 53(4):623–658, July 2002. http://gking.harvard.edu/files/abs/civil-abs.shtml.

[27] Gary King and Langche Zeng. The dangers of extreme counterfactuals. *Political Analysis*, 14(2):131–159, 2006. http://gking.harvard.edu/files/abs/counterft-abs.shtml.

[28] Gary King and Langche Zeng. When can history be our guide? the pitfalls of counterfactual inference. *International Studies Quarterly*, pages 183–210, March 2007. http://gking.harvard.edu/files/abs/counterf-abs.shtml.

[29] Patrick Lam. *gamma.gee: General Estimating Equation for Gamma Regression*, 2011.

[30] Patrick Lam. *logit.gee: General Estimating Equation for Logit Regression*, 2011.

[31] Patrick Lam. *normal.gee: General Estimating Equation for Normal Regression*, 2011.

[32] Patrick Lam. *poisson.gee: General Estimating Equation for Poisson Regression*, 2011.

[33] Olivia Lau, Kosuke Imai, and Gary King. *Bivariate Logistic Regression for Two Dichotomous Dependent Variables*, 2011.

[34] Olivia Lau, Kosuke Imai, and Gary King. *Bivariate Probit Regression for Two Dichotomous Dependent Variables*, 2011.

[35] Thomas Lumley and Patrick Heagerty. Weighted empirical adaptive variance estimators for correlated data regression. *jrssb*, 61(2):459–477, 1999.

[36] Lisa Martin. *Coercive Cooperation: Explaining Multilateral Economic Sanctions*. Princeton University Press, 1992. Please inquire with Lisa Martin before publishing results from these data, as this dataset includes errors that have since been corrected.

[37] Peter McCullagh and James A. Nelder. *Generalized Linear Models*. Number 37 in Monograph on Statistics and Applied Probability. Chapman & Hall, 2nd edition, 1989.

[38] Matt Owen and Skyler Cranmer. *Generalized Additive Model for Logistic Regression of Dichotomous Dependent Variables*, 2011.

[39] Matt Owen and Skyler Cranmer. *Generalized Additive Model for Normal Regression of Continuous Dependent Variables*, 2011.

[40] Matt Owen and Skyler Cranmer. *Generalized Additive Model for Poisson Regression of Count Dependent Variables*, 2011.

[41] Matt Owen and Skyler Cranmer. *Generalized Additive Model for Probit Regression of Dichotomous Dependent Variables*, 2011.

[42] Kenneth Scheve and Matthew Slaughter. Labor market competition and individual preferences over immigration policy. *Review of Economics and Statistics*, 83(1):133–145, February 2001. Sample data include only the first five of ten multiply imputed data sets.

[43] Heather Stoll, Gary King, and Langchee Zeng. Whatif: Software for evaluating counterfactuals. *Journal of Statistical Software*, 15(4), 2005. http://www.jstatsoft.org/index.php?vol=15.

[44] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, 4th edition, 2002.

[45] Halbert White. A heteroscedastic-consistent covariance matrix estimator and a direct test for heteroscedasticity. *Econometrica*, 48(4):817–838, 1980.

[46] Simon N. Wood. Modeling and smoothing parameter estimation wiht multiple quadratic penalties. *Journal of the Royal Statistical Society*, 62(2):413–428, 2000.

[47] Simon N. Wood. Stable and efficient multiple smoothing parameter estimation for generalized additive models. *Journal of the American Statistical Association*, 99:673–686, 2004.

[48] Simon N. Wood. *Generalized Additive Models: An Introduction with R*. CRC Press, London, 2006.

[49] T. W. Yee and T. J. Hastie. Reduced-rank vector generalized linear models. *Statistical Modelling*, 3:15–41, 2003.

[50] Achim Zeileis. Econometric computing with hc and hac covariance matrix estimators. *Journal of Statistical Software*, 11(10):1–17, 2004.

[51] Christopher Zorn. Generalized estimating equation models for correlated data: A review with applications. *American Journal of Political Science*, 45:470–490, April 2001.