Introduction to the R Statistical Computing Environment
R Programming

John Fox

McMaster University

ICPSR 2018

Programming Basics
Topics

- Function definition
- Control structures:
  - Conditionals: `if`, `ifelse`, `switch`
  - Iteration: `for`, `while`, `repeat`
  - Recursion
- Avoiding iteration: Vectorization and functions int the `apply()` family
- Large data sets

- There are two latent classes of cases:
  - Those for which the response variable y is *necessarily* zero
  - Those for which the response conditional on the predictors, the xs, is Poisson distributed and thus *may* be zero or a positive integer
- The probability π_i that a particular case i is in the first (necessarily zero) latent class may be dependent upon potentially distinct predictors, zs, according to a binary logistic-regression model:

$$\log_e \frac{\pi_i}{1-\pi_i} = \gamma_0 + \gamma_1 z_{i1} + \cdots + \gamma_p z_{ip}$$

- For an individual i in the second latent class, y follows a Poisson regression model with log link,

$$\log_e \mu_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}$$

where $\mu_i = E(y_i)$, and conditional distribution

$$p(y_i|x_1, \ldots, x_k) = \frac{\mu_i^{y_i} e^{-\mu_i}}{y_i!} \text{ for } y_i = 0, 1, 2, \ldots$$

- The probability of observing a zero count for case i, not knowing to which latent class the case belongs, is therefore

$$p(0) = \Pr(y_i = 0) = \pi_i + (1 - \pi_i)e^{-\mu_i}$$

and the probability of observing a particular nonzero count $y_i > 0$ is

$$p(y_i) = (1 - \pi_i)\frac{\mu_i^{y_i} e^{-\mu_i}}{y_i!}$$

Beyond the Basics: Maximizing a Likelihood
The ZIP (Zero-Inflated Poisson) Regression Model

- The log-likelihood for the ZIP model combines the two components, for $y = 0$ and for $y > 0$:

$$\begin{aligned} \log_e(\boldsymbol{\beta}, \boldsymbol{\gamma}) &= \sum_{y_i=0} \log_e\left[\pi_i + (1 - \pi_i)e^{-\mu_i}\right] \\ &\quad + \sum_{y_1>0} \log_e\left[(1 - \pi_i)\frac{\mu_i^{y_i} e^{-\mu_i}}{y_i!}\right] \end{aligned}$$

where

  - $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_k)'$ is the vector of parameters from the Poisson-regression component of the model (on which the μ_i depend)
  - $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, \ldots, \gamma_p)'$ is the vector of parameters from the logsitic-regression component of the model (on which the π_i depend)

Beyond the Basics: Maximizing a Likelihood

The ZIP (Zero-Inflated Poisson) Regression Model

- In maximizing the likelihood, it helps (but isn't essential) to have the *gradient* (vector of partial derivatives with respect to the parameters) of the log-likelihood.
- For the ZIP model the gradient is complicated:

$$\begin{aligned}
\frac{\partial \log L(\boldsymbol{\beta}, \gamma)}{\partial \boldsymbol{\beta}} &= - \sum_{i:y_i=0} \frac{\exp[-\exp(\mathbf{x}_i'\boldsymbol{\beta})]\exp(\mathbf{x}_i'\boldsymbol{\beta})}{\exp(\mathbf{z}_i'\gamma) + \exp[-\exp(\mathbf{x}_i'\boldsymbol{\beta})]}\mathbf{x}_i \\
&\quad + \sum_{i:y_i>0} [y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta})]\mathbf{x}_i \\
\frac{\partial \log L(\boldsymbol{\beta}, \gamma)}{\partial \gamma} &= \sum_{i:y_i=0} \frac{\exp(\mathbf{z}_i'\gamma)}{\exp(\mathbf{z}_i'\gamma) + \exp[-\exp(\mathbf{x}_i'\boldsymbol{\beta})]}\mathbf{z}_i \\
&\quad - \sum_{i=1}^{n} \frac{\exp(\mathbf{z}_i'\gamma)}{1 + \exp(\mathbf{z}_i'\gamma)}\mathbf{z}_i
\end{aligned}$$

Beyond the Basics: Maximizing a Likelihood

The ZIP (Zero-Inflated Poisson) Regression Model

- And the Hessian (the matrix of second-order partial derivatives, from which the covariance matrix of the coefficients is computed) is even more complicated (thankfully we won't need it):

$$\begin{aligned}
&\frac{\partial \log L(\boldsymbol{\beta}, \gamma)}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} = \\
&\sum_{i:y_i=0} \left\{ \frac{\exp(\mathbf{x}_i'\boldsymbol{\beta})\,[\exp(\mathbf{x}_i'\boldsymbol{\beta}) - 1]}{\exp\,[\exp(\mathbf{x}_i'\boldsymbol{\beta}) + \mathbf{z}_i'\gamma] + 1} - \frac{\exp(2\mathbf{x}_i'\boldsymbol{\beta})}{\{\exp\,[\exp(\mathbf{x}_i'\boldsymbol{\beta}) + \mathbf{z}_i'\gamma] + 1\}^2} \right\} \mathbf{x}_i\mathbf{x}_i' \\
&\quad - \sum_{i:y_i>0} \exp(\mathbf{x}_i'\boldsymbol{\beta})\mathbf{x}_i\mathbf{x}_i'
\end{aligned}$$

- (Hessian continued):

$$
\begin{aligned}
\frac{\partial \log L(\boldsymbol{\beta}, \boldsymbol{\gamma})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}'} &= \sum_{i: y_i = 0} \frac{\exp\left[\exp(\mathbf{x}_i' \boldsymbol{\beta}) + \mathbf{z}_i' \boldsymbol{\gamma}\right]}{\left\{\exp\left[\exp(\mathbf{x}_i' \boldsymbol{\beta}) + \mathbf{z}_i' \boldsymbol{\gamma}\right] + 1\right\}^2} \mathbf{z}_i \mathbf{z}_i' \\
&\quad - \sum_{i=1}^{n} \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{\left[\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + 1\right]^2} \mathbf{z}_i \mathbf{z}_i' \\
\frac{\partial \log L(\boldsymbol{\beta}, \boldsymbol{\gamma})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\gamma}'} &= \sum_{i: y_i = 0} \frac{\exp\left[\mathbf{x}_i' \boldsymbol{\beta} + \exp(\mathbf{x}_i' \boldsymbol{\beta}) + \mathbf{z}_i' \boldsymbol{\gamma}\right]}{\left\{\exp\left[\exp(\mathbf{x}_i' \boldsymbol{\beta}) + \mathbf{z}_i' \boldsymbol{\gamma}\right] + 1\right\}^2} \mathbf{x}_i \mathbf{z}_i'
\end{aligned}
$$

- We can let a general-purpose optimizer do the work of maximizing the log-likelihood
- Optimizers work by evaluating the gradient of the ‘objective function’ (the log-likelihood) at the current estimates of the parameters, either numerically or analytically
- They iteratively improve the parameter estimates using the information in the gradient
- Iteration ceases when the gradient is sufficiently close to zero.
- The covariance matrix of the coefficients is the inverse of the matrix of second derivatives of the log-likelihood, called the *Hessian*, which measures curvature of the log-likelihood at the maximum
- There is generally no advantage in using an analytic Hessian during optimization

Beyond the Basics: Maximizing a Likelihood

The ZIP (Zero-Inflated Poisson) Regression Model

- I'll use the `optim()` function to fit the ZIP model. It takes several arguments, including:
  - `par`, a vector of start values for the parameters
  - `fn`, the objective function to be minimized (in our case the *negative* of the log-likelihood), the first argument of which is the parameter vector; there may be other arguments
  - `gr` (optional), the gradient, also a function of the parameter vector (and possibly of other arguments)
  - `...` (optional), any other arguments to be passed to `fn` and `gr`
  - `method`, I'll use `"BFGS"`
  - `hessian`, set to `TRUE` to return the numerical Hessian at the solution
- See `?optim` for details and other optional arguments

Beyond the Basics: Maximizing a Likelihood

The ZIP (Zero-Inflated Poisson) Regression Model

- `optim()` returns a list with several element, including:
  - `par`, the values of the parameters that minimize the objective function
  - `value`, the value of the objective function at the minimum
  - `convergence`, a code indicating whether the optimization has converged: `0` means that convergence occurred
  - `hessian`, a numerical approximation to the Hessian at the solution
- Again, see `?optim` for details

Beyond the Basics: Object-Oriented Programming
The S3 Object System

- Three standard object-oriented programming systems in R: S3, S4, reference classes
- How the S3 object system works
- Method dispatch, for *object* of class *"class"*: `generic(`*object*`)` ⟹ `generic.`*class*`(`*object*`)` ⟹ `generic.default(`*object*`)`
  - For example, summarizing an object `mod` of class `"lm"`: `summary(mod)` ⟹ `summary.lm(mod)`
- Objects can have more than one class, in which case the first applicable method is used.
  - For example, objects produced by `glm()` are of class `c("glm", "lm")` and therefore can *inherit* methods from class `"lm"`.
- Generic functions: `generic <- function(object,` *other-arguments*`, ...) UseMethod("generic")`
  - For example, `summary <- function(object, ...) UseMethod("summary")`

Beyond the Basics: Debugging and Profiling R Code

- Tools integrated with the RStudio IDE:
  - Locating an error: `traceback()`
  - Setting a breakpoint and examining the local environment of an executing function: `browser()`
  - A simple interactive debugger: `debug()`
  - A post-mortem debugger: `debugger()`
- Measuring time and memory usage with `system.time` (or often better, `microbenchmark()` in the **microbenchmark** package) and `Rprof`