Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation

Scott A. Czepiel*

Abstract

This article presents an overview of the logistic regression model for dependent variables having two or more discrete categorical levels. The maximum likelihood equations are derived from the probability distribution of the dependent variables and solved using the Newton-Raphson method for nonlinear systems of equations. Finally, a generic implementation of the algorithm is discussed.

1 Introduction

Logistic regression is widely used to model the outcomes of a categorical dependent variable. For categorical variables it is inappropriate to use linear regression because the response values are not measured on a ratio scale and the error terms are not normally distributed. In addition, the linear regression model can generate as predicted values any real number ranging from negative to positive infinity, whereas a categorical variable can only take on a limited number of discrete values within a specified range.

The theory of generalized linear models of Nelder and Wedderburn [9] identifies a number of key properties that are shared by a broad class of distributions. This has allowed for the development of modeling techniques that can be used for categorical variables in a way roughly analogous to that in which the linear regression model is used for continuous variables. Logistic regression has proven to be one of the most versatile techniques in the class of generalized linear models.

Whereas linear regression models equate the expected value of the dependent variable to a linear combination of independent variables and their

*Any comments or feedback concerning this article are welcome. Please visit http://czep.net/contact.html

corresponding parameters, generalized linear models equate the linear component to some function of the probability of a given outcome on the dependent variable. In logistic regression, that function is the logit transform: the natural logarithm of the odds that some event will occur. In linear regression, parameters are estimated using the method of least squares by minimizing the sum of squared deviations of predicted values from observed values. This involves solving a system of N linear equations each having N unknown variables, which is usually an algebraically straightforward task. For logistic regression, least squares estimation is not capable of producing minimum variance unbiased estimators for the actual parameters. In its place, maximum likelihood estimation is used to solve for the parameters that best fit the data.

In the next section, we will specify the logistic regression model for a binary dependent variable and show how the model is estimated using maximum likelihood. Following that, the model will be generalized to a dependent variable having two or more categories. In the final section, we outline a generic implementation of the algorithm to estimate logistic regression models.

2 Theory

2.1 Binomial Logistic Regression

2.1.1 The Model

Consider a random variable Z that can take on one of two possible values. Given a dataset with a total sample size of M, where each observation is independent, $\boldsymbol{Z}$ can be considered as a column vector of M binomial random variables Z_i. By convention, a value of 1 is used to indicate "success" and a value of either 0 or 2 (but not both) is used to signify "failure." To simplify computational details of estimation, it is convenient to aggregate the data such that each row represents one distinct combination of values of the independent variables. These rows are often referred to as "populations." Let N represent the total number of populations and let $\boldsymbol{n}$ be a column vector with elements n_i representing the number of observations in population i for $i = 1$ to N where $\sum_{i=1}^{N} n_i = M$, the total sample size.

Now, let $\boldsymbol{Y}$ be a column vector of length N where each element Y_i is a random variable representing the number of successes of Z for population i. Let the column vector $\boldsymbol{y}$ contain elements y_i representing the observed counts of the number of successes for each population. Let $\boldsymbol{\pi}$ be a column

Scott A. Czepiel
http://czep.net/contact.html

vector also of length N with elements $\pi_i = P(Z_i = 1|i)$, i.e., the probability of success for any given observation in the i^{th} population.

The linear component of the model contains the design matrix and the vector of parameters to be estimated. The design matrix of independent variables, $\boldsymbol{X}$, is composed of N rows and $K + 1$ columns, where K is the number of independent variables specified in the model. For each row of the design matrix, the first element $x_{i0} = 1$. This is the intercept or the "alpha." The parameter vector, $\boldsymbol{\beta}$, is a column vector of length $K + 1$. There is one parameter corresponding to each of the K columns of independent variable settings in $\boldsymbol{X}$, plus one, β_0, for the intercept.

The logistic regression model equates the *logit* transform, the log-odds of the probability of a success, to the linear component:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \sum_{k=0}^{K} x_{ik}\beta_k \qquad i = 1, 2, \ldots, N \tag{1}$$

2.1.2 Parameter Estimation

The goal of logistic regression is to estimate the $K+1$ unknown parameters $\boldsymbol{\beta}$ in Eq. 1. This is done with maximum likelihood estimation which entails finding the set of parameters for which the probability of the observed data is greatest. The maximum likelihood equation is derived from the probability distribution of the dependent variable. Since each y_i represents a binomial count in the i^{th} population, the joint probability density function of $\boldsymbol{Y}$ is:

$$f(\boldsymbol{y}|\boldsymbol{\beta}) = \prod_{i=1}^{N} \frac{n_i!}{y_i!(n_i - y_i)!}\pi_i^{y_i}(1-\pi_i)^{n_i - y_i} \tag{2}$$

For each population, there are $\binom{n_i}{y_i}$ different ways to arrange y_i successes from among n_i trials. Since the probability of a success for any one of the n_i trials is π_i, the probability of y_i successes is $\pi_i^{y_i}$. Likewise, the probability of $n_i - y_i$ failures is $(1 - \pi_i)^{n_i - y_i}$.

The joint probability density function in Eq. 2 expresses the values of $\boldsymbol{y}$ as a function of known, fixed values for $\boldsymbol{\beta}$. (Note that $\boldsymbol{\beta}$ is related to $\boldsymbol{\pi}$ by Eq. 1). The *likelihood* function has the same form as the probability density function, except that the parameters of the function are reversed: the likelihood function expresses the values of $\boldsymbol{\beta}$ in terms of known, fixed values for $\boldsymbol{y}$. Thus,

$$L(\boldsymbol{\beta}|\boldsymbol{y}) = \prod_{i=1}^{N} \frac{n_i!}{y_i!(n_i - y_i)!} \pi_i^{y_i} (1 - \pi_i)^{n_i - y_i} \tag{3}$$

The maximum likelihood estimates are the values for $\boldsymbol{\beta}$ that maximize the likelihood function in Eq. 3. The critical points of a function (maxima and minima) occur when the first derivative equals 0. If the second derivative evaluated at that point is less than zero, then the critical point is a maximum (for more on this see a good Calculus text, such as Spivak [14]). Thus, finding the maximum likelihood estimates requires computing the first and second derivatives of the likelihood function. Attempting to take the derivative of Eq. 3 with respect to $\boldsymbol{\beta}$ is a difficult task due to the complexity of multiplicative terms. Fortunately, the likelihood equation can be considerably simplified.

First, note that the factorial terms do not contain any of the π_i. As a result, they are essentially constants that can be ignored: maximizing the equation without the factorial terms will come to the same result as if they were included. Second, note that since $a^{x-y} = a^x/a^y$, and after rearragning terms, the equation to be maximized can be written as:

$$\prod_{i=1}^{N} \left(\frac{\pi_i}{1 - \pi_i} \right)^{y_i} (1 - \pi_i)^{n_i} \tag{4}$$

Note that after taking e to both sides of Eq. 1,

$$\left(\frac{\pi_i}{1 - \pi_i} \right) = e^{\sum_{k=0}^{K} x_{ik} \beta_k} \tag{5}$$

which, after solving for π_i becomes,

$$\pi_i = \left(\frac{e^{\sum_{k=0}^{K} x_{ik} \beta_k}}{1 + e^{\sum_{k=0}^{K} x_{ik} \beta_k}} \right) \tag{6}$$

Substituting Eq. 5 for the first term and Eq. 6 for the second term, Eq. 4 becomes:

$$\prod_{i=1}^{N} (e^{\sum_{k=0}^{K} x_{ik} \beta_k})^{y_i} \left(1 - \frac{e^{\sum_{k=0}^{K} x_{ik} \beta_k}}{1 + e^{\sum_{k=0}^{K} x_{ik} \beta_k}} \right)^{n_i} \tag{7}$$

Use $(a^x)^y = a^{xy}$ to simplify the first product and replace 1 with $\frac{1+e^{\sum \boldsymbol{x}\boldsymbol{\beta}}}{1+e^{\sum \boldsymbol{x}\boldsymbol{\beta}}}$ to simplify the second product. Eq. 7 can now be written as:

$$\prod_{i=1}^{N}(e^{y_i \sum_{k=0}^{K} x_{ik}\beta_k})(1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k})^{-n_i} \tag{8}$$

This is the kernel of the likelihood function to maximize. However, it is still cumbersome to differentiate and can be simplified a great deal further by taking its log. Since the logarithm is a monotonic function, any maximum of the likelihood function will also be a maximum of the log likelihood function and vice versa. Thus, taking the natural log of Eq. 8 yields the log likelihood function:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{N} y_i \left(\sum_{k=0}^{K} x_{ik}\beta_k \right) - n_i \cdot \log(1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k}) \tag{9}$$

To find the critical points of the log likelihood function, set the first derivative with respect to each β equal to zero. In differentiating Eq. 9, note that

$$\frac{\partial}{\partial \beta_k} \sum_{k=0}^{K} x_{ik}\beta_k = x_{ik} \tag{10}$$

since the other terms in the summation do not depend on β_k and can thus be treated as constants. In differentiating the second half of Eq. 9, take note of the general rule that $\frac{\partial}{\partial x} \log y = \frac{1}{y}\frac{\partial y}{\partial x}$. Thus, differentiating Eq. 9 with respect to each β_k,

$$\begin{aligned}
\frac{\partial l(\beta)}{\partial \beta_k} &= \sum_{i=1}^{N} y_i x_{ik} - n_i \cdot \frac{1}{1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k}} \cdot \frac{\partial}{\partial \beta_k} \left(1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k} \right) \\
&= \sum_{i=1}^{N} y_i x_{ik} - n_i \cdot \frac{1}{1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k}} \cdot e^{\sum_{k=0}^{K} x_{ik}\beta_k} \cdot \frac{\partial}{\partial \beta_k} \sum_{k=0}^{K} x_{ik}\beta_k \\
&= \sum_{i=1}^{N} y_i x_{ik} - n_i \cdot \frac{1}{1 + e^{\sum_{k=0}^{K} x_{ik}\beta_k}} \cdot e^{\sum_{k=0}^{K} x_{ik}\beta_k} \cdot x_{ik} \\
&= \sum_{i=1}^{N} y_i x_{ik} - n_i \pi_i x_{ik}
\end{aligned} \tag{11}$$

The maximum likelihood estimates for $\boldsymbol{\beta}$ can be found by setting each of the $K + 1$ equations in Eq. 11 equal to zero and solving for each β_k.

Scott A. Czepiel
http://czep.net/contact.html

Each such solution, if any exists, specifies a critical point–either a maximum or a minimum. The critical point will be a maximum if the matrix of second partial derivatives is negative definite; that is, if every element on the diagonal of the matrix is less than zero (for a more precise definition of matrix *definiteness* see [7]). Another useful property of this matrix is that it forms the variance-covariance matrix of the parameter estimates. It is formed by differentiating each of the $K+1$ equations in Eq. 11 a second time with respect to each element of β, denoted by $\beta_{k'}$. The general form of the matrix of second partial derivatives is

$$\begin{aligned}
\frac{\partial^2 l(\beta)}{\partial\beta_k\partial\beta_{k'}} &= \frac{\partial}{\partial\beta_{k'}}\sum_{i=1}^{N} y_i x_{ik} - n_i x_{ik}\pi_i \\
&= \frac{\partial}{\partial\beta_{k'}}\sum_{i=1}^{N} -n_i x_{ik}\pi_i \\
&= -\sum_{i=1}^{N} n_i x_{ik}\frac{\partial}{\partial\beta_{k'}}\left(\frac{e^{\sum_{k=0}^{K} x_{ik}\beta_k}}{1+e^{\sum_{k=0}^{K} x_{ik}\beta_k}}\right) \qquad (12)
\end{aligned}$$

To solve Eq. 12 we will make use of two general rules for differentiation. First, a rule for differentiating exponential functions:

$$\frac{\mathrm{d}}{\mathrm{d}x}e^{u(x)} = e^{u(x)}\cdot\frac{\mathrm{d}}{\mathrm{d}x}u(x) \qquad (13)$$

In our case, let $u(x) = \sum_{k=0}^{K} x_{ik}\beta_k$. Second, the quotient rule for differentiating the quotient of two functions:

$$\left(\frac{f}{g}\right)'(a) = \frac{g(a)\cdot f'(a) - f(a)\cdot g'(a)}{[g(a)]^2} \qquad (14)$$

Applying these two rules together allows us to solve Eq. 12.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\frac{e^{u(x)}}{1+e^{u(x)}} &= \frac{(1+e^{u(x)})\cdot e^{u(x)}\frac{\mathrm{d}}{\mathrm{d}x}u(x) - e^{u(x)}\cdot e^{u(x)}\frac{\mathrm{d}}{\mathrm{d}x}u(x)}{(1+e^{u(x)})^2} \\
&= \frac{e^{u(x)}\frac{\mathrm{d}}{\mathrm{d}x}u(x)}{(1+e^{u(x)})^2} \\
&= \frac{e^{u(x)}}{1+e^{u(x)}}\cdot\frac{1}{1+e^{u(x)}}\cdot\frac{\mathrm{d}}{\mathrm{d}x}u(x) \qquad (15)
\end{aligned}$$

Thus, Eq. 12 can now be written as:

$$-\sum_{i=1}^{N} n_i x_{ik} \pi_i (1 - \pi_i) x_{ik'} \tag{16}$$

2.1.3 The Newton-Raphson Method

Setting the equations in Eq. 11 equal to zero results in a system of $K + 1$ nonlinear equations each with $K + 1$ unknown variables. The solution to the system is a vector with elements, β_k. After verifying that the matrix of second partial derivatives is negative definite, and that the solution is the global maximum rather than a local maximum, then we can conclude that this vector contains the parameter estimates for which the observed data would have the highest probability of occurrence. However, solving a system of nonlinear equations is not easy—the solution cannot be derived algebraically as it can in the case of linear equations. The solution must be numerically estimated using an iterative process. Perhaps the most popular method for solving systems of nonlinear equations is Newton's method, also called the Newton-Raphson method.

Newton's method begins with an initial guess for the solution then uses the first two terms of the Taylor polynomial evaluated at the initial guess to come up with another estimate that is closer to the solution. This process continues until it converges (hopefully) to the actual solution. Recall that the Taylor polynomial of degree n for f at the point $x = x_0$ is defined as the first n terms of the Taylor series for f:

$$\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i \tag{17}$$

provided that the first n derivatives of f at x_0 all exist. The first degree Taylor polynomial is also the equation for the line tangent to f at the point $(x_0, f(x_0))$. The point at which the tangent line crosses the x-axis, $(x_1, 0)$, is used in the next approximation of the root to be found where $f(x) = 0$. The first step in Newton's method is to take the first degree Taylor polynomial as an approximation for f, which we want to set equal to zero:

$$f(x_0) + f'(x_0) \cdot (x - x_0) = 0 \tag{18}$$

Solving for x, we have:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{19}$$

This new value of x is the next approximation for the root. We let $x_1 = x$ and continue in the same manner to generate $x_2, x_3, \ldots$, until successive approximations converge.

Generalizing Newton's method to a system of equations is not difficult. In our case, the equations whose roots we want to solve are those in Eq. 11, the first derivative of the log-likelihood function. Since Eq. 11 is actually a system of $K+1$ equations whose roots we want to find simultaneously, it is more convenient to use matrix notation to express each step of the Newton-Raphson method. We can write Eq. 11 as $l'(\boldsymbol{\beta})$. Let $\boldsymbol{\beta}^{(0)}$ represent the vector of initial approximations for each β_k, then the first step of Newton-Raphson can be expressed as:

$$\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(0)} + [-l''(\boldsymbol{\beta}^{(0)})]^{-1} \cdot l'(\boldsymbol{\beta}^{(0)}) \tag{20}$$

Let $\boldsymbol{\mu}$ be a column vector of length N with elements $\mu_i = n_i \pi_i$. Note that each element of $\boldsymbol{\mu}$ can also be written as $\mu_i = E(y_i)$, the expected value of y_i. Using matrix multiplication, we can show that:

$$l'(\boldsymbol{\beta}) = \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{\mu}) \tag{21}$$

is a column vector of length $K+1$ whose elements are $\frac{\partial l(\beta)}{\partial \beta_k}$, as derived in Eq. 11. Now, let $\boldsymbol{W}$ be a square matrix of order N, with elements $n_i \pi_i (1 - \pi_i)$ on the diagonal and zeros everywhere else. Again, using matrix multiplication, we can verify that

$$l''(\boldsymbol{\beta}) = -\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X} \tag{22}$$

is a $K+1 \times K+1$ square matrix with elements $\frac{\partial^2 l(\beta)}{\partial \beta_k \partial \beta_{k'}}$. Now, Eq. 20 can be written:

$$\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(0)} + [\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}]^{-1} \cdot \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{\mu}) \tag{23}$$

Continue applying Eq. 23 until there is essentially no change between the elements of $\boldsymbol{\beta}$ from one iteration to the next. At that point, the maximum likelihood estimates are said to have converged, and Eq. 22 will hold the variance-covariance matrix of the estimates.

2.1.4 Caveats

There are two cautionary notes to consider during the iteration procedure. First, it is possible for a parameter estimate to tend to infinity. This is usually a sign that the model is either poorly specified or is behaving badly

due to data sparseness in one or more populations. Obviously, a parameter that tends to infinity will never converge. However, it is sometimes useful to allow a model to converge even in the presence of infinite parameters. To accomplish this, each estimate can be tested against a threshold above which it is considered to be infinite. At that point, the iterations can continue while holding the infinite parameter constant, ignoring its new values in susbsequent iterations, and exempting it from the global test for convergence. The SAS System uses two criteria to test whether a parameter estimate is tending to infinity in PROC CATMOD with the /ML option. If (i) the absolute value of the estimate exceeds five divided by the range of the corresponding independent variable, and (ii) the standard error of the estimate is at least three times greater than the estimate itself.

A second cautionary note deals with a limitation of the Newton-Raphson method. Given certain conditions, it is possible for a given estimate to overshoot the true root in such a way that subsequent iterations enter into a repeating cycle that will never converge. To counter this possibility at each iteration, verify that the value for the likelihood function evaluated at that point is in fact higher than it was during the previous iteration. If at any point the likelihood decreases, this is a sign that the iterations have lost track of the true root and are in danger of converging to a local maximum or not converging at all. One strategy for dealing with this is to apply a "step-halving" function wherein half the distance between the current and prior estimates is tested. If the likelihood at that point is still lower than for the last iteration then half the distance again is tested. This continues for a reasonable number of "sub-iterations" until reaching a point where the likelihood does increase. This is again the method used by the SAS System in PROC CATMOD with the /ML option. We will look at these two caveats in more detail in the section on implementation.

2.2 Multinomial Logistic Regression

This section will generalize the results of the previous section for categorical dependent variables having two or more levels.

2.2.1 The Model

Generalizing to a multinomial dependent variable requires us to make some notational adaptations. Let J represent the number of discrete categories of the dependent variable, where $J \geq 2$. Now, consider random variable Z that can take on one of J possible values. If each observation is independent,

Scott A. Czepiel
http://czep.net/contact.html

then each Z_i is a multinomial random variable. Once again, we aggregate the data into populations each of which represents one unique combination of independent variable settings. As with the binomial logistic regression model, the column vector $\boldsymbol{n}$ contains elements n_i which represent the number of observations in population i, and such that $\sum_{i=1}^{N} n_i = M$, the total sample size.

Since each observation records one of J possible values for the dependent variable, Z, let $\boldsymbol{y}$ be a matrix with N rows (one for each population) and $J-1$ columns. Note that if $J = 2$ this reduces to the column vector used in the binomial logistic regression model. For each population, y_{ij} represents the observed counts of the j^{th} value of Z_i. Similarly, $\boldsymbol{\pi}$ is a matrix of the same dimensions as $\boldsymbol{y}$ where each element π_{ij} is the probability of observing the j^{th} value of the dependent variable for any given observation in the i^{th} population.

The design matrix of independent variables, $\boldsymbol{X}$, remains the same—it contains N rows and $K+1$ columns where K is the number of independent variables and the first element of each row, $x_{i0} = 1$, the intercept. Let $\boldsymbol{\beta}$ be a matrix with $K+1$ rows and $J-1$ columns, such that each element β_{kj} contains the parameter estimate for the k^{th} covariate and the j^{th} value of the dependent variable.

For the multinomial logistic regression model, we equate the linear component to the log of the odds of a j^{th} observation compared to the J^{th} observation. That is, we will consider the J^{th} category to be the omitted or baseline category, where logits of the first $J-1$ categories are constructed with the baseline category in the denominator.

$$\log\left(\frac{\pi_{ij}}{\pi_{iJ}}\right) = \log\left(\frac{\pi_{ij}}{1-\sum_{j=1}^{J-1}\pi_{ij}}\right) = \sum_{k=0}^{K} x_{ik}\beta_{kj} \qquad \begin{array}{c} i = 1, 2, \ldots, N \\ j = 1, 2, \ldots, J-1 \end{array} \tag{24}$$

Solving for π_{ij}, we have:

$$\begin{aligned} \pi_{ij} &= \frac{e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}}{1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}} \qquad j < J \\ \pi_{iJ} &= \frac{1}{1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}} \end{aligned} \tag{25}$$

Scott A. Czepiel
http://czep.net/contact.html

2.2.2 Parameter Estimation

For each population, the dependent variable follows a multinomial distribution with J levels. Thus, the joint probability density function is:

$$f(\boldsymbol{y}|\boldsymbol{\beta}) = \prod_{i=1}^{N}\left[\frac{n_i!}{\prod_{j=1}^{J} y_{ij}!} \cdot \prod_{j=1}^{J} {\pi_{ij}}^{y_{ij}}\right] \tag{26}$$

When $J = 2$, this reduces to Eq. 2. The likelihood function is algebraically equivalent to Eq. 26, the only difference being that the likelihood function expresses the unknown values of $\boldsymbol{\beta}$ in terms of known fixed constant values for $\boldsymbol{y}$. Since we want to maximize Eq. 26 with respect to $\boldsymbol{\beta}$, the factorial terms that do not contain any of the π_{ij} terms can be treated as constants. Thus, the kernel of the log likelihood function for multinomial logistic regression models is:

$$L(\boldsymbol{\beta}|\boldsymbol{y}) \simeq \prod_{i=1}^{N}\prod_{j=1}^{J} {\pi_{ij}}^{y_{ij}} \tag{27}$$

Replacing the J^{th} terms, Eq. 27 becomes:

$$\begin{aligned} & \prod_{i=1}^{N}\prod_{j=1}^{J-1} {\pi_{ij}}^{y_{ij}} \cdot {\pi_{iJ}}^{n_i - \sum_{j=1}^{J-1} y_{ij}} \\ = & \prod_{i=1}^{N}\prod_{j=1}^{J-1} {\pi_{ij}}^{y_{ij}} \cdot \frac{{\pi_{iJ}}^{n_i}}{{\pi_{iJ}}^{\sum_{j=1}^{J-1} y_{ij}}} \\ = & \prod_{i=1}^{N}\prod_{j=1}^{J-1} {\pi_{ij}}^{y_{ij}} \cdot \frac{{\pi_{iJ}}^{n_i}}{\prod_{j=1}^{J-1} {\pi_{iJ}}^{y_{ij}}} \end{aligned} \tag{28}$$

Since $a^{x+y} = a^x a^y$, the sum in the exponent in the denominator of the last term becomes a product over the first $J-1$ terms of j. Continue by grouping together the terms that are raised to the y_{ij} power for each j up to $J-1$:

$$\prod_{i=1}^{N}\prod_{j=1}^{J-1}\left(\frac{\pi_{ij}}{\pi_{iJ}}\right)^{y_{ij}} \cdot {\pi_{iJ}}^{n_i} \tag{29}$$

Now, substitute for π_{ij} and π_{iJ} using Eq. 24 and Eq. 25:

$$\begin{aligned} & \prod_{i=1}^{N}\prod_{j=1}^{J-1}(e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}})^{y_{ij}} \cdot \left(\frac{1}{1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}}\right)^{n_i} \\ = & \prod_{i=1}^{N}\prod_{j=1}^{J-1}e^{y_{ij}\sum_{k=0}^{K}x_{ik}\beta_{kj}} \cdot \left(1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}\right)^{-n_i} \end{aligned} \tag{30}$$

Taking the natural log of Eq. 30 gives us the log likelihood function for the multinomial logistic regression model:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{N}\sum_{j=1}^{J-1}\left(y_{ij}\sum_{k=0}^{K}x_{ik}\beta_{kj}\right) - n_i\log\left(1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}\right) \tag{31}$$

As with the binomial model, we want to find the values for $\boldsymbol{\beta}$ which maximize Eq. 31. We will do this using the Newton-Raphson method, which involves calculating the first and second derivatives of the log likelihood function. We can take the first derivatives using the steps similar to those in Eq. 11:

$$\begin{aligned} \frac{\partial l(\beta)}{\partial\beta_{kj}} &= \sum_{i=1}^{N}y_{ij}x_{ik} - n_i\cdot\frac{1}{1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}}\cdot\frac{\partial}{\partial\beta_{kj}}\left(1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}\right) \\ &= \sum_{i=1}^{N}y_{ij}x_{ik} - n_i\cdot\frac{1}{1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}}\cdot e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}\cdot\frac{\partial}{\partial\beta_{kj}}\sum_{k=0}^{K}x_{ik}\beta_{kj} \\ &= \sum_{i=1}^{N}y_{ij}x_{ik} - n_i\cdot\frac{1}{1+\sum_{j=1}^{J-1}e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}}\cdot e^{\sum_{k=0}^{K}x_{ik}\beta_{kj}}\cdot x_{ik} \\ &= \sum_{i=1}^{N}y_{ij}x_{ik} - n_i\pi_{ij}x_{ik} \end{aligned} \tag{32}$$

Note that there are $(J-1)\cdot(K+1)$ equations in Eq. 32 which we want to set equal to zero and solve for each β_{kj}. Although technically a matrix, we may consider $\boldsymbol{\beta}$ to be a column vector, by appending each of the additional columns below the first. In this way, we can form the matrix of second partial derivatives as a square matrix of order $(J-1)\cdot(K+1)$. For each β_{kj}, we need to differentiate Eq. 32 with respect to every other β_{kj}. We can express the general form of this matrix as:

$$\begin{aligned}\frac{\partial^2 l(\beta)}{\partial\beta_{kj}\partial\beta_{k'j'}} &= \frac{\partial}{\partial\beta_{k'j'}}\sum_{i=1}^{N} y_{ij}x_{ik} - n_i\pi_{ij}x_{ik} \\ &= \frac{\partial}{\partial\beta_{k'j'}}\sum_{i=1}^{N} -n_i x_{ik}\pi_{ij} \\ &= -\sum_{i=1}^{N} n_i x_{ik}\frac{\partial}{\partial\beta_{k'j'}}\left(\frac{e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}}{1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}}\right) \qquad (33)\end{aligned}$$

Applying the quotient rule of Eq. 14, note that the derivatives of the numerator and denominator differ depending on whether or not $j' = j$:

$$\begin{aligned} f'(a) = g'(a) = e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}} \cdot x_{ik'} \qquad & j' = j \\ f'(a) = 0 \qquad g'(a) = e^{\sum_{k=0}^{K} x_{ik}\beta_{kj'}} \cdot x_{ik'} \qquad & j' \neq j \qquad (34)\end{aligned}$$

Thus, when $j' = j$, the partial derivative in Eq. 33 becomes:

$$\begin{aligned} & \frac{\left(1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\right)\cdot e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\cdot x_{ik'} - e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\cdot e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\cdot x_{ik'}}{\left(1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\right)^2} \\ = & \frac{e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\cdot x_{ik'}\left(1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}} - e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\right)}{\left(1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\right)^2} \\ = & \pi_{ij}x_{ik'}(1-\pi_{ij}) \qquad (35)\end{aligned}$$

and when $j' \neq j$, they are:

$$\begin{aligned} & \frac{0 - e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\cdot e^{\sum_{k=0}^{K} x_{ik}\beta_{kj'}}\cdot x_{ik'}}{\left(1+\sum_{j=1}^{J-1} e^{\sum_{k=0}^{K} x_{ik}\beta_{kj}}\right)^2} \\ = & -\pi_{ij}x_{ik'}\pi_{ij'} \qquad (36)\end{aligned}$$

We can now express the matrix of second partial derivatives for the multinomial logistic regression model as:

$$\begin{aligned}\frac{\partial^2 l(\beta)}{\partial\beta_{kj}\partial\beta_{k'j'}} &= -\sum_{i=1}^{N} n_i x_{ik}\pi_{ij}(1-\pi_{ij})x_{ik'} \qquad j' = j \\ &= \sum_{i=1}^{N} n_i x_{ik}\pi_{ij}\pi_{ij'}x_{ik'} \qquad j' \neq j \qquad (37)\end{aligned}$$

Scott A. Czepiel
http://czep.net/contact.html

2.2.3 Newton-Raphson

To illustrate the iterative procedure of Newton-Raphson as it applies to the multinomial logistic regression model, we need an expression for Eq. 20. Let $\boldsymbol{\mu}$ be a matrix with N rows and $J-1$ columns, the same dimensions as $\boldsymbol{y}$ and $\boldsymbol{\pi}$, with elements equal to $n_i\pi_{ij}$. Then, Eq. 21 expresses a matrix with $K+1$ rows and $J-1$ columns, the same dimensions as $\boldsymbol{\beta}$. By matrix multiplication, the elements of this matrix are equivalent to those derived in Eq. 32.

The expression for the matrix of second partial derivatives is somewhat different from that derived in the binomial case, since the equations in Eq. 37 differ depending on whether or not $j' = j$.

For the diagonal elements of the matrix of second partial derivatives, i.e., where $j' = j$, let $\boldsymbol{W}$ be a square matrix of order N, with elements $n_i\pi_{ij}(1-\pi_{ij})$ on the diagonal and zeros everywhere else. Then, Eq. 22 generates a $K+1 \times K+1$ matrix. However, we can only use this formulation for the diagonal elements. For the off-diagonal elements, where $j' \neq j$, define $\boldsymbol{W}$ as a diagonal matrix with elements $n_i\pi_{ij}\pi_{ik}$, and use the *negative* of the expression in Eq. 22.

Using this dual formulation for $\boldsymbol{W}$, each step of the Newton-Raphson method can proceed as in the binomial logistic regression model, using Eq. 23.

3 Implementation

The following is an outline of a skeletal implementation for logistic regression using the C programming language. The routines presented here focus on a practical application of the mathematical theory outlined in the section above. To be useful in a real program, they would need to incorporate user and data interface methods, including a means of selecting variables and constructing an appropriate design matrix. Different methods of parameterization—dummy coding, full-rank center-point, direct specification, and interaction effects—are not discussed here. We will also not take into accout imputation strategies for dealing with missing values or any other subtleties for pre-processing the data or optimizing the modeling process. Also omitted are the calculation of goodness-of-fit tests and significance tests of the parameter estimates. There are also a number of auxilliary functions which are beyond the scope of this document and will not be covered here. For further details in any of these areas, the reader is strongly encouraged to investigate the texts included in the References sec-

tion. Furthermore, we will not deal with error handling, memory allocation, or compiler optimization techniques.

The function `mlelr` is used as the entry point to set up the iterations that will take place in the `newton_raphson` function. At minimum, this function will require the following arguments:

```
int mlelr (
    int      J,     /* number of discrete values of y */
    int      N,     /* number of populations */
    int      K,     /* number of columns in x */
    double  *n,     /* population counts - length N */
    double **y,     /* dv counts - N rows and J-1 columns */
    double **pi,    /* probabilities - N rows and J-1 columns */
    double **x,     /* design matrix - N rows and K+1 columns */
    double  *beta   /* parameters - K+1 * J-1 rows */
    double  *xrange /* range of x - length K+1 */
    ) {
```

We will try to abide by the naming convention established earlier, so far as it remains convenient. Note that `n`, `beta`, and `xrange` are declared as pointers to double. In C, we can access these using array subscripts, such as `n[i]`. The variables `y`, `pi`, and `x` are each declared as pointer to pointer to double, thus creating the effect of a matrix which can be accessed using syntax like `pi[i][j]`. The array `xrange` is needed in the test for parameter estimates tending toward infinity. It should contain as many elements as there are columns in the design matrix, with each element specifying the range from lowest to highest value for the corresponding independent variable. In this routine, we will treat our parameter matrix $\boldsymbol{\beta}$ as a vector, where each column is appended below the first. This makes it easier to construct the matrix involving second derivatives, which we will introduce below:

```
    /* local variables */
    int      i,j,k;
    const int max_iter = 30;
    const double eps = 1e-8;
    int      iter = 0;
    int      converged = 0;
    double  *beta_old;
    double  *beta_inf;
    double **xtwx;
```

```
double   loglike = 0;
double   loglike_old = 0;
```

`max_iter` is the maximum number of Newton-Raphson iterations to try before assuming that the model does not converge. `eps`, short for *epsilon*, is the threshold below which parameter estimates from subsequent iterations are assumed to be equal. When all the differences from the current to the prior iteration are less than `eps`, we assume the model has converged. The two arrays `beta_old` and `beta_inf` need to have space allocated to match the dimensions of `beta`. `beta_old` is used to store the parameter estimates from a prior iteration before starting a new one, and `beta_inf` is used in the test for infinite parameters. The matrix `xtwx`, read $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$, needs to be set up with `(K+1)*(J-1)` rows and columns.

```
/* allocate space for local arrays */
.
.   /* malloc code here */
.
/* initialize parameters to zero */
for (k = 0; k < (K + 1) * (J - 1); k++) {
    beta[k] = 0;
    beta_inf[k] = 0;
    for (j = 0; j < (K + 1) * (J - 1); j++) {
        xtwx[k][j] = 0;
    }
}
```

An alternative approach would be to run a linear regression of $\log(\pi_{ij}/\pi_{iJ})$ with the design matrix to obtain starting values for each `beta[k]`. This initially adds more computational cycles, but will usually reduce the number of Newton-Raphson iterations needed to bring the model to convergence. Now we can set up the main loop as follows:

```
/* main loop */
while (iter < max_iter && !converged) {

    /* copy beta to beta_old */
    for (k = 0; k < (K + 1) * (J - 1); k++) {
        beta_old[k] = beta[k];
    }
```

The main loop will run until the parameter estimates converge, or the number of iterations reaches the maximum allowed. The first step in the loop is to store the current values of `beta` in `beta_old`. The next step is to perform one iteration:

```
/* run one iteration of newton_raphson */
loglike_old = loglike;
loglike = newton_raphson(J,N,K,n,y,pi,x,beta,xtwx);
```

Our `newton_raphson` function returns the value for the log likelihood function evaluated at the current iteration. In a production system, it would be much safer to let this function return an error status code, since a number of problems can arise within that routine that would then need to be handled here.

```
/* test for decreasing likelihood */
if (loglike < loglike_old && iter > 0) {
    .
    .  /* code to backtrack here */
    .
}
```

After returning from an iteration, and verifying that the iteration completed successfully, the next step is to check whether the value of the log likelihood function has decreased since the previous iteration. If so, we can include code to backtrack in a series of sub-iterations which successively halve the distance between the current and prior iteration until a point is reached where the likelihood does increase. If such a point is not found after some number of sub-iterations, we conclude that the model had converged at the prior iteration, although it may be the case that the iterative procedure has degenerated and strayed too far from the true root. It would definitely be necessary to inform the user that this occurred.

```
/* test for infinite parameters */
for (k = 0; k < (K + 1) * (J - 1); k++) {
    if (beta_inf[k] != 0) {
        beta[k] = beta_inf[k];
    }
    else {
        if ((fabs(beta[k]) > (5 / xrange[k])) &&
            (sqrt(xtwx[k][k]) >= (3 * fabs(beta[k])))) {
```

```
                    beta_inf[k] = beta[k];
                }
            }
        }
```

The above code handles a test for parameter estimates tending to infinity as outlined in the section on caveats. If an element of the array `betainf` is not zero, then the value stored there is the last known value for `beta[k]` before it was assumed to be infinity. We hold it constant in all subsequent iterations so that it no longer interferes with the test for convergence. Note that the standard error of each `beta[k]` is the square root of the corresponding diagonal element of `xtwx`.

```
        /* test for convergence */
        converged = 1;
        for (k = 0; k < (K + 1) * (J - 1); k++) {
            if (fabs(beta[k] - beta_old[k]) >
                eps * fabs(beta_old[k])) {
                    converged = 0;
                    break;
            }
        }
        iter++;
    }  /* end of main loop */
```

The test for convergence requires every new parameter estimate to differ by the prior estimate by less than the value for `eps`. If this condition is not satisfied, the main loop will execute again.

The function that handles the Newton-Raphson iterations begins:

```
double newton_raphson(int J, int N, int K,
    double *n, double **y, double **pi, double **x,
    double *beta, double **xtwx) {

    /* local variables */
    int i, j, jj, jprime, k, kk, kprime;
    double *beta_tmp;
    double **xtwx_tmp;
    double loglike;
    double denom;
```

```
double *numer;  /* length J-1 */
double tmp1, tmp2, w1, w2;
```

The variables `beta_tmp` and `xtwx_tmp` are temporary versions of the variables they resemble that will be used to build the new values for the current iteration. Before continuing, these would need to have space allocated for them with `malloc`, and each element should be initialized to zero. The variable `loglike` will be used to store the return value for this function.

In the next step, we establish a loop for each row in the design matrix. This is a very busy loop where most of the work of Newton-Raphson will be accomplished. Refer to Eq. 23 as a reminder of the calculations that need to be made. Upon first entering the loop, we calculate the values for πij for the given row, i. This is done using Eq. 25.

```
/* main loop for each row in the design matrix */
for (i = 0; i < n; i++) {

    /* matrix multiply one row of x * beta */
    denom = 1;
    for (j = 0; j < J - 1; j++) {
        tmp1 = 0;
        for (k = 0; k < K + 1; k++) {
            tmp1 += x[i][k] * beta[j*(K+1)+k];
        }
        numer[j] = exp(tmp1);
        denom += numer[j];
    }

    /* calculate predicted probabilities */
    for (j = 0; j < J - 1; j++) {
        pi[i][j] = numer[j] / denom;
    }
```

Note that since we are treating `beta` as a vector, we need to offset its index by the j^{th} multiple of $K + 1$ before adding `k` to its index in the matrix multiplication.

Next, we can calculate the i^{th} row's contribution to the value of the log likelihood function. To do this, we need to consider all the terms in

Eq. 26, including the factorial terms that were omitted in the derivation of the kernel of the log likelihood. Taking the log of Eq. 26 yields:

$$\sum_{i=1}^{N}\left[\log(n_i!) + \sum_{j=1}^{J} y_{ij}\log(\pi_{ij}) - \log(y_{ij}!)\right] \tag{38}$$

Since it is highly dangerous to evaluate factorials directly, we can use the gamma approximation, where $\Gamma(x+1) \approx x!$. Thus, Eq. 38 becomes:

$$\sum_{i=1}^{N}\left[\log(\Gamma(n_i+1)) + \sum_{j=1}^{J} y_{ij}\log(\pi_{ij}) - \log(\Gamma(y_{ij}+1))\right] \tag{39}$$

We implement this in the following code:

```
/* add log likelihood for current row */
loglike += log_gamma(n[i] + 1);
for (j = 0, tmp1 = 0, tmp2 = 0; j < J - 1; j++) {
    tmp1 += y[i][j];
    tmp2 += pi[i][j];
    loglike = loglike - log_gamma(y[i][j]+1) +
        y[i][j] * log(pi[i][j]);
}

/* Jth category */
loglike = loglike - log_gamma(n[i]-tmp1+1) +
    (n[i]-tmp1) * log(1-tmp2);
```

The details of the `log_gamma` function are beyond the scope of this article. For more information, see [11] and [2]. Since we never explicitly store either y_{iJ} or π_{iJ}, we use `tmp1` to add the first $J-1$ values of y_{ij} and `tmp2` to add the first $J-1$ values of π_{ij}.

The following code builds the matrices in the last two terms of Eq. 23 by adding the contribution of the i^{th} row to the first and second derivatives of the log likelihood equations.

```
/* add first and second derivatives */
for (j = 0, jj = 0; j < J - 1; j++) {
    tmp1 = y[i][j] - n[i] * pi[i][j];
    w1 = n[i] * pi[i][j] * (1 - pi[i][j]);
    for (k = 0; k < K + 1; k++) {
        beta_tmp[jj] += tmp1 * x[i][k];
```

```
                kk = jj - 1;
                for (kprime = k; kprime < K + 1; kprime++) {
                    kk++;
                    xtwx_tmp[jj][kk] +=
                        w1 * x[i][k] * x[i][kprime];
                    xtwx_tmp[kk][jj] = xtwx_tmp[jj][kk];
                }
                for (jprime = j + 1; jprime < J - 1; jprime++) {
                    w2 = -n[i] * pi[i][j] * pi[i][jprime];
                    for (kprime = 0; kprime < K + 1; kprime++) {
                        kk++;
                        xtwx_tmp[jj][kk] +=
                            w2 * x[i][k] * x[i][kprime];
                        xtwx_tmp[kk][jj] = xtwx_tmp[jj][kk];
                    }
                }
                jj++;
            }
        }
    } /* end loop for each row in design matrix */
```

In the code above, `jj` maintains a running counter of the current row of `beta_tmp` and `xtwx_tmp`. The variable `kk` is used to maintain the current column index of `xtwx_tmp`. The outer loop is executed for each value of `j`. First, $y_{ij} - n_i\pi_{ij}$ is calculated and stored in `tmp1`. Then, `w1` is calculated as $n_i\pi_{ij}(1 - \pi_{ij})$, which is the i^{th} diagonal element in the matrix $\boldsymbol{W}$ when $j' = j$. The inner loop is executed for each `k`. `beta_tmp[jj]` is incremented by `tmp1 * x[i][k]`, which, after all rows are taken into account, will result in the first derivative term in Eq. 23, $\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{\mu})$.

The first loop over `kprime` adds the current contribution to the second derivative matrix, $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$, where $j' = j$. We start this loop at `k` rather than zero because the $K + 1 \times K + 1$ submatrix for the current value of j is symmetric, and once we calculate `xtwx_tmp[jj][kk]`, we also know `xtwx_tmp[kk][jj]`. Finally, a loop for each $j' \neq j$ is set up to repeat the loop over `kprime` using the alternate formulation for $\boldsymbol{W}$ as noted in Eq. 37.

The final step in the Newton-Raphson routine is to invert $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$, and solve for the next set of elements $\boldsymbol{\beta}$. Matrix inversion is a complicated subject constituting a major sub-field of numerical analysis unto itself, and we will not cover the details here. Since $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$ is symmetric and, in most cases, positive definite, the fastest way to invert it is through a Cholesky

factorization and backsubstitution. For more information, see [7] and [11]. For now, we will assume that the original matrix, stored in the local variable `xtwx_tmp`, is still intact, and its inverse has been computed and stored in `xtwx`. Note that since `xtwx` is passed to `newton_raphson` as a pointer, its newly modified contents will be accessible to the calling routine when this one returns. `xtwx` will be needed in the main routine `mlelr` as part of the test for infinite parameters, as well as any future implementations of significance tests that require the standard errors of the parameter estimates.

At last, we have all the information we need to apply Eq. 23. The direct approach would be to perform the cross multiplication of `xtwx` and `beta_tmp` and add the result to the contents of `beta`, which stores the parameter estimates of the (now previous) iteration. However, the additive terms are likely to be very small, and as a result the direct approach is highly susceptible to roundoff error. To maintain precision, we take advantage of the following identity:

$$\begin{aligned} \boldsymbol{\beta}^{(1)} &= [\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}]^{-1} \cdot [\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} \cdot \boldsymbol{\beta}^{(0)} + \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{\mu})] \\ &= \boldsymbol{I} \cdot \boldsymbol{\beta}^{(0)} + [\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}]^{-1} \cdot \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{\mu}) \end{aligned} \tag{40}$$

which is equivalent to Eq. 23. We will do this in two steps. First, by computing the second term [bracketed] in the first line of Eq. 40, then by post-multiplying that term with the newly inverted `xtwx`:

```
/* compute xtwx * beta(0) + x(y-mu) */
for (i = 0; i < (K + 1) * (J - 1); i++) {
    tmp1 = 0;
    for (j = 0; j < (K + 1) * (J - 1); j++) {
        tmp1 += xtwx_tmp[i][j] * beta[j];
    }
    beta_tmp[i] += tmp1;
}

/* solve for new betas */
for (i = 0; i < (K + 1) * (J - 1); i++) {
    tmp1 = 0;
    for (j = 0; (K + 1) * (J - 1); j++) {
        tmp1 += xtwx[i][j] * beta_tmp[j];
    }
    beta[i] = tmp1;
}
```

References

[1] Agresti, A. 1990. *Categorical Data Analysis*. New York: John Wiley.

[2] Cody, W.J. and Hillstrom, K.E. 1967. "Chebyshev Approximations for the Natural Logarithm of the Gamma Function," *Mathematics of Computation*, vol. 21, pp. 198-203.

[3] Schafer, J.L. 2001. Lecture Notes for *Statistics 544: Categorical Data Analysis I*, Fall 2001. Penn State Univ. http://www.stat.psu.edu/∼jls/

[4] Draper, N.R. and Smith, H. 1981. *Applied Regression Analysis*. 2nd ed. New York: John Wiley.

[5] Dobson, A.J. 2002. *An Introduction to Generalized Linear Models*. 2nd ed. Boca Raton, FL: Chapman & Hall/CRC.

[6] Eliason, S.R. 1993. *Maximum Likelihood Estimation: Logic and Practice*. Sage University Paper series on Quantitative Applications in the Social Sciences, series no. 07-096. Newbury Park, CA: Sage.

[7] Golub, G.H. and Van Loan, C.F. 1996. *Matrix Computations*. 3rd ed. Baltimore: Johns Hopkins.

[8] Long, J.S. 1997. *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: Sage.

[9] Nelder, J.A. and Wedderburn, R.W.M. 1972. "Generalized Linear Models," *Journal of the Royal Statistical Society*, Series A, vol. 135, pp. 370-384.

[10] Powers, D.A. and Xie, Y. 2000. *Statistical Methods for Categorical Data Analysis*. San Diego, CA: Academic Press.

[11] Press, W.H., et al. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. Cambridge, UK: Cambridge.

[12] Ross, S. 1998. *A First Course in Probability*. 5th ed. Upper Saddle River, NJ: Prentice-Hall.

[13] SAS Institute Inc. 1989. *SAS/STAT User's Guide, Version 6*. 4th ed. Cary, NC: SAS Institute Inc.

[14] Spivak, M. 1980. *Calculus*. 2nd ed. Houston, TX: Publish or Perish, Inc.

Scott A. Czepiel
http://czep.net/contact.html