Tutorial on Multivariate Logistic Regression

Javier R. Movellan

July 23, 2006

1 Motivation

The last few years have seen a resurgence of interest in the Machine Learning community for main-effect models, i.e., models without interaction effects. This is due to the emergence of learning algorithms, like SVMs and ADABoost that work well with very high dimensional representations, avoiding the need for modeling interaction effects. The goal of this tutorials is to introduce the most important main effect regression models, how they relate to each other and common algorithms used for training them.

2 General Model

Our goal is to train a well defined model based on examples of input-output pairs. In our notation the inputs will be n-dimensional, the outputs will be c-dimensional and our training sample will consist of m input output pairs. For convenience we organize the example inputs as an $m \times n$ matrix x. The corresponding example outputs are organized as a $c \times m$ matrix y. Our goal is to predict the rows of y using the rows of x. The models under consideration produce a intermediary linear transformation of the inputs and make predictions about y based on that transformation. Let λ be an $n \times c$ matrix and let $\hat{y} = x\lambda$ represent the intermediary linear transformation of the data.

We will evaluate the optimality of $\hat{y}$, and thus of λ, using the following criterion

$$\Phi(\lambda) = -\sum_{j=1}^{m} w_j \rho_j(y, \hat{y}) + \frac{\alpha}{2} \sum_{k=1}^{c} \lambda_k^T \lambda_k \tag{1}$$

where $y_{j\cdot}, \hat{y}_{j\cdot}$ represent the j^{th} rows of y and $\hat{y}$, ρ_j is an error function that measures the mismatch between the j^{th} rows of y and $\hat{y}$. For example $\rho_j(y, \hat{y}) = \sum_{k=1}^{c}(y_{j,k} - \hat{y}_{j,k})^2$. We let $\rho = (\rho_1, \rho_m)^T$. The terms $w_1, \ldots, w_m$ are positive weights that capture the relative importance of each of the m input-output pairs, and α is a positive constant. Informally, the first term can be seen as a negative log-likelihood function, capturing the degree of match between the data and the model, the second term can be interpreted as a negative log prior over λ.

A standard approach for minimizing Φ is the Newton-Raphson algorithm (See Appendix B), which calls for computation of the gradient and the Hessian matrix (See Appendix A).

Let λ_i represent the i^{th} row of λ. The gradient can be computed using the chain rule (See Appendix A)

$$\nabla_{\lambda_i} \frac{\alpha}{2} \sum_{k=1}^{c} \lambda_k^T \lambda_k = \alpha \lambda_i \tag{2}$$

$$\nabla_{\lambda_i} \sum_{j=1}^{m} w_j \rho_j(y, \hat{y}) = \nabla_{\lambda_i} \hat{y}_i \nabla_{\hat{y}_i} \rho \nabla_{\rho} \sum_{j=1}^{m} w_j \rho_j(y, \hat{y}) = x^T w \Psi_i \tag{3}$$

where w is a diagonal matrix with diagonal elements $w_1, \ldots, w_m$ and

$$\Psi_i = \begin{pmatrix} \frac{\partial \rho_1}{\partial \hat{y}_{1i}} \\ \vdots \\ \frac{\partial \rho_m}{\partial \hat{y}_{mi}} \end{pmatrix} \tag{4}$$

Thus

$$\nabla_{\lambda_i} \Phi = x^T w \Psi_i + \alpha \lambda_i \tag{5}$$

The overall gradient vector follows $\nabla_\lambda \Phi = (\nabla_{\lambda_1} \Phi, \ldots, \nabla_{\lambda_n} \Phi)^T$. The chain rule can be used to compute the component matrices of the overall Hessian matrix

$$\nabla_{\lambda_j} \nabla_{\lambda_i} \Phi = \nabla_{\lambda_j} \alpha \lambda_i + \nabla_{\lambda_j} x^T w \Psi_i \tag{6}$$

$$\nabla_{\lambda_j} \alpha \lambda_i = \delta_{ij} \alpha I_n \tag{7}$$

where δ_{ik} is the Kronecker delta function (0 valued unless $i = k$ in which case it takes value 1) and I_n is the $n \times n$ identity matrix.

$$\nabla_{\lambda_j} x^T w \Psi_i = \nabla_{\lambda_j} \hat{y}_j \nabla_{\hat{y}_j} \Psi_i \nabla_{\Psi_i} x^T w \Psi_i = x^T \Psi'_{ij} w x \tag{8}$$

and

$$\Psi'_{ij} = \begin{pmatrix} \frac{\partial^2 \rho_1}{\partial \hat{y}_{1j} \partial \hat{y}_{1i}} & 0 & \cdots & 0 \\ 0 & \frac{\partial^2 \rho_2(y, \hat{y})}{\partial \hat{y}_{2j} \partial \hat{y}_{2i}} & \ddots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial^2 \rho_m}{\partial \hat{y}_{mj} \partial \hat{y}_{mi}} \end{pmatrix} \tag{9}$$

Thus

$$\nabla_{\lambda_j} \nabla_{\lambda_i} \Phi = x^T \Psi'_{ij} w x + \delta_{ij} \alpha I_n \tag{10}$$

The overall Hessian matrix follows

$$\nabla_\lambda \nabla_\lambda \Phi = \begin{pmatrix} \nabla_{\lambda_1} \nabla_{\lambda_1} \Phi & \cdots & \nabla_{\lambda_1} \nabla_{\lambda_1} \Phi \\ \vdots & \ddots & \vdots \\ \nabla_{\lambda_n} \nabla_{\lambda_1} \Phi & \cdots & \nabla_{\lambda_n} \nabla_{\lambda_n} \Phi \end{pmatrix} \tag{11}$$

3 Linear Ridge Regression

Without loss of generality we will examine the case with a single response variable, i.e., $c = 1$. Our error function will be the sum of squared errors, i.e,

$$\rho_j(y, \hat{y}) = \frac{1}{2}(y_j - \hat{y}_j)^2 \tag{12}$$

Thus

$$\Psi_i = -\begin{pmatrix} \frac{\partial \rho_1}{\partial \hat{y}_{1i}} \\ \vdots \\ \frac{\partial \rho_m}{\partial \hat{y}_{mi}} \end{pmatrix} = \begin{pmatrix} y_1 - \hat{y}_1 \\ \vdots \\ y_m - \hat{y}_m \end{pmatrix} = y - x\lambda \tag{13}$$

The gradient follows

$$\nabla_\lambda \Phi = x^T w(y - x\lambda) + \alpha\lambda \tag{14}$$

Setting the equation to zero results on a linear equation on λ, that can be solved analytically

$$\hat{\lambda} = \left(x^T wx + \alpha I_n\right)^{-1} x^T wy \tag{15}$$

where $\hat{\lambda}$ represents the optimal value of λ.

4 Robust Regression

$$\frac{\alpha}{2}\lambda^T\lambda + \sum_{j=1}^{m} w_j \rho(y_j - \hat{y}_j) \tag{16}$$

where $\hat{y} = x\lambda$. Where ρ typically penalizes extreme deviations between y and $\hat{y}$ less than the squared error function does. It is easy to show that in this case

$$\nabla_\lambda \Phi = -x^T w\Psi + \alpha\lambda \tag{17}$$

where $\Psi_j = \rho'(y_j - \hat{y}_j)$ the derivative of the error function. This derivative is known as the *influence function* for it controls the influence of each example on the final solution. Note if $\rho(y_j\hat{y}_j) = (y_h - \hat{y}_j)^2$ then $\Psi = y - \hat{y}$. If ρ has no second derivative, a common procedure to find an optimal λ involves the use of the following heuristic. Note that (31) can be expressed as follows:

$$\nabla_\lambda \Phi = -x^t \tilde{w}(y - \hat{y}) + \lambda \tag{18}$$

where $\tilde{w}$ is an $m \times m$ diagonal matrix such that $w_{jj} = w_j\Psi_j/(y_j - \hat{y}_j)$. Setting this gradient to zero, and disregarding the fact that w is a function of λ, results on a linear weighted least squares problem whose solution we have already seen. This iterative version of weighted least squares we start with an arbitrary value λ and apply the following iteration:

$$\lambda_{t+1} = (x^t \tilde{w}_t x + \alpha I_n)^{-1} x^t \tilde{w}_t y \tag{19}$$

where w_t is the matrix of weights obtained using λ_t.

If ρ has second derivative, then the Hessian matrix exists and has the following form

$$\nabla_\lambda \nabla_\lambda \Phi = x^T \Psi' wx + \alpha I_n \tag{20}$$

Where Ψ is an $m \times m$ diagonal matrix such that $\Psi_{jj} = \rho^{''}(y_j - \hat{y}_j)$, the second derivative of the error function. The Newton-Raphson algorithm calls for the following iteration

$$\lambda_{t+1} = \lambda_t + (x^t \Psi' x)^{-1} x^t \Psi \tag{21}$$

which can also be casted as weighted least squares solution with weight matrix Ψ' and desired response vector $z = x\lambda_t + \Psi^{-1} x^T \Psi$

5 Multinomial Logistic Regression

Let

$$\rho_j(y, \hat{y}) = \sum_{k=1}^{c} y_{jk} \log h_k(\hat{y}_j) \tag{22}$$

where

$$h_k(\hat{y}_j) = \frac{e^{\hat{y}_{jk}}}{\sum_{i=1}^{c} e^{\hat{y}_{ji}}} \tag{23}$$

Note for any $u = (u_1, \ldots, u_c)^T \in \mathbb{R}^c$

$$\frac{\partial h_k(u)}{\partial u_i} = h_k(u) \frac{\partial \log h_k(u)}{\partial u_i} = h_k(u)(\delta_{ik} - h_i(u)) \tag{24}$$

Thus

$$\frac{\partial \rho_j(y, \hat{y})}{\partial \hat{y}_{ji}} = \sum_{k=1}^{c} \frac{y_{jk}}{h_k(\hat{y}_{jk})} h_k(\hat{y}_{jk})(\delta_{ik} - h(\hat{y}_{ji})) = y_{ji} - h_i(\hat{y}_{ji}) \tag{25}$$

$$\Psi_i = y_i - h_i(\hat{y}) \tag{26}$$

and

$$\Psi'_{ki} = \begin{pmatrix} \hat{y}_{1k}(\delta_{ki} - \hat{y}_{1i}) & 0 & \cdots & 0 \\ 0 & \hat{y}_{2k}(\delta_{ki} - \hat{y}_{2i}) & \ddots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{y}_{mk}(\delta_{ki} - \hat{y}_{mi}) \end{pmatrix} \tag{27}$$

5.1 Iteratively Reweighted Least Squares

When there are only two response alternatives and $\alpha = 0$ the Newton- Raphson algorithm simplifies as follows (See Appendix B)

$$\lambda(t+1) = \lambda(t) + \left(x^T \tilde{w}(t) x\right)^{-1} x^T w(y - \hat{y}(t)) \tag{28}$$

where $\lambda(t)$ represents the n-dimensional vector of weights at iteration t, y and $\hat{y}(t)$ are m-dimensional vectors containing the desired and predicted probabilities for the first response alternative and $w(t)$ is an $m \times m$ matrix $w = \nabla_{x\lambda}\hat{y}$, i.e., $w_{ii} = \hat{y}_i(1 - \hat{y}_i)$. Some simple algebra shows that this iteration can be casted as the solution to a weighted linear regression problem

$$\lambda(t+1) = \left(x^T \tilde{w}(t) x\right)^{-1} x^T \tilde{w}(t) z(t) \tag{29}$$

where $z(t) = x\lambda(t) + w\tilde{w}^{-1}(t)(y - \hat{y})$ is a modified target variable.

It is common practice to start the iterations at the zero vector, i.e., $\lambda_0 = 0$. It is easy to show that in such case the matrix w has constant diagonal terms and the solution obtained after the first iteration is the standard least squares solution. Successive iterations change the weight assigned to each example.

6 Robust Regression

In robust regression the criterion function is of the form

$$\Phi(\lambda) = \frac{\alpha}{2}\lambda^T\lambda + \sum_{j=1}^{m} w_j \rho(y_j - \hat{y}_j) \tag{30}$$

where $\hat{y} = x\lambda$. Where ρ typically penalizes extreme deviations between y and $\hat{y}$ less than the squared error function does. It is easy to show that in this case

$$\nabla_\lambda \Phi = -x^T w \Psi + \alpha\lambda \tag{31}$$

where $\Psi_j = \rho'(y_j - \hat{y}_j)$ the derivative of the error function. This derivative is known as the *influence function* for it controls the influence of each example on the final solution. Note if $\rho(y_j\hat{y}_j) = (y_h - \hat{y}_j)^2$ then $\Psi = y - \hat{y}$. If ρ has no second derivative, a common procedure to find an optimal λ involves the use of the following heuristic. Note that (31) can be expressed as follows:

$$\nabla_\lambda \Phi = -x^t \tilde{w}(y - \hat{y}) + \lambda \tag{32}$$

where $\tilde{w}$ is an $m \times m$ diagonal matrix such that $w_{jj} = w_j\Psi_j/(y_j - \hat{y}_j)$. Setting this gradient to zero, and disregarding the fact that w is a function of λ, results on a linear weighted least squares problem whose solution we have already seen. This iterative version of weighted least squares we start with an arbitrary value λ and apply the following iteration:

$$\lambda_{t+1} = (x^t\tilde{w}_t x + \alpha I_n)^{-1} x^t \tilde{w}_t y \tag{33}$$

where w_t is the matrix of weights obtained using λ_t.

If ρ has second derivative, then the Hessian matrix exists and has the following form

$$\nabla_\lambda \nabla_\lambda \Phi = x^T \Psi' w x + \alpha I_n \tag{34}$$

Where Ψ is an $m \times m$ diagonal matrix such that $\Psi_{jj} = \rho''(y_j - \hat{y}_j)$, the second derivative of the error function. The Newton-Raphson algorithm calls for the following iteration

$$\lambda_{t+1} = \lambda_t + (x^t\Psi' x)^{-1} x^t \Psi \tag{35}$$

which can also be casted as weighted least squares solution with weight matrix Ψ' and desired response vector $z = x\lambda_t + \Psi^{-1}x^T\Psi$

7 Support Vector Machines

In support vector machines (SVMs) there typically is a single response variable $c = 1$ and the error function has the following form

$$\rho(y_i, \hat{y}_i) = -\max\{1 - y_i\hat{y}, 0\} \tag{36}$$

where $y \in \{0, 1\}$. It is easy to show that as $\alpha \to 0$, the solution is the hyperplane that maximizes the minimum distance from the training examples to the hyperplane.

It is interesting to note tha tthe SVM error function and the logistic error function are are very similar, the logistic error function being a smooth version of the SVM error function.

8 ADABoost and the Backfitting algorithm

Under construction.

A Vector Calculus

Definition Gradient Matrix: Let $y = f(x)$ where $y \in \mathbb{R}^n$, $x \in \mathbb{R}^m$. The gradient of y with respect to x, symbolized $\nabla_x\, y$, is an $m \times n$ matrix defined as follows

$$\nabla_x y = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{pmatrix} \tag{37}$$

Gradient of quadratic functions: Let $y = x^T ax + bx + c$ where $x \in \mathbb{R}^n$. It can be shown that

$$\nabla_x\, y = b^T + (a + a^T)x \tag{38}$$

where a^T is the transpose of a. Note $y \in \mathbb{R}^n$. If a is symmetric then $a + a^T = 2a$.

Chain Rule: Let $y = f(x)$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ and $z = h(y)$, $z \in \mathbb{R}^o$. Then

$$\nabla_x\, z = \nabla_x\, y\, \nabla_y\, z \tag{39}$$

Example: Let $y = (x - \mu)^T a(x - \mu)$ where $x, \in \mathbb{R}^n$ and μ is a constant vector. Then

$$\nabla_y\, x = \nabla_x\, (x - \mu)\, \nabla_{(x-\mu)}\, y = I_n a(x - \mu) = a(x - \mu). \tag{40}$$

where I_n is the $n \times n$ identity matrix.

Definition Hessian Matrix: Let $y = f(x)$, $y \in \mathbb{R}$, $x \in \mathbb{R}^n$. The matrix

$$\nabla_x \nabla_x f(x) = \begin{pmatrix} \frac{\partial^2 y}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 y}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} \end{pmatrix} \tag{41}$$

is called the Hessian matrix of y. **Example**: Let $y = x^T ax + bx + c$ then

$$\nabla_x\, \nabla_x y = \nabla_x ax + b^T = a \tag{42}$$

B Newton-Raphson Optimization Method

Let $y = f(x)$, for $y \in \mathbb{R}$, $x \in \mathbb{R}^n$. The Newton-Raphson algorithm is an iterative method for optimizing y. We start the process at an arbitrary point $x_0 \in \mathbb{R}^n$. Let $x_t \in \mathbb{R}^n$ represent the state

of the algorithm at iteration t. We approximate the function f using the linear and quadratic terms of the Taylor expansion of f around x_t.

$$\hat{f}_t(x) = f(x_t) + \nabla_x f(x_t)(x - x_t)^T + \frac{1}{2}(x - x_t)^T \left(\nabla_x \nabla_x f(x_t)\right)(x - x_t) \tag{43}$$

and then we then find the extremum of $\hat{f}_t$ with respect to x and move directly to that extremum. To do so note that

$$\nabla_x \hat{f}_t(x) = \nabla_x f(x_t) + \left(\nabla_x \nabla_x f(x_t)\right)(x - x_t) \tag{44}$$

We let $x(t+1)$ be the value of x for which $\nabla_x \hat{f}_t(x) = 0$

$$x_{t+1} = x_t + \left(\nabla_x \nabla_x f(x_t)\right)^{-1} \nabla_x f(x_t) \tag{45}$$

It is useful to compare the Newton-Raphson method with the standard method of gradient ascent. The gradient ascent iteration is defined as follows

$$x_{t+1} = x_t + \epsilon I_n \nabla_x f(x_t) \tag{46}$$

where ϵ is a small positive constant. Thus gradient descent can be seen as a Newton-Raphson method in which the Hessian matrix is approximated by $\frac{1}{\epsilon} I_n$.

C Gauss-Newton Optimization Method

Let $f(x) = \sum_{i=1}^{n} r_i(x)^2$ for $r_i : \mathbb{R}^n \to \mathbb{R}$. We start the process with an arbitrary point $x_0 \in \mathbb{R}^n$. Let $x_t \in \mathbb{R}^n$ represent the state of the algorithm at iteration t. We approximate the functions r_i using the linear term of their Taylor expansion around x_t.

$$\hat{r}_i(x_t) = r_i(x_t) + (\nabla_x r_i(x_t))^T (x - x_t) \tag{47}$$

$$\hat{f}_t(x) = \sum_{i=1}^{n} (\hat{r}_i(x_t))^2 = \sum_{i=1}^{n} (r_i(x_t) - (\nabla_x r_i(x_t))^T x_t + (\nabla_x r_i(x_t))^T x)^2 \tag{48}$$

$$\tag{49}$$

Minimizing $\hat{f}_t(x)$ is a linear least squares problem of well known solution. If we let $y_i = (\nabla_x r_i(x_t))^T x_t - r_i(x_t)$ and $u_i = \nabla_x r_i(x_t)$ then

$$x_{t+1} = (\sum_{i=1}^{n} u_i u_i^T)^{-1} (\sum_{i=1}^{n} u_i y_i) \tag{50}$$

History

- The first version of this document was written by Javier R. Movellan, May 2002. It was 7 pages long.

- The document was made open source under GNU FDL license 1.1 as part of the Kolmogorov project on August 9 2002.
- Javier Movellan added Gauss-Newton Method on March 14, 2003.
- October 9, 2003. Javier R. Movellan changed the license to GFDL 1.2 and included an endorsement section.

References