** Work Sample

Chris Larson

9-15-2016

Problem statement: In this problem we're asked to fit a linear model to data with heterogeneous noise. The linear model is as follows:

$$y_i = a + bx_i + \epsilon_i \tag{1}$$

where y_i are the observed responses, x_i is the observed covariate, a and b are parameters to be optimized, and ϵ_i is an independent noise term that varies with x.

Brief outline of approach: I first use ordinary least squares to get initial estimates. I then explore two approaches based on: (1) Iteratively re-weighted least squares, and (2) maximum likelihood estimation optimized using gradient descent.

Solution: First I obtain initial point estimates $\hat{a}_{OLS}$ and $\hat{b}_{OLS}$ using ordinary least squares. The optimal parameters are obtained from the squared loss (ℓ_{OLS}) according to $\hat{B}_{OLS} \leftarrow \min_B (Y - XB)^2$. This is solved analytically for each dataset as follows:

$$\begin{aligned} \frac{d\ell_{OLS}}{dB} = 0 &= \frac{d}{dB}(Y - B^T X)^2 \\ &= \frac{d}{dB}(YY^T - 2Y^T XB + B^T X^T XB) \\ &= -2Y^T X - 2(X^T X)B \\ \hat{B}_{OLS} &= (X^T X)^{-1} X^T Y \end{aligned} \tag{2}$$

In this equation Y is an n x 1 column vector of responses, y_i, X is an n x 2 matrix with ones in its zeroth column and x_i's in its first column, and B is a 2 x 1 vector containing the intercept (a) and slope (b) terms. This is implemented in the script *v.py* with the function *run_OLS()*. The fits are shown visually in Figure 1a-e (black lines).

I next use iteratively re-weighted least squares with a Huber objective function. The Huber loss, $\ell_H(e)$, assumes the behavior of least squares for $|e_i| \leq k$, and that of the l1-loss for $|e_i| > k$. This function is shown in Equation 3.

$$\ell_H(e) = \begin{cases} \frac{1}{2}e^2 & if \quad |e| \leq k \\ k|e| - \frac{1}{2}k^2 & if \quad |e| > k \end{cases} \tag{3}$$

The parameters $\hat{B}_{IRLS}$ are obtained by minimizing $\ell_H(e)$ as follows:

$$\begin{aligned}\frac{d\ell_H(e)}{dB} = 0 &= \frac{d\ell_H(Y-XB)}{dB} \\ &= W(Y-XB)X \\ &= X^TWY - X^TWXB \\ \hat{B}_{IRLS} &= (X^TWX)^{-1}X^TWY\end{aligned} \tag{4}$$

where W is a n x n diagonal matrix with entries $d\ell_H/|e(i)|$. The quantity $d\ell_H$ is given by:

$$d\ell_H(e) = \begin{cases} |e_i| & if \quad |e| \leq k \\ k & otherwise. \end{cases} \tag{5}$$

At each timestep, t, the function *run_IRLS()* computes e_i^t and W^t from B^{t-1}. $\hat{B}_{IRLS}$ is then computed using Equation 4; this processes is repeated until convergence. Figure 1a-e (green lines) shows this fit ($k = 3$) plotted against the raw data; it is slightly more robust to the heavy-tailed noise than the OLS estimate. Figure 1f shows a plot of the residuals (absolute value) obtained from the IRLS fit; the noise are clearly not random. While IRLS is more robust than OLS, it still overfits to the observed noise. This is because each of the weights are (essentially) being set manually according to our semi-arbitrary choice of k.

Next I use maximum likelihood estimation to incorporate the heterogeneous variance into the model. I place two different priors on the distribution $p(D|\Theta)$, where Θ and D represent the model parameters and observed data, respectively. Borrowing from [M. Aitkin. JRSS-C, 1987], I start by giving the variance, σ^2, a new functional form:

$$\begin{aligned}\sigma^2(x) &= e^{\lambda^T Z} \\ \lambda &= \begin{bmatrix}\lambda_0 \\ \lambda_1\end{bmatrix} \\ Z &= \begin{bmatrix}1 & 1 & ... & 1 \\ x_1^2 & x_2^2 & ... & x_n^2\end{bmatrix}\end{aligned} \tag{6}$$

I choose this because $e(x)$ appears to be quadratic in x (by visual inspection of Figure 1f). Placing a Gaussian distribution on the likelihood, $p(D|B,\lambda)$, and then taking the negative of its logarithm yields a new objective function, ℓ_G, that has the familiar l2 norm term in addition to a new $\sigma^2(x)$ term parametrized by λ. This is shown below.

$$\begin{aligned}p_G(B,\lambda|D) &\sim N(Y - B^TX, \sigma^2(x)) \\ &= \frac{1}{\sqrt{2\pi\sigma^2(x)}} e^{\frac{(Y-B^TX)^2}{2\sigma^2(x)}} \\ -ln(p_G(B,\lambda|D)) &\propto ln(\sigma^2(x)) + \sigma^{-2}(x)(Y-B^TX)^2 \\ &= \lambda^T Z + e^{-\lambda^T Z}(Y-B^TX)^2 \\ \ell_G(a,b,\lambda_0,\lambda_1) &= \frac{1}{n}\sum_{i=1}^{n} \lambda_0 + \lambda_1 x_1^2 + e^{-\lambda_0 - \lambda_1 x_i^2}(y_i - a - bx_i)^2\end{aligned} \tag{7}$$

In these equations Y is a row vector of responses, y_i, and X is a 2 x n matrix with ones in its zeroth row and x_i's in its first row. I optimize against ℓ_G using gradient descent with a vanilla update rule: $\Theta^t \leftarrow \Theta^{t-1} - \eta\ \nabla_\Theta \ell(\Theta)$, where η is the learning rate. The function *train_gaussian*() in the script *v.py* computes $(\hat{B}_G, \hat{\lambda}_G)$. I used a learning rate of $\eta = 9.5 \times 10^{-5}$ to obtain the fits shown in Figure 1a-e (blue lines). The gradients used to update B and λ at each training step are shown below in Equation 8.

$$\begin{aligned}
\nabla_{\lambda_0}\ell_G &= 1 - \frac{1}{n}\sum_1^n e^{-\lambda_0-\lambda_1 x_i^2}(y_i - a - bx_i) \\
\nabla_{\lambda_1}\ell_G &= \frac{1}{n}\sum_1^n x_i^2(1 - e^{-\lambda_0-\lambda_1 x_i^2}(y_i - a - bx_i)) \\
\nabla_{a}\ell_G &= -\frac{2}{n}\sum_1^n e^{-\lambda_0-\lambda_1 x_i^2}(y_i - a - bx_i) \\
\nabla_{b}\ell_G &= -\frac{2}{n}\sum_1^n x_i e^{-\lambda_0-\lambda_1 x_i^2}(y_i - a - bx_i)
\end{aligned} \tag{8}$$

The maximum likelihood estimate (MLE) appears to be a better fit than IRLS and OLS. Given that it still has a l2 term in its loss function, however, we might expect a similar model with a l1 loss term to be influenced even less by the heavy-tail. I checked this by placing a Laplacian distribution on $p(D|B,\lambda)$, which yields an l1-loss term in its logarithm. The loss function, ℓ_L, is shown in Equation 9.

$$\begin{aligned}
p_L(B,\lambda|D) &\sim L(Y - B^TX, \sigma(x)) \\
&= \frac{1}{\sigma(x)}e^{\frac{|Y-B^TX|}{\sigma(x)}} \\
-ln(p_L(B,\lambda|D)) &\propto ln(\sigma(x)) + \sigma^{-1}(x)|Y - B^TX| \\
&= \frac{1}{2}\lambda^TZ + e^{-\frac{1}{2}\lambda^TZ}|Y - B^TX| \\
\ell_L(a,b,\lambda_0,\lambda_1) &= \sum_{i=1}^n \frac{1}{2}(\lambda_0 + \lambda_1 x_1^2) + e^{-\frac{1}{2}(-\lambda_0-\lambda_1 x_i^2)}|y_i - a - bx_i|
\end{aligned} \tag{9}$$

The gradients, $\nabla_\Theta \ell_L(\Theta)$, are shown below in Equation 10.

$$\begin{aligned}
\nabla_{\lambda_0}\ell_L &= \frac{1}{2} - \frac{1}{2n}\sum_1^n e^{-\frac{1}{2}(\lambda_0-\lambda_1 x_i^2)}|y_i - a - bx_i| \\
\nabla_{\lambda_1}\ell_L &= \frac{1}{2n}\sum_1^n x_i^2(1 - e^{-\frac{1}{2}(\lambda_0-\lambda_1 x_i^2)}|y_i - a - bx_i|) \\
\nabla_{a}\ell_L &= -\frac{1}{n}\sum_1^n sgn(y_i - a - bx_i)e^{-\frac{1}{2}(\lambda_0-\lambda_1 x_i^2)} \\
\nabla_{b}\ell_L &= -\frac{1}{n}\sum_1^n sgn(y_i - a - bx_i)x_i e^{-\frac{1}{2}(\lambda_0-\lambda_1 x_i^2)}
\end{aligned} \tag{10}$$

I found that a slightly higher learning rate of $\eta = 2.85 \times 10^{-4}$ was optimal here, which makes sense given that we expect the rate of convergence with l1 to be lower than with l2. The fit from the optimized point estimates are shown in Figure 1a-e (red lines).

Visually, the MLE estimates appear more robust to the heavy-tail than the IRLS and OLS estimates, particularly for the datasets that aren't centered on $x = 0$–where the noise is highly asymmetric. The point estimates in *estimates.csv* correspond to the Laplacian MLE estimate. To ensure that I wasn't fitting to the noise in this data, I ran 5-fold cross validation on these estimates. The training and validation loss are plotted in Figure 2. With the exception of dataset *data_1_4*, they decrease fairly monotonically and are close in absolute value, this is an indication that the $\hat{B}_L$ point estimates are not over-fitted to the noise. This is implemented in the script *k_fold_cv.py*.

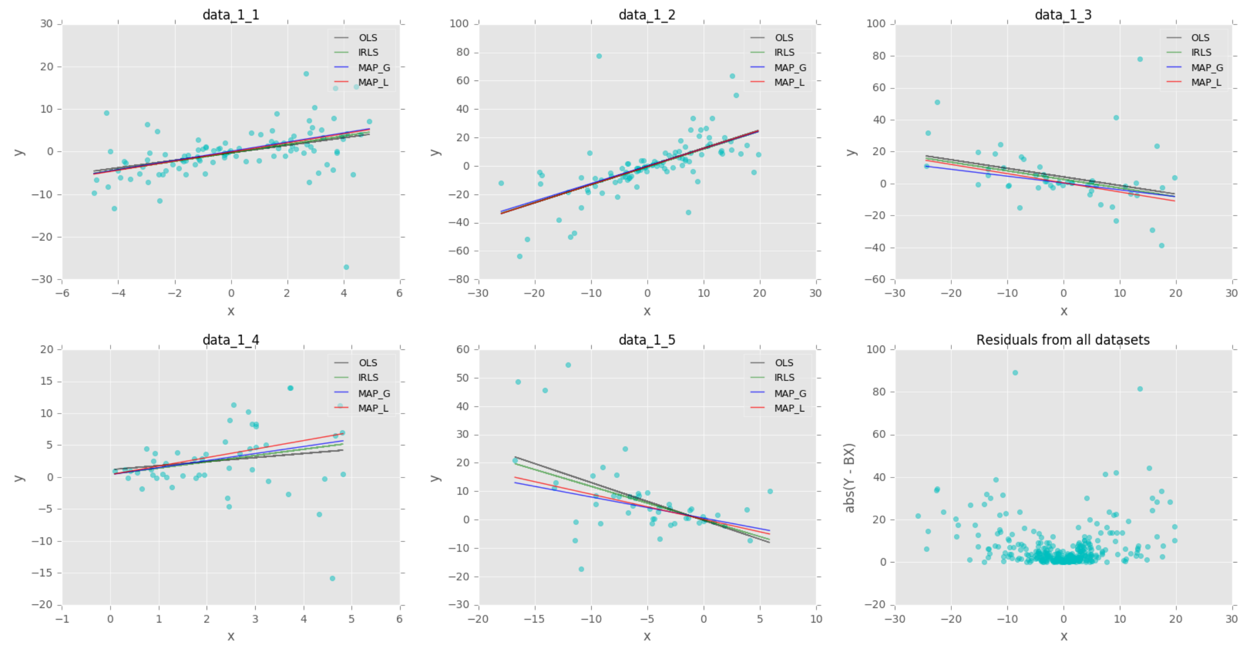

Figure 1: (a-e) Linear fits obtained using OLS (black), IRLS (green), Gaussian MLE (blue), and Laplacian MLE (red) estimates. (f) Absolute residuals obtained from the IRLS fit plotted versus x.

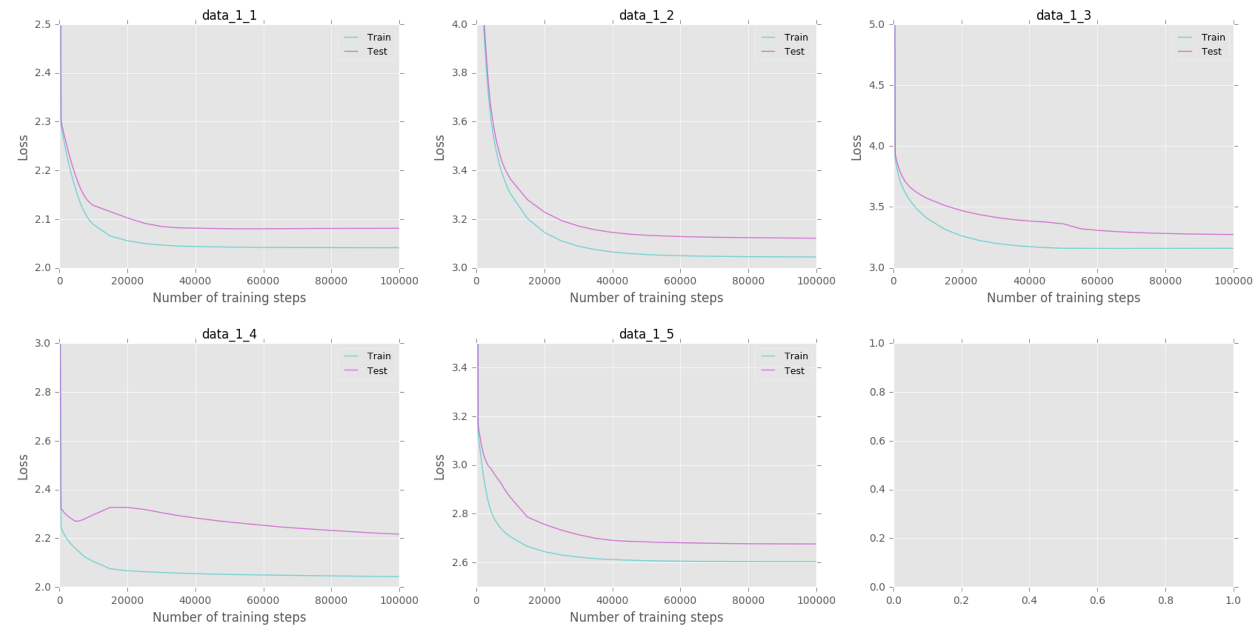

Figure 2: (a-e) Training and test loss of the Laplacian MLE estimate plotted as a function of number of training steps for each of the five datasets.