Linear regression with heavy-tailed residuals

Amin Saied

May 14, 2018

1 Introduction

We compare several methods estimating the parameters (a, b) of the linear model,

$$y_i = ax_i + b + e_i,$$

under the assumption that the residuals e_i are drawn independently from a heavy-tailed distribution with conditional mean zero given x. A natural choice for such a distribution is the t-distribution with ν degrees of freedom and a scale parameter σ^2,

$$e_i \sim t(\nu, \sigma^2),$$

where ν controls the amount of noise. Indeed, as $\nu \to \infty$ we converge to normally distributed residuals, where the ordinary least squares (OLS) estimate of the parameters (a, b) is effective. However, for small ν the OLS can perform poorly as it is sensitive to outliers. This sensitivity stems from the fact that the OLS minimizes the cost function

$$J(a, b) = \sum_{i=1}^{N} \frac{1}{2} e_i^2,$$

and thus penalizes errors quadratically.

M-estimates. Our first attempt to remedy this shortcoming is to consider a more general class of estimates known as M-estimates. These can be seen as a modified least squares estimate, in which outliers are down-weighted. Concretely, we set up an objective function,

$$J(a, b; \rho) := \sum_i \rho(e_i) \tag{1}$$

in terms of a norm $\rho : \mathbb{R} \to \mathbb{R}$ that is chosen to penalize outliers less than quadratically. For example, the Huber norm ρ assigns quadratic cost to small errors, say $|e_i| < t$, and linear cost otherwise (see Eq. A1 in Section A). By minimizing this modified cost function we compute an M-estimate, which can be seen to outperform the OLS for small ν. We spell out the details of this approach in Section A.

The constant $t \in \mathbb{R}_{>0}$ is called a *tuning parameter*. Common sense suggests that this optimal choice for tuning parameter t should depend on the nature of the noise, i.e., on the parameter(s) ν (and σ) of the t-distribution. That is to say, there is no *a priori* best choice for tuning parameter t, and indeed, we run simulations that show that ν significantly affects performance. We are therefore motivated to estimate the degrees of freedom ν (and scaling parameter σ).

EM-type method. Our data generating process is equivalent to the following mixture of Gaussians model. Let $\mu_i = ax_i + b$ and let,

$$y_i | \mu_i, \sigma, w_i \sim \mathcal{N}\left(\mu_i, \sigma^2 / w_i\right)$$

where,

$$w_i | \nu \sim \text{Gamma}\left(\frac{\nu}{2}, \frac{\nu}{2}\right).$$

As a mixture of Gaussians, its parameters $\{a, b, \sigma^2, \nu\}$ can be learned by an EM-type algorithm. Specifically, we implement a modified version of the ECME algorithm introduced by Liu-Rubin in [1] - full details of which are presented in Section B. In addition to iteratively estimating the weights w_i and the coefficients (a, b), this approach introduces iterative estimates of the degrees of freedom ν, that avoid the ambiguity inherent in the M-estimates approach.

Implementation. We use Python. The OLS and M-estimates are commonplace enough that there exist libraries computing them (we use the `statsmodels` library). We implement the ECME-algorithm ourself based on the more general methods presented in [1].

2 Simulation

We run simulations generating $N = 100$ samples from our linear model,

$$y_i = ax_i + b + e_i$$

in which the residuals $e_i \sim t(\nu, \sigma^2)$ are t-distributed. We are interested in how the performance of various estimates is related to the degrees of freedom parameter ν. For a fixed $\nu = \nu_0$ we randomly select parameters a, b, σ, generate a sample and compute our estimates, recording their deviation from the true parameters. For more details of the experiment see the script `simulation.py`.

Experiment 1. Comparing the tuning parameter t of the Huber norm $\rho(e; t)$.

We compare the OLS estimate against the M-estimates computed corresponding to the Huber norm (A1) $\rho(e; t)$ with tuning parameters $t = 1, 2, 3, 4$.

First, we note that the OLS is seen to perform poorly for small ν as expected. Second, we see that the optimal tuning parameter t varies as a function of ν. For example, for $\nu \leq 4$ we should choose $t = 1$, but for $\nu \geq 5$ we should choose $t = 4$.

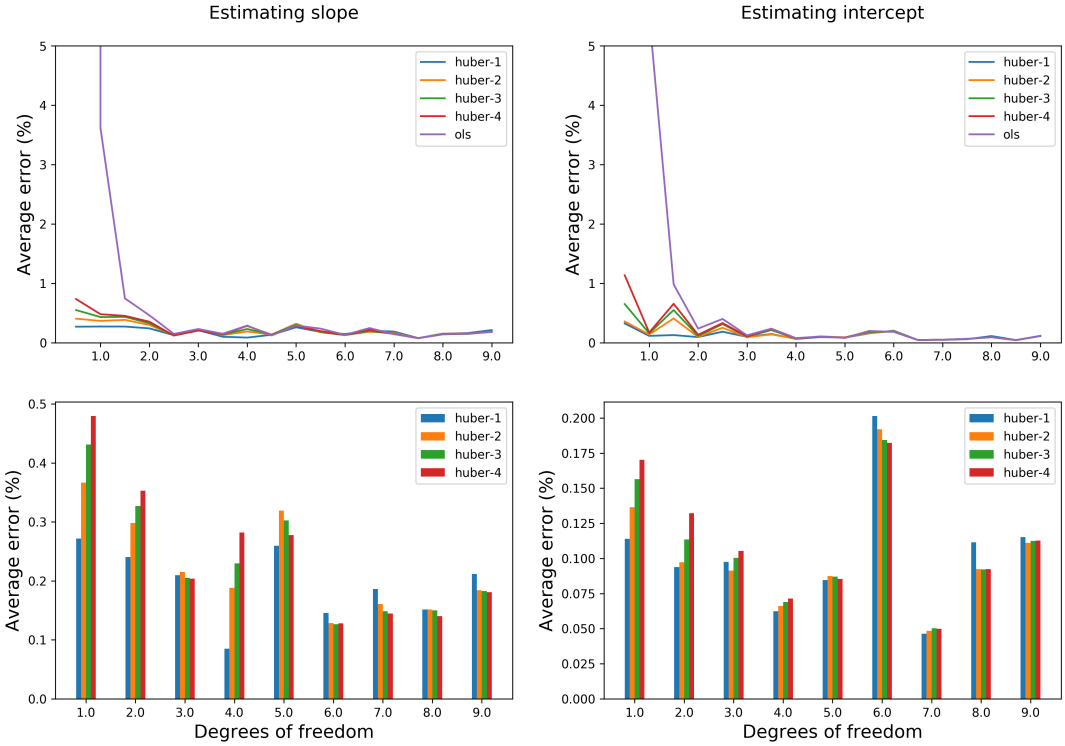


Figure 1: The left hand figures refer to the slope estimate, and the right hand figures to the intercept estimate. The first row presents an overview including the OLS estimate. The bottom row restricts attention to the Huber estimates for tuning parameters $t = 1, 2, 3, 4$.

Experiment 2. Comparing the ECME algorithm against the M-estimates.

We compare the ECME algorithm's estimates agains the M-estimates corresponding to Huber norm $\rho(e;t)$ with tuning parameter $t = 1, 4$. We see that for small ν the ECME estimate outperforms the M-estimates.

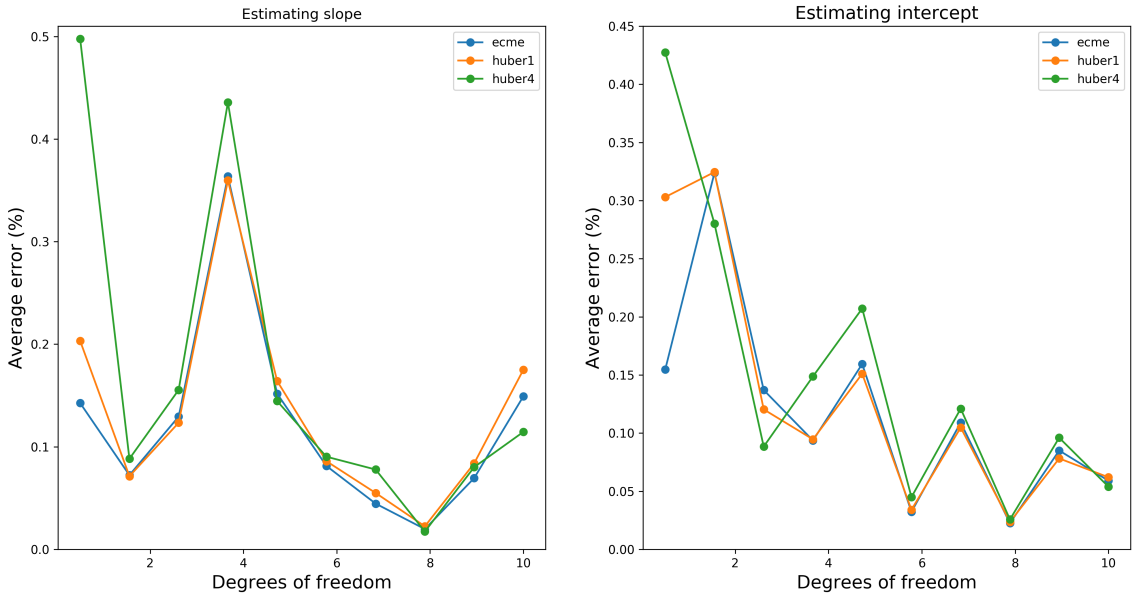
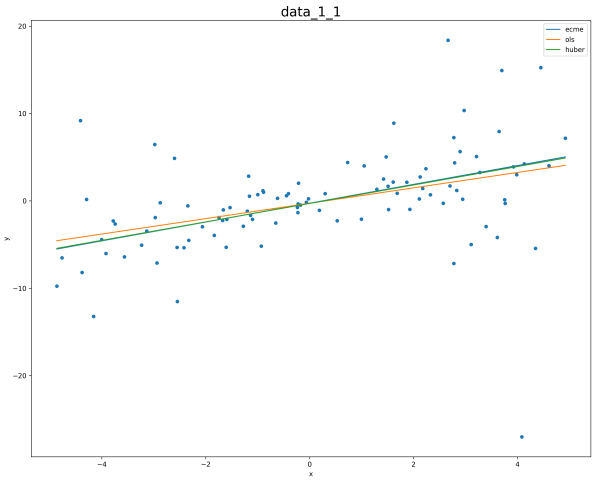


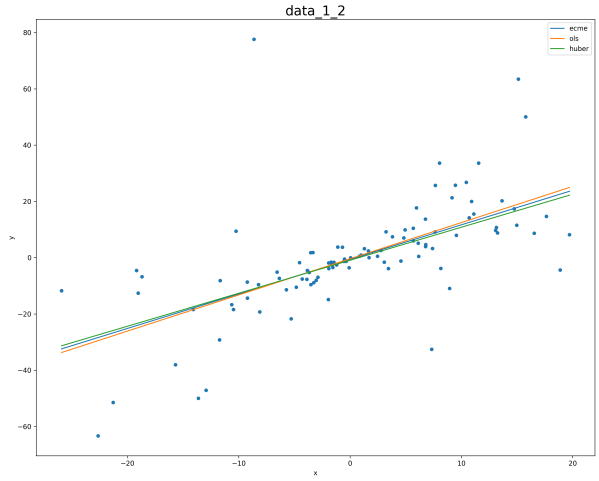
Figure 2: Comparison of estimates using the ECME algorithm, and Huber M-estimates with tuning parameter $t = 1$ and $t = 4$.

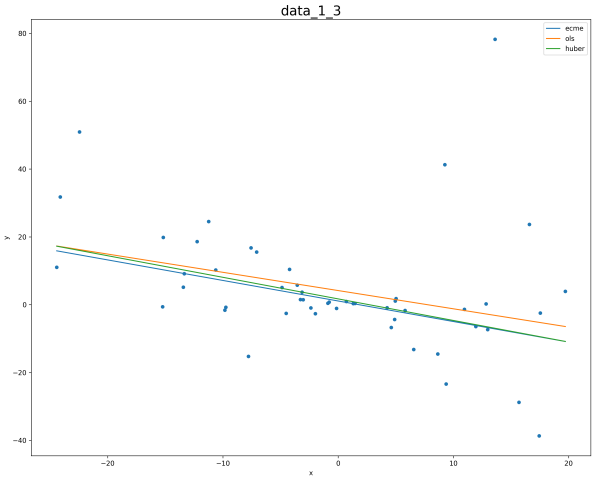
3 Results

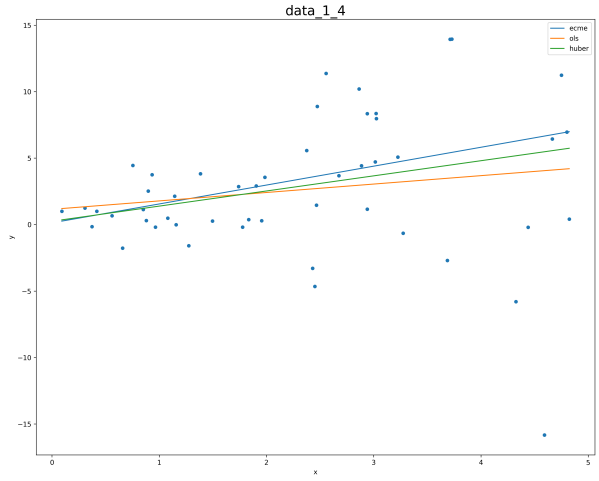
We present a brief summary of the estimates provided for each of the five datasets.

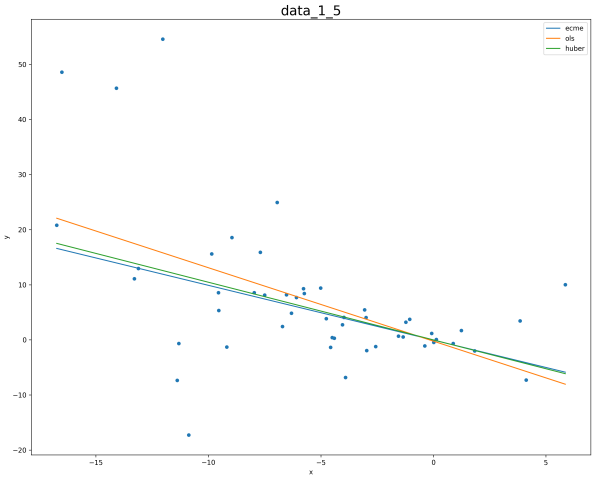
Filename	a	b
data_1_1	1.078	−0.265
data_1_2	1.227	−0.605
data_1_3	−0.605	1.101
data_1_4	1.422	0.130
data_1_5	−0.994	−0.044











Appendix

A M-estimates

Define a **cost function** J_ρ in terms of an abstract norm $\rho : \mathbb{R} \to \mathbb{R}$ as,

$$J_\rho(a, b) = \sum_{i=1}^{N} \rho(e_i),$$

For example, given the function $\rho(e) = \frac{1}{2}e^2$, minimizing the cost function $J_\rho(a, b)$ is equivalent to finding the ordinary least squares estimate. We will focus our attention on the Huber norm,

$$\rho(e; t) = \begin{cases} \frac{1}{2}e^2 & |e| < t \\ t|e| - \frac{1}{2}t^2 & \text{else} \end{cases} \tag{A1}$$

where $t \in \mathbb{R}_{>0}$ is a tuning parameter. This is a continuous function that assigns quadratic cost to errors of size at most t, and otherwise assigns linear cost. Consequently, the associated cost function J_ρ is less sensitive to outliers in the data, as desired.

The influence $\psi = \nabla\rho$ is defined as the gradient of ρ with respect to $\Theta = (a, b)$. Notice that minimizing the cost function J_ρ is equivalent to solving the system of equations,

$$\sum_{i=1}^{N} \psi(y_i - B \cdot X_i)X_i = 0.$$

where we encode x_i as a vector $X_i = (1, x_i)$, and our parameters (a, b) as a vector $B = (b, a)$. By defining the define the **weight function**,

$$w(e) = \frac{\psi(e)}{e},$$

we can rewrite this as,

$$\sum_{i=1}^{N} w_i(y_i - B \cdot X_i)X_i = 0.$$

where $w_i := w(e_i)$ denotes the weight associated to the i-th error. Notice that for the OLS estimate we have that $w(e) = 1$, and it is in this way that the more general class of M-estimates can be seen as a re-weighted least-squares estimate.

Optimization. It remains to minimize our objective function J_ρ. Let $B^{(0)}$ be an initial estimate for the parameter B. For example, let $B^{(0)}$ be the usual least squares estimate.

Repeat the following until convergence:

1. Let $e_i^{(t)}$ denote the residuals computed using the estimate $B^{(t)}$.
2. Compute the weights $w_i^{(t)}$ from the residuals $e_i^{(t)}$ via the weight function w.
3. Let $B^{(t+1)}$ be the weighted least-squares estimate,

$$B^{(t+1)} = \arg\min \sum_i \left(w_i^{(t+1)} \cdot e_i^{(t+1)} \right)^2.$$

Implementation. This algorithm is implemented in the Python library `statsmodels`.

B ECME algorithm

Recall that we have the following equivalent form for our noisy linear model.

$$y_i|\mu_i, \sigma, w_i \sim \mathcal{N}\left(\mu_i, \sigma^2/w_i\right)$$

where we write $\mu_i = ax_i + b$ and where,

$$w_i|\nu \sim \text{Gamma}\left(\frac{\nu}{2}, \frac{\nu}{2}\right).$$

Notice that here the w_i's are playing a similar role to weight function w that arise in the theory of M-estimates in that they serve to detect and down-weight noisy samples.

The advantage of this perspective however is that it allows us to employ an EM-type algorithm, which we now describe. This is a modified version of the ECME algorithm of Liu-Rubin [1].

Let $y' = (y_1, \cdots, y_N)$ and $X' = (X_1, \cdots, X_N)$ denote the concatenations of our samples into matrices, each with N columns. Suppose at iteration t you have the following (tentative) estimates $B^{(t)}, w^{(t)}, \sigma^{(t)}, \nu^{(t)}$ for the parameters of interest.

E-step Set,

$$w_i^{(t+1)} = \frac{\nu^{(t)} + 1}{\nu^{(t)} + \delta_i^{(t)}},$$

where $\delta_i^{(t)} = 1/(\sigma^{(t)})^2(y_i - \mu_i^{(t)})$ and $\mu_i^{(t)} = B^{(t)} \cdot X_i$.

CM-step-1 Set,

$$B^{(t+1)} = (X'W^{(t)}X)^{-1}X'W^{(t)}y$$

and

$$\sigma^{(t+1)} = \sqrt{\frac{\sum_{i=1}^N w_i^{(t+1)}(y_i - \mu_i^{(t+1)})}{N}}$$

CM-step-2 Find $\nu^{(t+1)}$ as the unique solution in $\mathbb{R}_{>0}$ to

$$1 - \phi(\nu/2) + \ln(\nu/2) + \frac{\sum_{i=1}^N \ln(w_i) - w_i}{N} + \frac{\sum_{i=1}^N \phi\left(\frac{\nu+1}{2}\right) - \ln\left(\frac{\nu+1}{2}\right)}{N} = 0$$

where $\phi(x)$ is the digamma function.

Remarks.

1. It is shown in [1] that this algorithm has stable monotone convergence to a local maximum likelihood estimate.
2. In CM-step-2 we use the bisection method to find the solution $\nu^{(t+1)}$.

References

[1] Liu, C., and Rubin, D. B. ML estimates of the t-distribution using EM and its extensions, ECM and ECME. *Statistica Sinica 5* (1995), 19–39.