ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D MATH
Department of Mathematics
Seminar für Statistik

Avoiding the pitfalls of S-estimators with categorical predictors

Manuel Koller

Icors 2012, University of Vermont

1 Outline

We want to compute MM-estimates on data with many factors.

2 Example: `NOxEmissions`

A typical medium sized environmental data set with hourly measurements of NOx pollution content in the ambient air.

The dataset consists of 8088 observations on the following 4 variables.

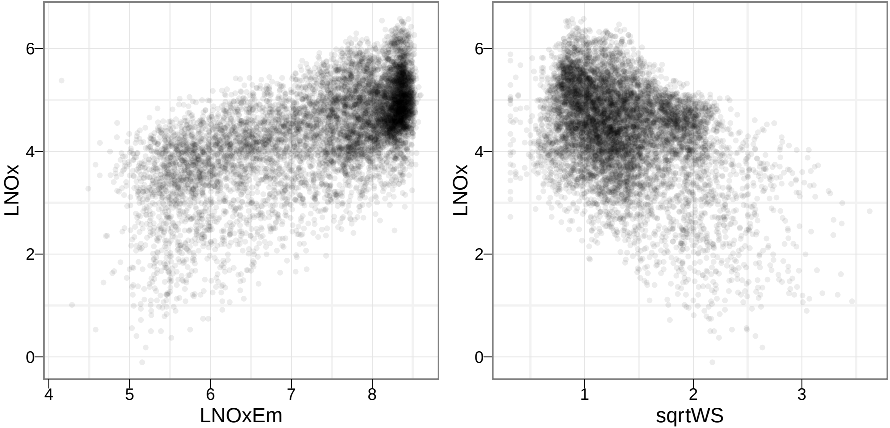
LNOx log of hourly mean of NOx concentration in ambient air [ppb] next to a highly frequented motorway.

LNOxEm log of hourly sum of NOx emission of cars on this motorway in arbitrary units.

sqrtWS Square root of wind speed [m/s].

julday day number, a factor with levels '373' ... '730', typically with 24 hourly measurements.

(The data set comes with the R package `robustbase`.)


LNOx
0
2
4
6
4
5
6
7
8
LNOxEm
LNOx
0
2
4
6
1
2
3
sqrtWS

When using (an older version) of `lmrob` on such a dataset, it usually failed to compute the initial S-estimate:

```
> lmrob(LNOx ~ ., data = NOxEmissions)

Too many singular resamples
Aborting fast_s_w_mem()

Error in lmrob.S(x, y, control = control) :
  C function R_lmrob_S() exited prematurely
```

(`lmrob` is a function in the R package `robustbase`. It computes MM-estimates.)

3 The problem

MM-estimates consist of

- an initial S-estimate with high breakdown point,
- a final M-estimate with high efficiency.

The algorithm for computing the initial S-estimate usually involves subsampling, which is problematic for such data.

S-estimates are defined as

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}\, \hat{\sigma}(r(\beta)) \,,$$

where $\hat{\sigma}()$ is an M-estimate of scale and $r()$ are the residuals.

The algorithm to compute S-estimates, simplified:

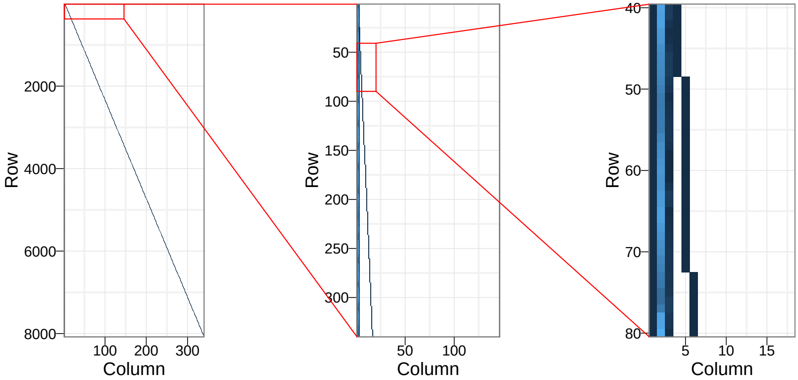
1. Take a random subsample of size p, the number of parameters; solve the problem on the subsample.

2. Find local minimum starting from the solution found in 1.

3. Repeat for a fixed number of times.

The S-estimate is then the result of 2. with the smallest scale estimate.

Zooming in the design matrix from the `NOxEmissions` example.



A valid subsample in this example must not contain any 0 only columns.

4 Strategies before robustbase version 0.9

The user had the following options:

- increase the allowed number of singular subsamples,
- use `lmRob` (R package `robust`) that uses an M/S-estimate as initial estimate for such datasets,
- switch to another estimator to solve the problem, e.g., M or L1.

In other words: wait a long time or ditch `lmrob`.

So... we improved `lmrob`!

As of version 0.9 of `robustbase`:

- Support for M/S-estimates.
- Improved algorithm to compute S-estimates to deal with such datasets as well (nonsingular subsampling).

5 Nonsingular subsampling

"Algorithm":
Build up the subsample observation by observation, checking for collinearities each time. If an observation introduces collinearities, then discard it and continue with another one.

This always works, except if:

- such a subsample does not exist, i.e., the design matrix is not of full rank, or,
- the design matrix is ill-conditioned, causing numerical problems.

And the best thing about it:
Checking for collinearities comes (almost) for free.

6 LU-factorization

The LU-decomposition of a nonsingular matrix $\boldsymbol{A}$ consists of

- a lower triangular matrix $\boldsymbol{L}$,
- an upper triangular matrix $\boldsymbol{U}$, and,
- a permutation matrix $\boldsymbol{P}$ (to avoid divisions by 0),

such that

$$\boldsymbol{PA} = \boldsymbol{LU}.$$

This is used, e.g., for solving linear systems of equations $\boldsymbol{A}\beta = b$, since

$$\beta = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}^{-1}b.$$

Basic algorithm to compute the LU-factorization: a series of Gaussian eliminations (Doolittle's algorithm).

Example: Compute LU-factorization of the (singular) matrix $\boldsymbol{A}$.

$$\boldsymbol{A}$$

$$\begin{pmatrix} 6 & 18 & 24 & 12 \\ 3 & 17 & 20 & 2 \\ 2 & 4 & 6 & 15 \\ 3 & 5 & 8 & 8 \end{pmatrix}$$

Note: The third column is the sum of the first two columns.

LU-doolittle step 1:

$$\boldsymbol{L}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 \\ -0.33 & 0 & 1 & 0 \\ -0.5 & 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{A}^{(1)} = \begin{pmatrix} 6 & 18 & 24 & 12 \\ 0 & 8 & 8 & -4 \\ 0 & -2 & -2 & 11 \\ 0 & -4 & -4 & 2 \end{pmatrix}$$

LU-doolittle step 2:

$$\boldsymbol{L}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.25 & 1 & 0 \\ 0 & 0.5 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{A}^{(2)} = \begin{pmatrix} 6 & 18 & 24 & 12 \\ 0 & 8 & 8 & -4 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-doolittle step 3:

$$\boldsymbol{L}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & NaN & 1 \end{pmatrix} \qquad \boldsymbol{A}^{(3)} = \begin{pmatrix} 6 & 18 & 24 & 12 \\ 0 & 8 & 8 & -4 \\ 0 & 0 & 0 & 10 \\ NaN & NaN & NaN & NaN \end{pmatrix}$$

Better: Gaxpy variant of LU-factorization algorithm.

This variant of the algorithm does the operations in a different order. It only calculates the elements of $\boldsymbol{U}$ when they are actually needed.

LU-gaxpy step 1:

$$\boldsymbol{L}^{(1)} \qquad \boldsymbol{U}^{(1)}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.33 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-gaxpy step 2:

$$\boldsymbol{L}^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.33 & -0.25 & 1 & 0 \\ 0.5 & -0.5 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{U}^{(2)} = \begin{pmatrix} 6 & 18 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-gaxpy step 3:

$$\boldsymbol{L}^{(3)} \qquad \boldsymbol{U}^{(3)}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.33 & -0.25 & 1 & 0 \\ 0.5 & -0.5 & NaN & 1 \end{pmatrix} \quad \begin{pmatrix} 6 & 18 & 24 & 0 \\ 0 & 8 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-gaxpy step 2:

$$\boldsymbol{L}^{(2)} \qquad \boldsymbol{U}^{(2)}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.33 & -0.25 & 1 & 0 \\ 0.5 & -0.5 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 6 & 18 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Two key facts about LU-gaxpy:

- Collinearities are detected immediately.
- In the i-th step, the algorithm only touches columns 1 to i.

Now consider the transposed design matrix:

$$\text{columns} = \text{observations},$$
$$\text{rows} = \text{predictor variables}.$$

Applying the LU-gaxpy, we only need to repeat one step if an observation introduces collinearity. All computations from the previous steps are still valid.

The LU-factorization is needed anyway, so the only extra work comes from repeating steps in case of collinearities.

7 Where's the random part?

Permute the observations in the design matrix.

8 The nonsingular subsampling algorithm

1. Permute the observations in the design matrix randomly.
2. Run LU-gaxpy step by step on the transposed design matrix, discarding any observation that introduces collinearities.
3. Use computed LU-factorization to solve the least-squares problem.

9 Avoiding numerical problems

Sometimes, for ill-conditioned design matrices, the nonsingular subsampling algorithm falsely declares a subsample nonsingular.

Preconditioning the design matrix helps to avoid this problem. `lmrob` uses a technique called matrix equilibration on the whole design matrix.

Instead of

$$\boldsymbol{A}\beta = y,$$

we solve

$$(\boldsymbol{D}_{\text{row}}\boldsymbol{A}\boldsymbol{D}_{\text{col}})\bar{\beta} = \boldsymbol{D}_{\text{row}}y, \quad \beta = \boldsymbol{D}_{\text{col}}\bar{\beta}.$$

The diagonal matrices $\boldsymbol{D}_{\text{col}}$ and $\boldsymbol{D}_{\text{col}}$ need to be computed only once for the whole design matrix.

10 M/S-estimates (Maronna & Yohai, 2000)

Split the design matrix in a categorical part and a continuous part.

- For the categorical part, use an M-estimate (usually L1), while
- for the continuous part, use an S-estimate.

This avoids computational difficulties on the categorical part while keeping the better robustness for the continuous part.

In formulas:

$$y_i = x_{1i}^{\mathsf{T}}\beta_1 + x_{2i}^{\mathsf{T}}\beta_2 + \varepsilon_i.$$

Then use an S-estimate for the continuous part β_2:

$$\hat{\beta}_2 = \operatorname*{argmin}_{\beta_2} \hat{\sigma}\left(r\left(\beta_1^*\left(\beta_2\right), \beta_2\right)\right),$$

and an M-estimate for the categorical part β_1:

$$\beta_1^*\left(\beta_2\right) = \operatorname*{argmin}_{\beta_1} \sum_{i=1}^{n} \rho\left(y_i - x_{1i}^{\mathsf{T}}\beta_1 - x_{2i}^{\mathsf{T}}\beta_2\right).$$

What about interactions of categorical and continuous variables?

11 Comparison

How long does it take to fit a multiple linear regression model?

$$\texttt{LNOx} \sim \texttt{1 + LNOxEm + sqrtWS + julday}$$

Estimator	Running time [s]
Least Squares (`lm`)	1.213
L1 (`lmrob.lar`)	2.680
S, simple subsampling (max tries = ∞)	$>$ 2592000
M/S	919.513
S, nonsingular subsampling	1082.109

Design matrix: $n = 8088$, $p = 340$.

Conclusions

- `lmrob` of the R-package `robustbase` is now suitable also for datasets with many factors, even when some levels have low frequency.
- Nonsingular subsampling allows us to use the regular S-estimate. This does not require extra work for easy problems.

References

M. Koller (to appear). Nonsingular subsampling for S-estimators with categorical predictors.

M. Salibian-Barrera and V. J. Yohai (2006). A fast algorithm for S-regression estimates. *Journal of Computational and Graphical Statistics*, 15(2), 414–427.

R. A. Maronna and V. J. Yohai (2000). Robust regression with both continuous and categorical predictors. *Journal of Statistical Planning and Inference*, 89, 197–214.

Data: $p \times p$ matrix $\boldsymbol{A}$.
Result: Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.

$\boldsymbol{A}^{(0)} \leftarrow \boldsymbol{A}$
for $j \leftarrow 1$ **to** p **do**

$$\boldsymbol{L}_j \leftarrow \begin{pmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & -\frac{a_{j+1,j}^{(j-1)}}{a_{j,j}^{(j-1)}} & \ddots & \\ & & \vdots & & \ddots \\ 0 & & -\frac{a_{p,j}^{(j-1)}}{a_{j,j}^{(j-1)}} & & 1 \end{pmatrix}$$

$\boldsymbol{A}^{(j)} \leftarrow \boldsymbol{L}_j \boldsymbol{A}^{(j-1)}$

$\boldsymbol{U} \leftarrow \boldsymbol{A}^{(p)}$
$\boldsymbol{L} \leftarrow \boldsymbol{I} + \sum_{j=1}^{p} \left(\boldsymbol{I} - \boldsymbol{L}_j\right)$

Algorithm 1: LU-doolittle (without pivoting).

Data: $n \times p$ matrix $\boldsymbol{X}$, response vector y, singularity treshold ε.

Result: Return code (0 for success, otherwise failing step), initial estimate $\hat{\beta}$.

$\boldsymbol{U} \leftarrow \mathbf{0}$; $\boldsymbol{L} \leftarrow \boldsymbol{I}$; $s \leftarrow 1:p$; $k \leftarrow 1$

$t \leftarrow \texttt{perm}(1:n)$; $\boldsymbol{A} \leftarrow \boldsymbol{X}^{\mathsf{T}}_{t,1:p}$; $y \leftarrow y_t$

for j **in** 1 **to** p **do**

    if $j == 1$ **then** $v_{1:p} \leftarrow \boldsymbol{A}_{1:p,k}$

    else

        $\boldsymbol{U}_{1:j-1,j} \leftarrow \boldsymbol{L}^{-1}_{1:j-1,1:j-1}\boldsymbol{A}_{1:j-1,k}$

        $v_{j:p} \leftarrow \boldsymbol{A}_{j:p,k} - \boldsymbol{L}_{j:p,1:j-1}\boldsymbol{U}_{1:j-1,j}$

    if $j < p$ **then**

        if $|v_j| \geq \varepsilon$ **then**

            $s_j \leftarrow k$

            $\boldsymbol{L}_{j+1:p,j} \leftarrow v_{j+1:p}/v_j$

    ...

...

...

for j **in** 1 **to** p **do**

    ...

    if $|v_j| < \varepsilon$ **then**

        if $k < n$ **then**

            $k \leftarrow k + 1$

            Goto *4*

        else

            return j

    $\boldsymbol{U}_{j,j} \leftarrow v_j$

    $k \leftarrow k + 1$

$\hat{\beta} \leftarrow \boldsymbol{L}^{-\mathsf{T}} \boldsymbol{U}^{-\mathsf{T}} y_s$

return $0, \hat{\beta}$

Algorithm 2: Nonsingular subsampling using modified LU-gaxpy (without pivoting). $1 : p - 1 = (1, 2, \dots, p - 1)$.