Huber Regression and Huber Lasso Regression

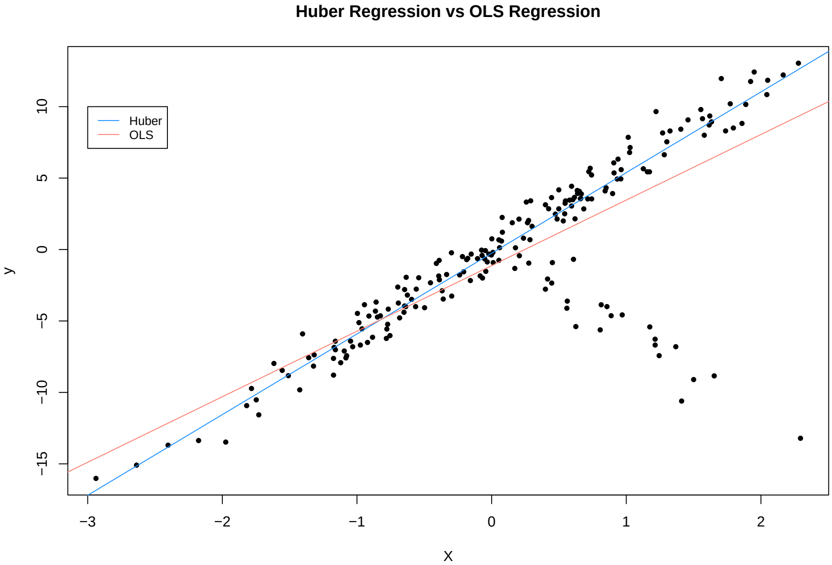
John Kenney

2023-11-25

Huber Regression

- Huber Regression is a useful regression model to use in the case of outliers in the data being studied.
- The Huber loss function is a middle ground between MSE(Mean Squared Error) and MAE(Mean Absolute Error) by being more forgiving to outliers.
- Motivation is develop algorithms to efficiently implement Huber Regression and Huber Lasso Regression.

Simple Huber Regression Example



R package functions

Main functions

- cd_huber
- cd_huber_lasso
- plot_cd_huber_Lasso
- cv.lasso.huber
- data.generate

Rcpp helper functions

- cdhubercpp
- resids
- lossc
- weightc

Huber Regression

Objective function:
$L(\beta) = \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta)$
Optimize the objective function such that:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta)$$

where ϕ is the Huber loss function with a threshold of M > 0.

$$\phi(e) = \begin{cases} e^2, & \text{if } |e| \leq M, \\ 2M|e| - M^2, & \text{if } |e| > M, \end{cases}$$

For small residuals the Huber regression is equal to the Least Squares, but for large residuals the penalty is linear and lower.

Huber Algorithm via Coordinate Descent

- Initialize $\beta^{(0)}$ coefficients, $W^{(0)} = \boldsymbol{I}_{nn}$, $M = IQR(y)/10$.
- Repeat the following steps until convergence $|\phi^{(t)} - \phi^{(t-1)}| < \text{tol}$: For $t = 1, \ldots, \text{maxiter}$,
  1. Compute the residuals e.
  2. For each weight $i = 1, \ldots, n$, Update the Weight matrix W by
  $$W_{ii}^{(t+1)} = \begin{cases} 1, & \text{if } |e_i| \le M, \\ M/|e_i|, & \text{if } |e_i| > M, \end{cases}$$
  3. For each predictor $j = 1, \ldots, p$, Update the coefficient β_j by
  $$\beta_j^{(t+1)} = \frac{n^{-1} \sum_{i=1}^{n} r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij}}{n^{-1} \sum_{i=1}^{n} w_{ii}^{(t+1)} x_{ij}^2}$$
  where $r_{ij}^{(t)} = y_i - \beta_0^{(t+1)} - \beta_1^{(t+1)} x_{i1} - \ldots - \beta_{j-1}^{(t+1)} x_{i(j-1)} - \beta_{j+1}^{(t)} x_{i(j+1)} \ldots - \beta_p^{(t)} x_{ip}$.
  4. Update the new residuals.
  5. For each residual $i = 1, \ldots, n$, Update the Loss $\phi^{(t)}$.

Huber Lasso Regression

Objective function:
$L(\beta) = \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta) + \lambda \sum_{j=1}^{p} |\beta_j|$
Optimize the objective function such that:

$$\hat{\beta}_{\text{Lasso}} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta) + \lambda \sum_{j=1}^{p} |\beta_j|$$

where ϕ is the Huber loss function with a threshold of M > 0 and $\lambda > 0$.

$$\phi(e) = \begin{cases} e^2, & \text{if } |e| \leq M, \\ 2M|e| - M^2, & \text{if } |e| > M, \end{cases}$$

For small residuals the Huber regression is equal to the Least Squares, but for large residuals the penalty is linear and lower.

Huber Lasso Algorithm via Coordinate Descent

- Initialize $\beta^{(0)}$ coefficients, $W^{(0)} = \boldsymbol{I}_{nn}$, M = IQR(y)/10.
- Repeat the following steps until convergence $|\phi^{(t)} - \phi^{(t-1)}| <$ tol: For $t = 1, \ldots,$ maxiter,
  1. Compute the residuals e.
  2. For each weight $i = 1, \ldots, n$, Update the Weight matrix W by

$$W_{ii}^{(t+1)} = \begin{cases} 1, & \text{if } |e_i| \leq M, \\ M/|e_i|, & \text{if } |e_i| > M, \end{cases}$$

  3. For each predictor $j = 1, \ldots, p$, Update the coefficient β_j by

$$\beta_j^{(t+1)} = \begin{cases} \frac{n^{-1}\sum_{i=1}^n r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij} - \lambda}{n^{-1}\sum_{i=1}^n w_{ii}^{(t+1)} x_{ij}^2}, & \text{if } n^{-1}\sum_{i=1}^n r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij} > \lambda, \\ \\ \frac{n^{-1}\sum_{i=1}^n r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij} + \lambda}{n^{-1}\sum_{i=1}^n w_{ii}^{(t+1)} x_{ij}^2}, & \text{if } n^{-1}\sum_{i=1}^n r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij} < -\lambda, \\ \\ 0, & \text{else}, \end{cases}$$

     where $r_{ij}^{(t)} = y_i - \beta_0^{(t+1)} - \beta_1^{(t+1)} x_{i1} - \ldots - \beta_{j-1}^{(t+1)} x_{i(j-1)} - \beta_{j+1}^{(t)} x_{i(j+1)} \ldots - \beta_p^{(t)} x_{ip}$.
  4. Update the new residuals.
  5. For each residual $i = 1, \ldots, n$, Update the Loss $\phi^{(t)}$.

Simulation Data

For a simple simulated data example for the Huber Regression and Huber Lasso regression I used the following settings:

p = 300, n = 200, $\rho = 0.5$, lambda.list $= \sqrt{\frac{\log p}{n}} * (e^{-10/5}, e^{-9/5}, \ldots, e^{10/5})$

$$\boldsymbol{\beta} = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 0, \ldots, 0\}$$

$$\boldsymbol{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}_p(\rho)), \text{ with } \Sigma_{i,j}(\rho) = \rho^{|i-j|} \text{ for any } 1 \leq i, j \leq p$$

$$\boldsymbol{o} \sim 2 \times Bin(n, \text{prob } = 1 - 0.1) - 1, \text{ if } \boldsymbol{o}_i = -1 \text{ then } \boldsymbol{o}_i = -5, \ \boldsymbol{o}_i \in \{-5, 1\}$$

$$\boldsymbol{\epsilon} \sim N(0, 1)$$

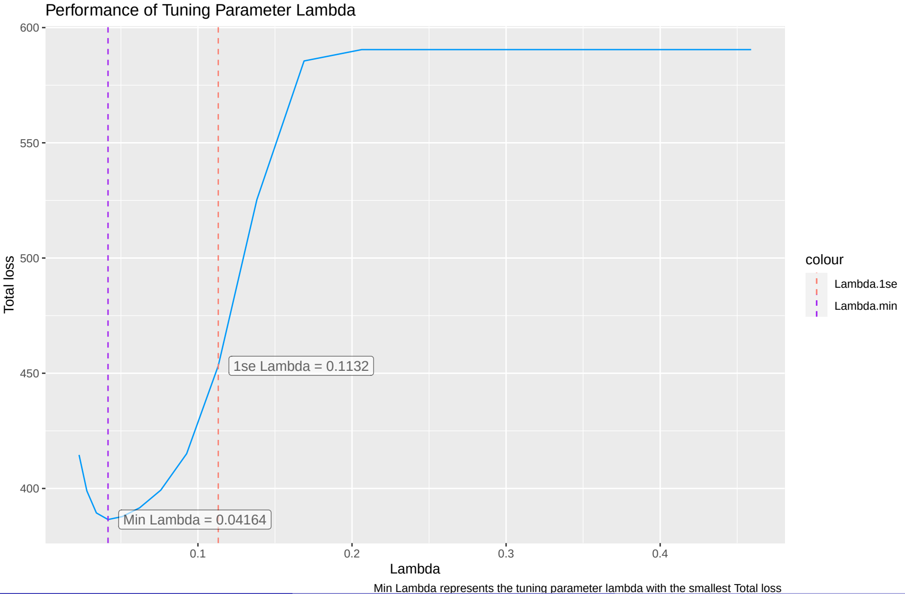
$$\boldsymbol{y} = \boldsymbol{o} \times (\mathbf{1}_n, \boldsymbol{X})\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Beta Comparison with Hqreg and CVXR with prob of 0.1 of flipping y * 5 for outliers

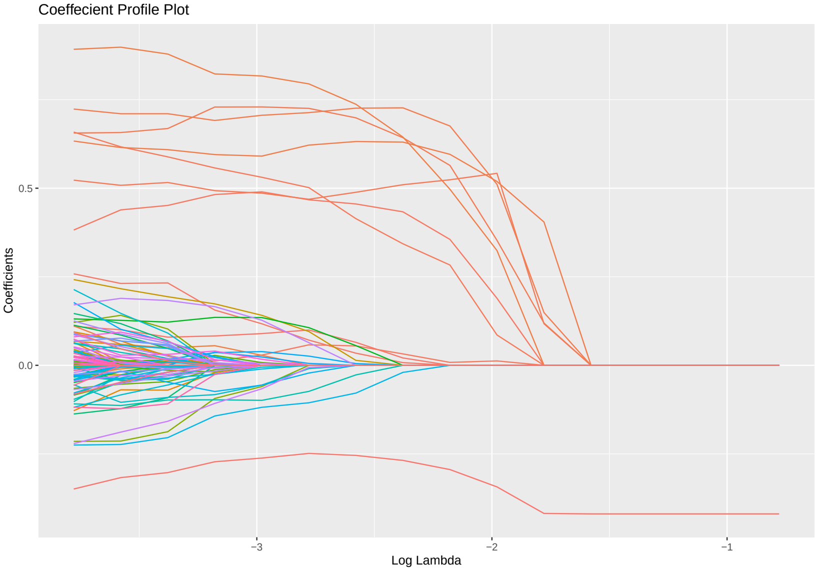
Table 1: First 20 Beta Estimations for different methods

	True	OLS	My Huber	CVXR Huber	Hqreg Huber	My CV.Hub.Lass.B.Min	My CV.Hub.Lass.B.1SE
β_0	0.0	-11.6640934	0.4858236	0.5576697	0.3240740	-0.2659899	-0.3203582
β_1	0.1	0.8379829	0.3884892	-0.2159526	0.5560011	0.0889684	0.0000000
β_2	0.2	1.6297845	-0.3504293	-0.0929982	-0.3673730	0.1265190	0.0000000
β_3	0.3	-20.9693144	0.2991818	1.0409126	0.5028312	0.5023785	0.2703319
β_4	0.4	15.7131122	-0.4656678	-1.1093972	-0.4260353	0.0188692	0.0251203
β_5	0.5	-1.4205602	1.4483560	0.8988197	1.5025241	0.5420627	0.1649969
β_6	0.6	-6.5964260	0.0163151	0.3588239	0.2155487	0.4807850	0.5461583
β_7	0.7	-11.7726897	0.1148258	0.0168000	0.1194449	0.7447108	0.4657214
β_8	0.8	7.6317047	-0.5936663	-0.4728397	-0.5847637	0.6951486	0.5977820
β_9	0.9	7.4623911	1.0031848	0.7442152	1.0202888	0.5798182	0.5265666
β_{10}	1.0	-0.4303810	-0.5237145	-0.7601152	-0.4769229	0.8201683	0.3771278
β_{11}	0.0	5.9153259	0.1766881	0.1100426	0.2222920	0.0364179	0.0000000
β_{12}	0.0	11.3836589	0.9589023	0.8736083	0.9460358	0.0000000	0.0000000
β_{13}	0.0	-1.6554911	0.4723407	0.3888498	0.5162937	0.0000000	0.0000000
β_{14}	0.0	-8.8653321	-0.7531445	-0.3739318	-0.7689088	0.0000000	0.0000000
β_{15}	0.0	13.7362892	0.1537146	0.0166114	0.1748268	0.0000000	0.0000000
β_{16}	0.0	-1.8037999	0.6461941	0.7888944	0.6240673	0.0000000	0.0000000
β_{17}	0.0	-2.4002965	-0.1057129	0.0508151	-0.0966411	0.0000000	0.0000000
β_{18}	0.0	18.7229551	-0.1563897	-0.4603921	-0.1961418	0.0000000	0.0000000
β_{19}	0.0	-18.4625350	0.6531029	0.6083739	0.6608917	0.0000000	0.0000000

Performance of Tuning Parameter Lambda in Cross Validation Huber Lasso on Simulated Data



Beta Coefficients plot



Huber Regression Computation Time Comparison

Table 2: Huber Simulation Time Comparison (Milliseconds)

expr	mean	median	neval
n = 80 , p = 20			
my hub	5.199350	5.0653500	100
cvxr	147.533855	142.4225515	100
hqreg	0.318049	0.3310010	100
n = 80 , p = 150			
my hub	256.318883	256.4429510	100
cvxr	143.850938	143.2065010	100
hqreg	3.100203	3.0345510	100
n = 300 , p = 20			
my hub	74.659322	74.2417010	100
cvxr	148.888531	147.8995515	100
hqreg	0.582633	0.5751515	100
n = 300 , p = 150			
my hub	4436.828423	4426.9337000	100
cvxr	174.753524	171.1159010	100
hqreg	3.785990	3.5945515	100

Simulation Evaluation Results

Table 3: Huber Lasso Regression Cross Validation Simulation

	p = 20	p = 40	p = 50
Average Time in Milliseconds			
n = 50	119.5195675	533.3150148	685.9400630
n = 100	241.8092728	726.1304617	1263.5112286
n = 200	770.3215480	1797.7575898	2524.4353771
Average L2 Error of Beta Min			
n = 50	1.4061758	1.8346273	1.5294536
n = 100	0.9417172	0.9495458	1.0467645
n = 200	0.5412399	0.6435196	0.7250445
SE L2 Error of Beta Min			
n = 50	0.2090930	0.2063513	0.1962621
n = 100	0.1161284	0.0765464	0.1288632
n = 200	0.0524856	0.0593337	0.0749206
Average L2 Error of Beta 1SE			
n = 50	1.5814216	2.0357213	1.8123745
n = 100	1.3530198	1.2784921	1.3083043
n = 200	1.0130262	1.0937808	1.0554567
SE L2 Error of Beta 1SE			
n = 50	0.1520949	0.1642038	0.1593463
n = 100	0.0858899	0.0733095	0.1115142
n = 200	0.0579941	0.0514703	0.0490095

Real Data: Boston Housing data scaled with $\mu_j = 0$ and $\sigma_j = 1$ for $j = 1, \ldots, 13$

Response variable:

- medv: median value of owner-occupied homes in \$1000s.

Explanatory variables scaled with $\mu_j = 0$ and $\sigma_j = 1$ for $j = 1, \ldots, 12$:

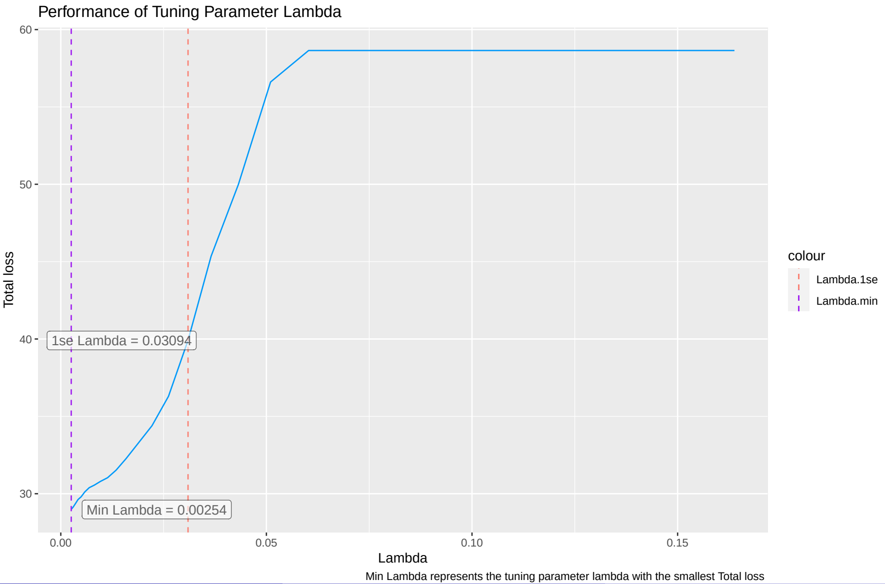
- crim: per capita crime rate by town.
- zn: proportion of residential land zoned for lots over 25,000 sq.ft.
- indus: proportion of non-retail business acres per town.
- chas: Charles River dummy variable ($= 1$ if tract bounds river; 0 otherwise).
- nox: nitrogen oxides concentration (parts per 10 million).
- rm: average number of rooms per dwelling.
- age: proportion of owner-occupied units built prior to 1940.
- dis: weighted mean of distances to five Boston employment centres.
- rad: index of accessibility to radial highways.
- tax: full-value property-tax rate per \$10,000.
- ptratio: pupil-teacher ratio by town.
- lstat: lower status of the population (percent).

Boston Housing Beta Coefficients

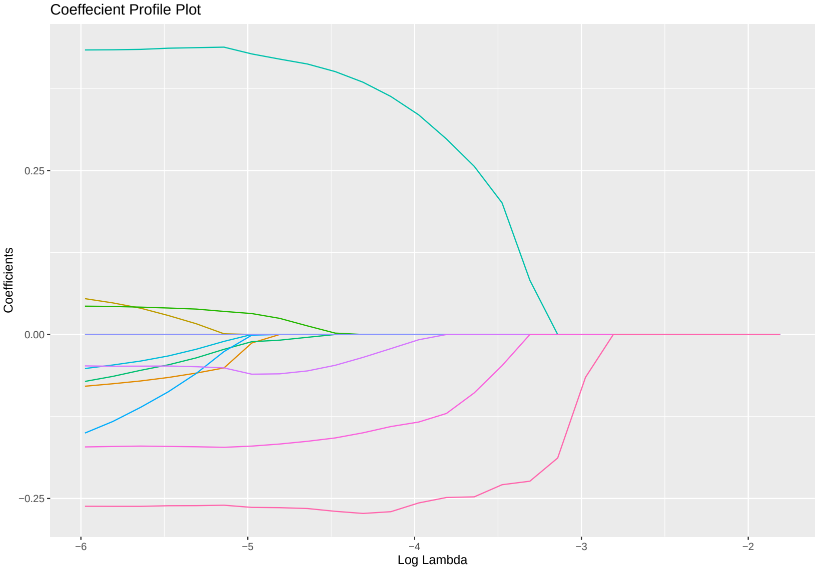
Table 4: Boston Housing Beta Coefficient Estimation Comparisons

	OLS	My Huber	CVXR Huber	Hqreg Huber	My CV.Hub.Lass.B.Min	My CV.Hub.Lass.B.1SE
β_0	0.0000000	-0.0899253	-0.0900113	-0.0896988	0.0000000	0.0000000
β_1	-0.1135281	-0.1166324	-0.1167406	-0.1162627	-0.0762960	0.0000000
β_2	0.1190922	0.0937564	0.0933594	0.0946902	0.0467552	0.0000000
β_3	0.0100459	0.0050221	0.0054821	0.0040637	0.0000000	0.0000000
β_4	0.0784314	0.0476145	0.0476592	0.0474335	0.0422311	0.0000000
β_5	-0.2363392	-0.1440796	-0.1446304	-0.1412894	-0.0644833	0.0000000
β_6	0.2794637	0.3698876	0.3707354	0.3676571	0.4435595	0.1897909
β_7	0.0110510	-0.0561053	-0.0566206	-0.0542277	-0.0601475	0.0000000
β_8	-0.3413134	-0.2307421	-0.2303214	-0.2306147	-0.1435638	0.0000000
β_9	0.2739906	0.1598940	0.1600303	0.1593809	0.0000000	0.0000000
β_{10}	-0.2323976	-0.2011892	-0.2011228	-0.2019339	-0.0496788	0.0000000
β_{11}	-0.2206899	-0.1729578	-0.1730456	-0.1725510	-0.1716782	-0.0410652
β_{12}	-0.4286134	-0.2785586	-0.2775446	-0.2817749	-0.2526991	-0.2269995

Performance of Tuning Parameter Lambda in Cross Validation Huber Lasso for Boston Housing



Boston Housing Beta Coefficients plot



Future Work

- I would like to implement my cross validation into Rcpp.
- I would also like to parallelize the my loops in my cross validation and simulation evaluations for different parameters to speed up computation time using the CPU and GPU.
- Going forward I want to implement different penalties to the Huber Loss function such as Ridge penalty and the Elastic Net Penalty.
- Lastly, I wish to apply different loss function besides Huber and MSE to further compare the different regression methods.
- Simulated comparisons of L2 error Beta coefficients, standard error, and time of known packages such as Hqreg with my package.

Conclusion

- The Huber Regression functions are excellent tools when you are working with data that contains outliers.
- Also with the use of computation of β_j using coordinate descent we are able to solve for the β_j solutions without computing any large matrix inverses.
- My Huber Regression estimated $\hat{\beta}$ coincides with existing packages such as CVXR and Hqreg besides some small inconsistencies.
- One issue is ensuring convergence of the Huber loss function for data that is not either scaled or centered and scaled, so a good solution is to scale your data before performing Huber Regression.
- Thank you!