HuberReg: Huber and Huber Lasso Regression via Coordinate Descent

John Kenney
University of Texas at Dallas
jfk150030@utdallas.edu

Abstract

This paper will introduce a coordinate descent algorithm to computing Huber Regression and Huber Regression with a L1 penalty term in the R package HuberReg. We will also discuss the efficiency of the algorithm and results on simulated data and on the Boston Housing Data set. For evaluating our framework and implementation we will compare our implementation of Huber Regression with current R packages CVXR Fu et al. (2020) and Hqreg Yi and Huang (2017).

1 Introduction

The presence of outliers in data can lead to misleading results and inferences in the data. In the case of linear models there have been many methods constructed to combat the influence of outliers in the model. One such method to dampen the influence of outliers on the parameter estimates is the use of the Huber Loss function Huber (1964).

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

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

The effect of this loss function is that in the case of large residuals the penalty applied to that observation increases linearly, but for small residuals the Huber function is identical to least squares regression. The main idea of the Huber loss function is to find the best middle ground between the squared loss function used in least squares regression and the absolute loss function. Another advantage is that the Huber loss function is strongly convex which means that there is a unique global minimum. In this paper we will solve the parameter estimates using a coordinate descent algorithm. The main idea of this method is to iteratively solve for parameter estimates, where for each iteration iteratively update each estimate fixing all others to its current value, until some convergence criteria is satisfied. One disadvantage of the implementation of my Huber regression functions found in the R package HuberReg is that the computation of the parameter estimates does not scale well with number of observations as will be shown in the simulation section of this paper.

2 Implementation

2.1 Huber Regression Implementation

To implement our method for Huber Regression we will optimize the the objective function:

$$L(\beta) = \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta)$$

such that

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

The algorithm for the calculation of the beta coefficients for the function cd_huber found in our R package HuberReg is defined below:

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

$$\beta_j^{(t+1)} = \frac{n^{-1} \sum_{i=1}^{n} r_{ij}^{(t)} w_{ii}^{(t+1)} x_{ij}}{n^{-1} \sum_{i=1}^{n} w_{ii}^{(t+1)} x_{ij}^2}$$

   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 $\phi^{(t)}$.

In our R package this code is written using Rcpp and RcppArmadillo to improve efficiency and time of computation.

2.2 Huber Lasso Regression Implementation

The other implementation in our R package is the Huber Lasso Regression. The implementation of this method is similar to the Huber regression algorithm defined above but has some small changes in the updating of the beta coefficients step. The objective function we wish to optimize to find the global minimum is:

$$L(\beta) = \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta) + \lambda \sum_{j=1}^{p} |\beta_j|; \lambda > 0$$

such that

$$\hat{\beta}_{\text{Lasso}} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \phi(y_i - x_i^\top \beta) + \lambda \sum_{j=1}^{p} |\beta_j|; \lambda > 0$$

The algorithm for how Huber Lasso regression is computed in the function cd_huber_lasso in our R package HuberReg is defined below using a soft threshold:

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 $\phi^{(t)}$.

In our R package this code is similarly calculated using Rcpp and RcppArmadillo to speed up time of computations.

3 Simulation

The Code for the Tables and Figures can be found in Section 6 denoted as Code. To show the accuracy of my results and efficiency of my algorithms we will work with the following data generating process:

$$\begin{aligned} &\boldsymbol{\beta} = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 0, \ldots, 0\} \\ &\boldsymbol{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}_p(\rho)), \text{ with } \Sigma_{i,j}(\rho) = \rho^{|i-j|} \text{ for any } 1 \leq i, j \leq p \\ &\boldsymbol{o} \sim 2 \times Bin(n, \text{prob } = 1 - 0.1) - 1, \text{ if } \boldsymbol{o}_i = -1 \text{ then } \boldsymbol{o}_i = -5, \boldsymbol{o}_i \in \{-5, 1\} \\ &\boldsymbol{\epsilon} \sim N(0, 1) \\ &\boldsymbol{y} = \boldsymbol{o} \times (\boldsymbol{1}_n, \boldsymbol{X})\boldsymbol{\beta} + \boldsymbol{\epsilon} \end{aligned}$$

This data generation process can be repeated by using the data.generate function in the HuberReg R package.

For the simulation results we will be comparing the estimation and time efficiency of the HuberReg R package Huber Regression to two existing R packages CVXR Fu et al. (2020) and Hqreg Yi and Huang (2017).

For the first simulation setting shown in Table 1 we set $n = 500$, $p = 20$,$\rho = 0.5$, lambda.list = $\sqrt{\frac{\log p}{n}} * (e^{-10/5}, e^{-9/5}, \ldots, e^{10/5})$.

We can see from the Table 1 that the estimation of the cd_huber function from the R package HuberReg denoted in the table as My Huber, gives similar estimates of the beta coefficients. One thing to note here is the strength of the Huber Regression in comparison with the Ordinary Least Squares (OLS) estimates, since we perturb our response variable, we introduce many outliers into our data. We can see from the coefficient estimates while the My Huber (cd_huber), CVXR Huber, and

Hreg Huber are close to the true beta values the OLS estimates are not close to the true beta values. It can also be seen from the table that results of the cv.lasso.huber function from the HuberReg R package that the My CV.Hub.Lasso.B.Min (beta min based on the lambda with the minimum Huber loss found via cross validation) does not do the best variable selection the My CV.Hub.Lasso.B.1SE (beta based on the largest lambda with a Huber loss within one standard error of the Huber loss of lambda min found via cross validation) performs the variable selection fairly well.

Table 1: Beta Estimations for different methods simulation setting 1

	True	OLS	My Huber	CVXR Huber	Hqreg Huber	My CV.Hub.Lasso.B.Min	My CV.Hub.Lasso.B.1SE
β_0	0.0	-0.1978889	-0.0931066	-0.0930684	-0.0929563	-0.0931530	-0.1165291
β_1	0.1	-0.2876345	-0.0085941	-0.0084859	-0.0093276	0.0000000	0.0000000
β_2	0.2	-0.3131428	0.1282096	0.1281852	0.1287591	0.1128706	0.0000000
β_3	0.3	0.4468167	0.3571887	0.3571977	0.3581159	0.3447685	0.0000000
β_4	0.4	-0.0554169	0.4053966	0.4053116	0.4062180	0.3996518	0.2027747
β_5	0.5	0.0708962	0.3926379	0.3926452	0.3917839	0.3858315	0.1015183
β_6	0.6	0.2256396	0.5173872	0.5174786	0.5171889	0.5158208	0.4006142
β_7	0.7	0.3302164	0.8337441	0.8336382	0.8358376	0.8135584	0.5153348
β_8	0.8	0.7110359	0.7254494	0.7255425	0.7249398	0.7217221	0.5986224
β_9	0.9	-0.0865526	0.8544701	0.8543863	0.8533433	0.8494088	0.4835031
β_{10}	1.0	0.6541675	1.0387491	1.0388021	1.0382329	0.9895806	0.6280313
β_{11}	0.0	-0.0183600	-0.0617479	-0.0618105	-0.0614698	-0.0104453	0.0000000
β_{12}	0.0	-0.7001126	0.0157001	0.0157807	0.0166061	0.0000000	0.0000000
β_{13}	0.0	0.1399218	-0.0470160	-0.0470116	-0.0476512	-0.0245906	0.0000000
β_{14}	0.0	-0.1439347	-0.0165171	-0.0165403	-0.0160759	0.0000000	0.0000000
β_{15}	0.0	0.7692238	0.0716805	0.0716567	0.0718036	0.0463708	0.0000000
β_{16}	0.0	-0.3682829	0.0640127	0.0640973	0.0634395	0.0686542	0.0000000
β_{17}	0.0	0.0630192	0.0418861	0.0418148	0.0422477	0.0166664	0.0000000
β_{18}	0.0	0.2931244	0.0920068	0.0919742	0.0919036	0.0412015	0.0000000
β_{19}	0.0	0.1849113	-0.0853017	-0.0852670	-0.0852061	-0.0224125	0.0000000

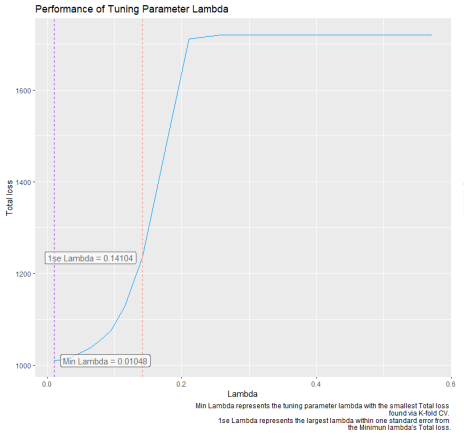

Figure 1: Huber Loss in relation to λ for Simulation

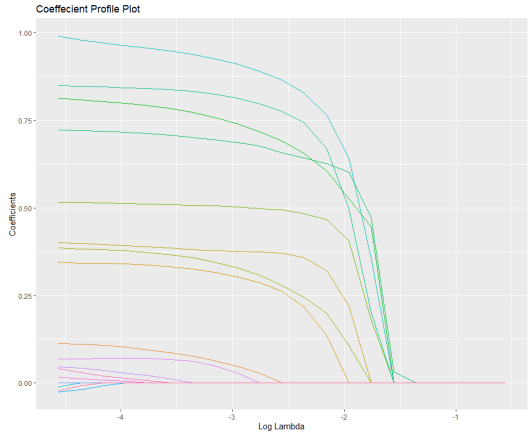

Figure 2: Coefficient plot for Simulation

In Figure 1 we can see that the lambda min is very close to zero indicating that the best lambda is around zero. We can also see from the Figure 2 that the coefficients converge to zero at -1.5 on the log lambda scale.

In Table 2 below it is shown that our Huber regression performs well against CVXR with low n, but with high n and p the cd_huber method is much slower. Also, the hqreg consistently preforms fastest out of the three methods. The reason why my method is so slow for large n may be because of how we are calculating the Weight and Loss functions. To do both calculations we implement a simple for loop and if else statement. This is something I wish to investigate further and see if I can do these calculations in a more efficient manner to better improve calculation time.

From the Table 3 below that we still observe high computation times and for a sample size of n = 400 and p = 200 it takes on average 52 secs to do the cross validation. This further highlights the scalability issues with my implementation. For the accuracy of our implementation of Huber Lasso Cross-Validation we can observe from the Table 3 that our average L2 error of beta estimates improve as more information is available. The standard error of the L2 errors of the beta coefficients also

decrease as expected with more information provided to the model, but we also must consider the computation time for larger sample sizes.

Table 2: Huber Simulation setting 2 Time Comparison (Milliseconds)

expr	mean	median	neval
n = 50 , p = 20			
my hub	5.012370	4.98265	20
cvxr	126.231430	125.94810	20
hqreg	0.285040	0.30385	20
n = 50 , p = 100			
my hub	47.125360	46.63330	20
cvxr	126.986090	126.03570	20
hqreg	1.052860	1.03135	20
n = 50 , p = 300			
my hub	46.363715	46.49215	20
cvxr	138.849630	131.55430	20
hqreg	0.771710	0.76295	20
n = 200 , p = 20			
my hub	32.840010	32.85885	20
cvxr	129.973700	131.16845	20
hqreg	0.392450	0.41185	20
n = 200 , p = 100			
my hub	1329.591700	1318.75190	20
cvxr	136.516030	137.33525	20
hqreg	1.785265	1.76260	20
n = 200 , p = 300			
my hub	4742.144035	4749.02710	20
cvxr	154.304300	153.77885	20
hqreg	14.424120	14.42580	20
n = 500 , p = 20			
my hub	147.919385	146.82325	20
cvxr	148.375530	139.23595	20
hqreg	0.725915	0.72840	20
n = 500 , p = 100			
my hub	1737.819735	1738.48920	20
cvxr	157.349135	157.58430	20
hqreg	2.263410	2.23650	20
n = 500 , p = 300			
my hub	70405.833415	70750.10335	20
cvxr	296.590445	259.71950	20
hqreg	10.947095	10.80250	20

Table 3: Huber Lasso Regression Cross Validation Simulation

	p = 30	p = 100	p = 200
Average Time in Milliseconds			
n = 150	644.1934824	5.253257e+03	2.869037e+04
n = 300	2483.8338971	1.178094e+04	3.788518e+04
n = 400	4320.1789975	1.770577e+04	5.253052e+04
Average L2 Error of Beta Min			
n = 150	0.6828216	8.525324e-01	7.385335e-01
n = 300	0.5047258	5.220552e-01	5.434010e-01
n = 400	0.3919515	3.892076e-01	4.083907e-01
SE L2 Error of Beta Min			
n = 150	0.0956117	1.292345e-01	6.241440e-02
n = 300	0.0391608	5.273170e-02	3.053900e-02
n = 400	0.0440564	2.669530e-02	2.921810e-02
Average L2 Error of Beta 1SE			
n = 150	1.0404418	1.361417e+00	1.172126e+00
n = 300	1.0086097	9.311054e-01	9.719488e-01
n = 400	0.9764258	9.261635e-01	9.387065e-01
SE L2 Error of Beta 1SE			
n = 150	0.0981282	1.103856e-01	7.310670e-02
n = 300	0.0422598	4.329100e-02	4.528580e-02
n = 400	0.0421968	4.314620e-02	3.088180e-02

For these simulation results we can see that the accuracy of our results is good, but the computation time involved for higher dimensions highlights the issues in efficiency. We can further observe that while computation time does increase as the number of covariates increases it also increases substantially with an increase in the number of observations. This demonstrates that there may be a problem with the computation of observation individual computations such as the weight matrix and residuals. As mentioned earlier these are both calculated in a for loop, so a more efficient method for both calculations is needed. The algorithm used in this paper was mainly focused on the computation of beta coefficients using coordinate descent, but while this method is computationally efficient the slowdowns of the weight and loss functions hide the advantages of the method.

4 Boston Housing Data

The Boston Housing data set is based on the data collected by the United States Census Service that collected information on the housing prices in the Boston, MA area. For this paper we used the Boston Housing data included with the MASS R package Venables and Ripley (2002). This data set has known influential observations that makes using robust regression methods such as Huber regression a good option for this data.

The data set description is as follows:

Explanatory variables:

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

Response Variable:

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

To speed up efficiency of the algorithm we scaled the data such that $\mu_j = 0$ and $\sigma_j = 1$ for $j = 1, \ldots, 13$.

Table 4: Boston Housing Beta Coefficient Estimation Comparisons

	OLS	My Huber	CVXR Huber	Hqreg Huber	My CV.Hub.Lasso.B.Min	My CV.Hub.Lasso.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

From our Huber Regression Models shown by their coefficients in Table 4 our method still estimates well in comparison with CVXR and hqreg for regular Huber Regression. It seems that for my Huber Lasso Regression that Beta min has somewhat similar beta coefficients as the other methods, and it identifies β_3 as not significant that corresponds to the results of the results of the OLS method that indus (proportion of non-retail business acres per town) is not a significant predictor. The beta min also identifies that the explanatory variable rad (index of accessibility to radial highways) as not a significant predictor for predicting the value of homes. For the Beta 1 standard error from the beta min it seems that too much penalty is applied here. Based on the results of my Huber regression coefficients and Huber lasso Regression Coefficients it shows that β_6 or rm (average number of rooms per dwelling) and β_{12} lstat (lower status of the population (percent)) have the greatest impact on predicting housing prices in Boston for each unit increase.

In the Figure 3 we see that the min lambda is very close to zero which indicates that there are few predictors that are insignificant. This corresponds with the results of the ordinary least squares method which found only indus and age to be non-significant predictors. For the coefficient plot shown in the Figure 4 we can see that all the coefficients converge to zero around log lambda value of -2.75 or a lambda value 0.064.

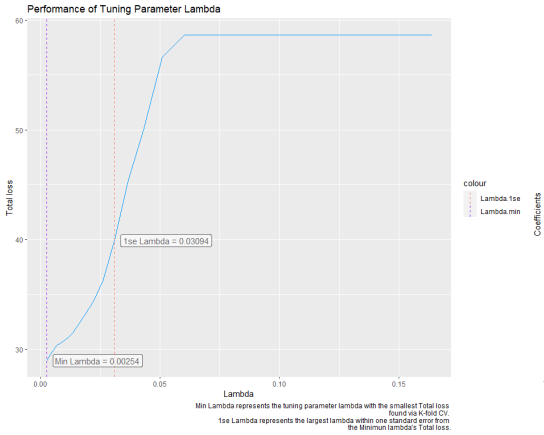

Figure 3: Huber Loss in relation to λ for Boston Housing Data

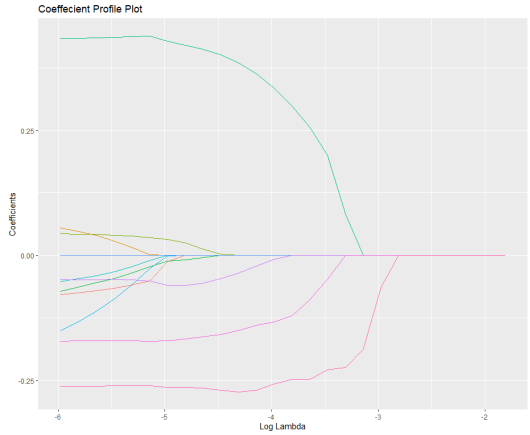

Figure 4: Coefficient plot for Boston Housing Data

5 Conclusion and Future Work

Overall, it seems that the estimation of the functions is consistent with existing R packages, but the efficiency of the implementation is not good in higher dimensions. The first thing want to work on in the future is improving the computation efficiency of computing the weight and loss functions. The hqreg R package by Yi and Huang (2017) proposes the use of a semismooth Newton coordinate descent algorithm which they show to scale well with high dimensions $p >> n$. In the future I would like to implement their coordinate descent algorithm to improve the efficiency of computations and convergence in my R package HuberReg. I also want to take advantage of parallelization features that that can be implemented for loops such as in the cross-validation implementation in Rcpp. For the real data set Boston Housing most of the predictors were significant so the advantages of Huber Lasso Regression are not highlighted for this data in the case of variable selection. Moving forward I would like to analyze my results on a real data set that is higher dimensional and compare it with existing R packages.

6 Code

6.1 Code used for Simulation

```
library(HuberReg)
Simulation1 <- function(t,n,p,rho,q = 10,kfold = 5,seed = 6341,
                        maxiter = 1000,secs = FALSE) {
  #set seed
  set.seed(seed)

  # initialize parameters
  beta <- c(seq(0.1,1,.1),rep(0,p-q))
  lambda.list <- sqrt(log(p)/n)*exp( seq(from = -10,to =5)/4)

  # initialize matrices
  Beta_min <- matrix(nrow = t,ncol = p)
  Beta_1se <- matrix(nrow = t,ncol = p)

  time <- numeric(t) #initializes the time vector
  # that will store the time taken for the M ridge computations

  # for each loop calculate and record the Beta.min and Beta_1se
  for (i in 1:t) {
     # generate data
    data <-  data.generate(n, beta, rho,pr = 0.1, intercept = T,seed = NULL)
    data$X <- scale(data$X)
    start.time <- Sys.time()
    temp2 <- cv.lasso.huber(data = data,lambda.list = lambda.list,
                            intercept = T,plot = F,trace = F,maxiter = maxiter)
    end.time <- Sys.time()
    # records the amount of time taken for step 4
    if (secs) {
      time.taken <- difftime(end.time,start.time,units = "secs")
      # time is recorded in seconds so multiply by 1000 for milliseconds
    }
    else {
      time.taken <- difftime(end.time,start.time,units = "secs")*1000
      # time is recorded in seconds so multiply by 1000 for milliseconds

    }

    Beta_min[i,] <- temp2$beta_min
    Beta_1se[i,] <- temp2$beta_1se
    time <- c(time,time.taken)
  }

  #L_2 errors of each estimator

  L2.err.beta_min <- sqrt(apply(sweep(Beta_min,2,beta)^2,1,sum))
  L2.err.beta_1se <- sqrt(apply(sweep(Beta_1se,2,beta)^2,1,sum))

  #print(L2.err.beta_1se)
  # Beta.min

  Beta_min_Mean <- mean(L2.err.beta_min)
  # calculates the mean of the L2 errors for lambda min
  Beta_min_se <- sd(L2.err.beta_min)/sqrt(t)
  # calculates the se of the L2 errors for lambda min
```

```
  # Beta_1se

  Beta_1se_Mean <- mean(L2.err.beta_1se)
  # calculates the mean of the L2 errors for lambda 1se
  Beta_1se_se <- sd(L2.err.beta_1se)/sqrt(t)
  # calculates the se of the L2 errors for lambda 1se

  time.avg <- mean(time)
  # the average running time (in milliseconds) of the "cv.lasso.huber" function



  return(list(time.avg = time.avg,
              Beta_min_Mean = Beta_min_Mean, Beta_min_se = Beta_min_se,
              Beta_1se_Mean = Beta_1se_Mean, Beta_1se_se = Beta_1se_se))
}
```

6.2 Code for Table 1

```
library(HuberReg)
set.seed(6341)
n <- 500; p <- 20; rho <- 0.5; q = 11
beta_true <-  c(seq(0,1,.1),rep(0,p-q))
lambda.list <- sqrt(log(p)/n)*exp( seq(from = -10,to =10)/5)

data <- data.generate(n, beta_true, rho,pr = 0.1, intercept = T,seed = 6341)

M <- IQR(data$y)/10

set.seed(6341)
beta_cv_lasso_huber <- cv.lasso.huber(data = data,
        lambda.list = lambda.list,intercept = T,plot = T)

best_lambda <- beta_cv_lasso_huber$lam.min

# CVXR estimate
library(CVXR)
beta <- Variable(dim(data$X)[2]+1)
obj <- sum(CVXR::huber(data$y - cbind(1,data$X) %*% beta, M))
prob <- Problem(Minimize(obj))
result <- solve(prob)
beta_hub <- result$getValue(beta)

# hqreg estimate
library(hqreg)
beta_hqreg <- unname(hqreg_raw(data$X, data$y, method = c("huber"),
        gamma = M, tau = 0.5, alpha = 1, nlambda = 2, lambda.min = best_lambda,
        lambda = c(best_lambda,best_lambda),
        intercept = TRUE,max.iter = 1000, eps = 1e-5,
        message = FALSE,penalty.factor = rep(0,p-1))$beta[,1])


beta_ols <- unname(coef(lm(data$y~data$X)))

beta_cdhuber <- cd_huber(data$X,data$y,M = M,intercept = T,trace = FALSE)$beta

library(kableExtra)
temp <- data.frame(beta_true,beta_ols,beta_cdhuber,beta_hub,
```

```
                    beta_hqreg, beta_cv_lasso_huber$beta_min,
                    beta_cv_lasso_huber$beta_1se)
rownames(temp) <- paste("$\\beta_{",0:(p-1),"}$",sep = "")
colnames(temp) <- c("True", "OLS", "My Huber",
                    "CVXR Huber","Hqreg Huber",
                    "My CV.Hub.Lass.B.Min", "My CV.Hub.Lass.B.1SE")
knitr::kable(temp[1:20,], booktabs = T,escape = F,
    caption = "Beta Estimations for different methods") %>%
    kable_styling(latex_options = c("scale_down"))
```

6.3 Code for Figure 1

Run Code from 6.2 and then call the fig

```
beta_cv_lasso_huber$fig
```

6.4 Code for Figure 2

```
library(HuberReg)
set.seed(6341)
n <- 500; p <- 20; rho <- 0.5; q = 11
beta_true <-  c(seq(0,1,.1),rep(0,p-q))
lambda.list <- sqrt(log(p)/n)*exp( seq(from = -10,to =10)/5)

data <- data.generate(n, beta_true, rho,pr = 0.1, intercept = T,seed = 6341)

M <- IQR(data$y)/10
plot_cd_huber_Lasso(data = data,M = M,
    lambda.list = lambda.list,trace = FALSE,intercept = T)
```

6.5 Code for Table 2

Function that performs CVXR Huber regression

```
CVXr <- function(data,intercept = TRUE,M = NULL){
  require(CVXR)
  if(is.null(M)) {
    M <- IQR(data$y)/10
  }

  if(intercept) {
    beta <- Variable(dim(data$X)[2]+1)
    obj <- sum(CVXR::huber(data$y - cbind(1,data$X) %*% beta, M))
  }
  else {
    beta <- Variable(dim(data$X)[2])
    obj <- sum(CVXR::huber(data$y - data$X %*% beta, M))
  }
  prob <- Problem(Minimize(obj))
  result <- solve(prob)
  return(result$getValue(beta))
}
```

Simulation settings and table creation

```
p.list <- c(20, 100, 300)
n.list <- c(50, 200, 500)
library(microbenchmark)
tables2 <- c()
set.seed(6341)
setting <- c()
#u <- 1
for (i in 1:length(n.list)) {
  # iterates through the different number of sample sizes
  n <- n.list[i]
  #u <- i
  for (j in 1:length(p.list)) {
    # iterates through the different number of predictors
    p <- p.list[j]
    rho <- 0.5; q = 11
    beta_true <-  c(seq(0,1,.1),rep(0,p-q))

    data <- data.generate(n, beta_true, rho,pr = 0.1, intercept = T,seed = NULL)
    data$X <- scale(data$X)
    M <- IQR(data$y)/10
    setting <- c(setting,paste("n = ", n,", p = ",p))
    #print(paste("n = ", n,", p = ",p))
    yelp <- microbenchmark(unit = "ms",cd_huber(data$X,data$y,M = M,
                           intercept = T,trace = FALSE,maxiter = 1000)$beta,
                           CVXr(data),
                           unname(hqreg_raw(data$X, data$y, method = c("huber"),
                           gamma = M, tau = 0.5, alpha = 1, nlambda = 2,
                           lambda.min = 0,lambda = c(0,0), intercept = TRUE,
                           max.iter = 1000, eps = 1e-5, message = FALSE,
                           penalty.factor = rep(0,p-1))$beta[,1]),
                           times = 20L)
    levels(yelp$expr) <- c("my hub","cvxr","hqreg")
    tables2 <- rbind(tables2,summary(yelp))
  }
}
library(kableExtra)
temp6 <- as.data.frame(tables2)
knitr::kable(temp6[,c(1,4,5,8)],booktabs = T,longtable = T,
  caption= "Huber Simulation Time Comparison (Milliseconds)") %>%
  kable_styling(latex_options = c("hold_position"))  %>%
  pack_rows(setting[1], 1, 3) %>%
  pack_rows(setting[2], 4, 6) %>%
  pack_rows(setting[3], 7, 9) %>%
  pack_rows(setting[4], 10, 12) %>%
  pack_rows(setting[5], 13, 15) %>%
  pack_rows(setting[6], 16, 18) %>%
  pack_rows(setting[7], 19, 21) %>%
  pack_rows(setting[8], 22, 24) %>%
  pack_rows(setting[9], 25, 27)
```

6.6 Code for Table 3

```
p.list <- c(30, 100, 200)
n.list <- c(150, 300, 400)

tables <- matrix(nrow = 5*length(n.list),ncol = length(p.list))

#u <- 1
```

```
for (i in 1:length(n.list)) {
  # iterates through the different number of sample sizes
  n <- n.list[i]
  #u <- i
  for (j in 1:length(p.list)) {
    # iterates through the different number of predictors
    p <- p.list[j]
    # calculates the the avgerage simulation time for a given (n,p)
    # and saves the result
    sim <- Simulation1(t = 10, n = n, p = p,rho = 0.5,
    q = 10,seed = 6341,maxiter,maxiter = 10000)# saves the (n,p) used

    tables[i,j] <- sim$time.avg
    tables[(i+length(n.list)),j] <- sim$Beta_min_Mean
    tables[(i+2*length(n.list)),j] <- sim$Beta_min_se
    tables[(i+3*length(n.list)),j] <- sim$Beta_1se_Mean
    tables[(i+4*length(n.list)),j] <- sim$Beta_1se_se
  }
}
library(kableExtra)
temp1 <- as.data.frame(tables)
rownames(temp1) <-  paste(rep(c("␣","␣␣","␣␣␣","␣␣␣␣","␣␣␣␣␣"),each=3),
    rep(paste("n␣=",n.list,"␣"),5) )
colnames(temp1) <- paste("p␣=",p.list)

knitr::kable(temp1,booktabs = T,longtable = T,
  caption= "Huber␣Lasso␣Regression␣Cross␣Validation␣Simulation") %>%
  kable_styling(latex_options = c("hold_position"),font_size = 7) %>%
  pack_rows("Average␣Time␣in␣Milliseconds", 1, 3) %>%
  pack_rows("Average␣L2␣Error␣of␣Beta␣Min", 4, 6) %>%
  pack_rows("SE␣L2␣Error␣of␣Beta␣Min", 7, 9) %>%
  pack_rows("Average␣L2␣Error␣of␣Beta␣1SE", 10, 12) %>%
  pack_rows("SE␣L2␣Error␣of␣Beta␣1SE", 13, 15)
```

6.7 Code for Table 4

```
library(MASS)
data(Boston)
data2 <- Boston[,-12]
#data2[,-dim(data2)[2]] <- scale(data2[,-dim(data2)[2]])
data2 <- scale(data2)
data2 <- list(X = as.matrix(data2[,-dim(data2)[2]]),
          y = as.matrix(data2[,dim(data2)[2]]))
set.seed(6341)
p2 <- dim(data2$X)[2]+1
n2 <- dim(data2$X)[1]
lambda.list2 <- sqrt(log(p2)/n2)*exp( seq(from = -20,to =5)/6)


M2 <- IQR(data2$y)/10

set.seed(6341)
beta_cv_lasso_huber2 <- cv.lasso.huber(data = data2,
    lambda.list = lambda.list2,intercept = T,plot = T,trace = F)

best_lambda2 <- beta_cv_lasso_huber2$lam.min

library(CVXR)
```

```
beta2 <- Variable(dim(data2$X)[2]+1)
obj2 <- sum(CVXR::huber(data2$y - cbind(1,data2$X) %*% beta2, M2))
prob2 <- Problem(Minimize(obj2))
result2 <- solve(prob2)
beta_hub2 <- result2$getValue(beta2)

library(hqreg)
beta_hqreg2 <- unname(hqreg_raw(data2$X, data2$y,
    method = c("huber"), gamma = M2, tau = 0.5, alpha = 1,
    nlambda = 2, lambda.min = best_lambda2, lambda = c(best_lambda2,best_lambda2),
    intercept = TRUE, max.iter = 1000, eps = 1e-5,message = FALSE,
    penalty.factor = rep(0,p2-1))$beta[,1])


beta_ols2 <- unname(coef(lm(data2$y~data2$X)))

beta_cdhuber2 <- cd_huber(data2$X,data2$y,M = M2,
                          intercept = T,trace = FALSE)$beta

library(kableExtra)
temp2 <- data.frame(beta_ols2,beta_cdhuber2,beta_hub2,
                    beta_hqreg2,beta_cv_lasso_huber2$beta_min,
                    beta_cv_lasso_huber2$beta_1se)
rownames(temp2) <- paste("$\\beta_{",0:(p2-1),"}$",sep = "")
colnames(temp2) <- c( "OLS", "My Huber",
                    "CVXR Huber","Hqreg Huber",
                    "My CV.Hub.Lass.B.Min", "My CV.Hub.Lass.B.1SE")
knitr::kable(temp2,booktabs = T,escape = F,
  caption = "Boston Housing Beta Coefficient Estimation Comparisons") %>%
  kable_styling(latex_options = c("scale_down"))

fig2 <- plot_cd_huber_Lasso(data = data2,M = M2,
            lambda.list = lambda.list2,trace = FALSE,intercept = T)
```

6.8 Code for Figure 3 run code from 6.7 first

```
beta_cv_lasso_huber2$fig
```

6.9 Code for Figure 4 run code from 6.7 first

```
plot_cd_huber_Lasso(data = data2,M = M2,
    lambda.list = lambda.list2,trace = FALSE,intercept = T)
```

7 Session Info

```
R version 4.3.1 (2023-06-16 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 11 x64 (build 22631)

Matrix products: default


locale:
[1] LC_COLLATE=English_United States.utf8  LC_CTYPE=English_United States.utf8
LC_MONETARY=English_United States.utf8 LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

time zone: America/Chicago
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
base

other attached packages:
[1] HuberReg_0.1.0

loaded via a namespace (and not attached):
[1] compiler_4.3.1     tools_4.3.1        rstudioapi_0.15.0 Rcpp_1.0.11
```

References

Fu, A., Narasimhan, B., and Boyd, S. (2020). Cvxr : An r package for disciplined convex optimization. *Journal of statistical software*, 94(14):1–34.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of mathematical statistics*, 35(1):73–101.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Yi, C. and Huang, J. (2017). Semismooth newton coordinate descent algorithm for elastic-net penalized huber loss regression and quantile regression. *Journal of computational and graphical statistics*, 26(3):547–557.