Chapter 13

Robust and Resistant Regression

When the errors are normal, least squares regression is clearly best but when the errors are nonnormal, other methods may be considered. A particular concern is long-tailed error distributions. One approach is to remove the largest residuals as outliers and still use least squares but this may not be effective when there are several large residuals because of the leave-out-one nature of the outlier tests. Furthermore, the outlier test is an accept/reject procedure that is not smooth and may not be statistically efficient for the estimation of β. Robust regression provides an alternative.

There are several methods. M-estimates choose β to minimize

$$\sum_{i=1}^{n} \rho\left(\frac{y_i - x_i^T\beta}{\sigma}\right)$$

Possible choices for ρ are

1. $\rho(x) = x^2$ is just least squares
2. $\rho(x) = |x|$ is called least absolute deviations regression (LAD). This is also called L_1 regression.
3. $$\rho(x) = \begin{cases} x^2/2 & \text{if } |x| \le c \\ c|x| - c^2/2 & \text{otherwise} \end{cases}$$

   is called Huber's method and is a compromise between least squares and LAD regression. c can be an estimate of σ but not the usual one which is not robust. Something $\propto$ median$|\hat{\varepsilon}_i|$ for example.

Robust regression is related to weighted least squares. The normal equations tell us that

$$X^T(y - X\hat{\beta}) = 0.$$

With weights and in non-matrix form this becomes:

$$\sum_{i=1}^{n} w_i x_{ij}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j) = 0 \quad j = 1, \ldots p$$

Now differentiating the M-estimate criterion with respect to β_j and setting to zero we get

$$\sum_{i=1}^{n} \rho'\left(\frac{y_i - \sum_{j=1}^{p} x_{ij}\beta_j}{\sigma}\right) x_{ij} = 0 \quad j = 1, \ldots p$$

Now let $u_i = y_i - \sum_{j=1}^{p} x_{ij}\beta_j$ to get

$$\sum_{i=1}^{n} \frac{\rho'(u_i)}{u_i} x_{ij}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j) = 0 \quad j = 1, \ldots p$$

so we can make the identification of

$$w(u) = \rho'(u)/u$$

and we find for our choices of ρ above:

1. LS: $w(u)$ is constant.
2. LAD: $w(u) = 1/|u|$ - note the asymptote at 0 - this makes a weighting approach difficult.
3. Huber:
$$w(u) = \begin{cases} 1 & \text{if } |u| \le c \\ c/|u| & \text{otherwise} \end{cases}$$

There are many other choices that have been used. Because the weights depend on the residuals, an iteratively reweighted least squares approach to fitting must be used. We can sometimes get standard errors by $\widehat{\text{var}}\ \hat{\beta} = \hat{\sigma}^2(X^TWX)^{-1}$ (use a robust estimate of σ^2 also).

We demonstrate the methods on the Chicago insurance data. Using least squares first.

```
> data(chicago)
> g <- lm(involact ~ race + fire + theft + age + log(income),chicago)
> summary(g)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.57398    3.85729   -0.93  0.35958
race         0.00950    0.00249    3.82  0.00045
fire         0.03986    0.00877    4.55  4.8e-05
theft       -0.01029    0.00282   -3.65  0.00073
age          0.00834    0.00274    3.04  0.00413
log(income)  0.34576    0.40012    0.86  0.39254

Residual standard error: 0.335 on 41 degrees of freedom
Multiple R-Squared: 0.752,      Adjusted R-squared: 0.721
F-statistic: 24.8 on 5 and 41 degrees of freedom,       p-value: 2.01e-11
```

Least squares works well when there are normal errors but can be upset by long-tailed errors. A convenient way to apply the Huber method is to apply the `rlm()` function which is part of the MASS (see the book Modern Applied Statistics in S+) which also gives standard errors. The default is to use the Huber method but there are other choices.

```
> library(MASS)
> g <- rlm( involact ~ race + fire + theft + age + log(income), chicago)
Coefficients:
            Value  Std. Error t value
(Intercept) -2.926  3.397     -0.861
race         0.008  0.002      3.583
```

```
fire         0.046  0.008      5.940
theft       -0.010  0.002     -3.912
age          0.006  0.002      2.651
log(income)  0.283  0.352      0.803

Residual standard error: 0.249 on 41 degrees of freedom
```

The R^2 and F-statistics are not given because they cannot be calculated (at least not in the same way). The numerical values of the coefficients have changed a small amount but the general significance of the variables remains the same and our substantive conclusion would not be altered. Had we seen something different, we would need to find out the cause. Perhaps some group of observations were not being fit well and the robust regression excluded these points.

Another method that can be used is Least Trimmed Squares(LTS). Here one minimizes $\sum_{i=1}^{q} \hat{\varepsilon}_{(i)}^2$ where q is some number less than n and (i) indicates sorting. This method has a high *breakdown* point because it can tolerate a large number of outliers depending on how q is chosen. The Huber and L_1 methods will still fail if some $\varepsilon_i \to \infty$. LTS is an example of a *resistant* regression method. Resistant methods are good at dealing with data where we expect there to be a certain number of "bad" observations that we want to have no weight in the analysis.

```
> library(lqs)
> g <- ltsreg(involact ~ race + fire + theft + age + log(income),chicago)
> g$coef
(Intercept)        race        fire       theft         age log(income)
 -1.6950187   0.0037348   0.0549117  -0.0095883   0.0018549   0.1700325
> g <- ltsreg(involact ~ race + fire + theft + age + log(income),chicago)
> g$coef
(Intercept)        race        fire       theft         age log(income)
  2.2237795   0.0050697   0.0423565  -0.0084868   0.0008755  -0.2398183
```

The default choice of q is $\lfloor n/2 \rfloor + \lfloor (p+1)/2 \rfloor$ where $\lfloor x \rfloor$ indicates the largest integer less than or equal to x. I repeated the command twice and you will notice that the results are somewhat different. This is because the default genetic algorithm used to compute the coefficients is non-deterministic. An exhaustive search method can be used

```
> g <- ltsreg(involact ~ race + fire + theft + age + log(income),chicago,
               nsamp="exact")
> g$coef
(Intercept)        race        fire       theft         age log(income)
-1.12093591  0.00575147  0.04859848 -0.00850985  0.00076159  0.11251547
```

This takes about 20 minutes on a 400Mhz Intel Pentium II processor. For larger datasets, it will take much longer so this method might be impractical.

The most notable difference from LS for the purposes of this data is the decrease in the race coefficient - if the same standard error applied then it would verge on insignificance. However, we don't have the standard errors for the LTS regression coefficients. We now use a general method for inference which is especially useful when such theory is lacking - the Bootstrap.

To understand how this method works, think about how we might empirically determine the distribution of an estimator. We could repeatedly generate artificial data from the true model, compute the estimate each time and gather the results to study the distribution. This technique, called simulation, is not available to us for real data because we don't know the true model. The Bootstrap emulates the simulation procedure above except instead of sampling from the true model, it samples from the observed data itself. Remarkably, this technique is often effective. It sidesteps the need for theoretical calculations that may be extremely difficult or even impossible. The Bootstrap may be the single most important innovation in Statistics in the last 20 years.

To see how the bootstrap method compares with simulation, let's spell out the steps involved. In both cases, we consider X fixed.

Simulation

In general the idea is to sample from the known distribution and compute the estimate, repeating many times to find as good an estimate of the sampling distribution of the estimator as we need. For the regression case, it is easiest to start with a sample from the error distribution since these are assumed to be independent and identically distributed:

1. Generate ε from the known error distribution.
2. Form $y = X\beta + \varepsilon$ from the known β.
3. Compute $\hat{\beta}$.

We repeat these three steps many times. We can estimate the sampling distribution of $\hat{\beta}$ using the empirical distribution of the generated $\hat{\beta}$, which we can estimate as accurately as we please by simply running the simulation for long enough. This technique is useful for a theoretical investigation of the properties of a proposed new estimator. We can see how its performance compares to other estimators. However, it is of no value for the actual data since we don't know the true error distribution and we don't know the true β.

The bootstrap method mirrors the simulation method but uses quantities we do know. Instead of sampling from the population distribution which we do not know in practice, we resample from the data itself.

Bootstrap

1. Generate ε^* by sampling with replacement from $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$.
2. Form $y^* = X\hat{\beta} + \varepsilon^*$
3. Compute $\hat{\beta}^*$ from (X, y^*)

This time, we use only quantities that we know. For small n, it is possible to compute $\hat{\beta}^*$ for every possible sample from $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$, but usually we can only take as many samples as we have computing power available. This number of bootstrap samples can be as small as 50 if all we want is an estimate of the variance of our estimates but needs to be larger if confidence intervals are wanted.

To implement this, we need to be able to take a sample of residuals with replacement. `sample()` is good for generating random samples of indices:

```
> sample(10,rep=T)
 [1] 7 9 9 2 5 7 4 1 8 9
```

and hence a random sample (with replacement) of RTS residuals is:

```
> g$res[sample(47,rep=T)]
    60639     60641     60634     60608     60608     60612     60651     60620
 0.091422 -0.039899  0.013526  0.342344  0.342344 -0.022214  0.255031  0.333714
```

(rest deleted

You will notice that there is a repeated value even in this small snippet. We now execute the bootstrap - first we make a matrix to save the results in and then repeat the bootstrap process 1000 times: (This takes about 6 minutes to run on a 400Mhz Intel Pentium II processor)

```
> x <- model.matrix(˜ race+fire+theft+age+log(income),chicago)[,-1]
> bcoef <- matrix(0,1000,6)
> for(i in 1:1000){
+ newy <- g$fit + g$res[sample(47,rep=T)]
+ brg <- ltsreg(x,newy,nsamp="best")
+ bcoef[i,] <- brg$coef
+ }
```

It is not convenient to use the `nsamp="exact"` since that would require 1000 times the 20 minutes it takes to make original estimate. That's about two weeks, so I compromised and used the second best option of `nsamp="best"`. This likely means that our bootstrap estimates of variability will be somewhat on the high side. This illustrates a common practical difficulty with the bootstrap — it can take a long time to compute. Fortunately, this problem recedes as processor speeds increase. It is notable that this calculation was the only one in this book that did not take a negligible amount of time. You typically do not need the latest and greatest computer to do statistics on the size of datasets encountered in this book.

To test the null hypothesis that $H_0 : \beta_{race} = 0$ against the alternative $H_1 : \beta_{race} > 0$ we may figure what fraction of the bootstrap sampled β_{race} were less than zero:

```
> length(bcoef[bcoef[,2]<0,2])/1000
[1] 0.019
```

So our p-value is 1.9% and we reject the null at the 5% level.

We can also make a 95% confidence interval for this parameter by taking the empirical quantiles:

```
> quantile(bcoef[,2],c(0.025,0.975))
      2.5%      97.5%
0.00099037 0.01292449
```

We can get a better picture of the distribution by looking at the density and marking the confidence interval:

```
> plot(density(bcoef[,2]),xlab="Coefficient of Race",main="")
> abline(v=quantile(bcoef[,2],c(0.025,0.975)))
```

See Figure 13.1. We see that the distribution is approximately normal with perhaps so longish tails.

This would be more accurate if we took more than 1000 bootstrap resamples. The conclusion here would be that the race variable is significant but the effect is less than that estimated by least squares. Which is better? This depends on what the ”true” model is which we will never know but since the QQ plot did not indicate any big problem with non-normality I would tend to prefer the LS estimates. However, this does illustrate a general problem that occurs when more than one statistical method is available for a given dataset.

Summary

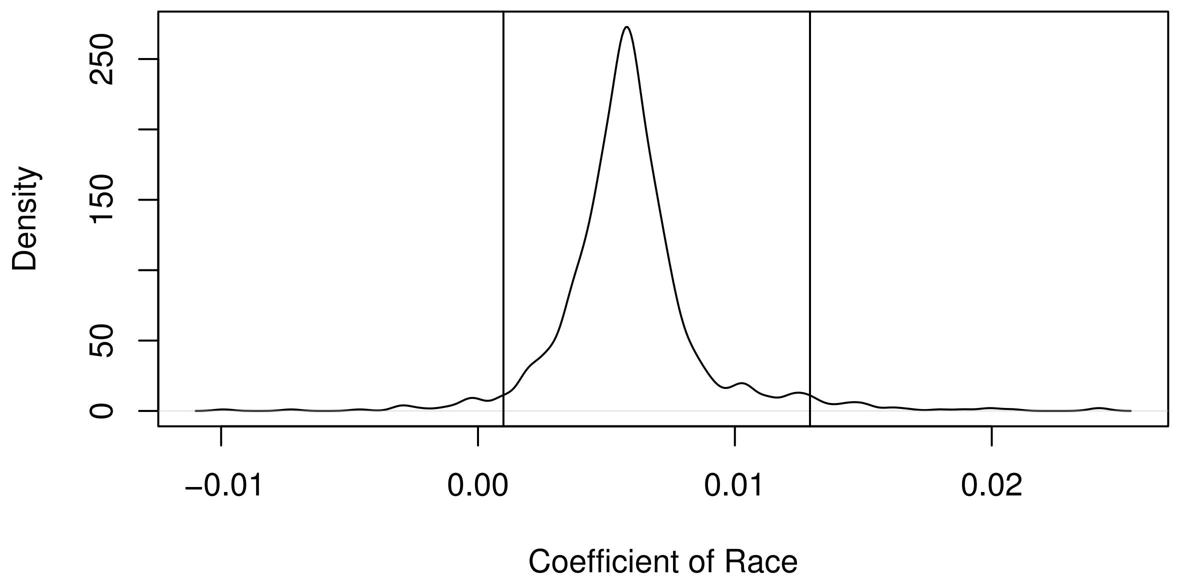

Figure 13.1: Bootstrap distribution of $\hat{\beta}_{race}$ with 95% confidence intervals

1. Robust estimators provide protection against long-tailed errors but they can't overcome problems with the choice of model and its variance structure. This is unfortunate because these problems are more serious than non-normal error.

2. Robust estimates just give you $\hat{\beta}$ and possibly standard errors without the associated inferential methods. Software and methodology for this inference is not easy to come by. The bootstrap is a general purpose inferential method which is useful in these situations.

3. Robust methods can be used in addition to LS as a confirmatory method. You have cause to worry if the two estimates are far apart.