HW5

Jinhong Du - 12243476

2019/10/08

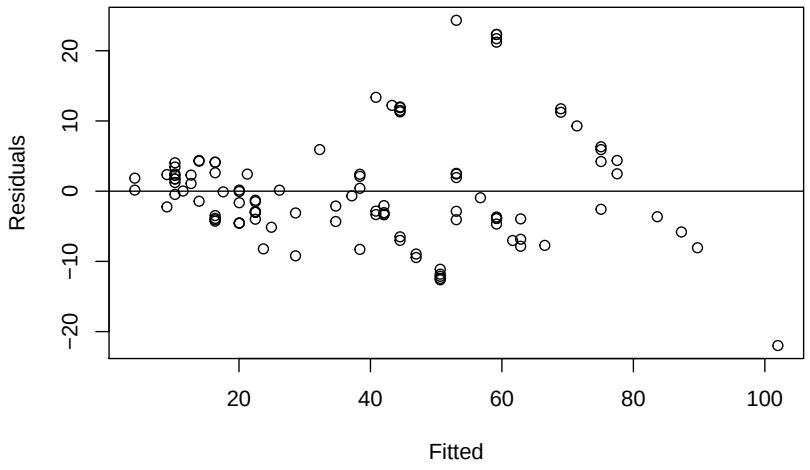
Contents

Problem 1 . . . 2
- (a) . . . 2
- (b) . . . 2
- (c) . . . 3
Problem 2 . . . 4
Problem 3 . . . 7
- (a) . . . 7
- (b) . . . 8
Problem 4 . . . 11
- (a) . . . 11
- (b) . . . 11

1. (Faraway 6.1) Researchers at National Institutes of Standards and Technology (NIST) collected `pipeline` data on ultrasonic measurements of the depths of defects in the Alaska pipeline in the field. The depth of the defects were then remeasured in the laboratory. These measurements were performed in six different batches. It turns out that this batch effect is not significant and so can be ignored in the analysis that follows. The laboratory measurements are more accurate than the in-field measurements, but more time consuming and expensive. We want to develop an regression equation for correcting the in-field measurements.

(a) Fit a regression model `Lab` $\sim$ `Field`. Check for nonconstant variance.

From the plot below, as fitted value increases, the residual seems to fluctuate more greatly, i.e. larger variance. So there is nonconstant varaince.




```
library(faraway)
data(pipeline)

model <- lm(Lab~Field, pipeline)
plot(fitted(model),residuals(model),xlab="Fitted",ylab="Residuals")
abline(h=0)
```

(b) We wish to use weights to account for the nonconstant variance. Here we split the range of `Field` into 12 groups of size nine (except for the last group which has only eight values). Within each group, we compute the variance of `Lab` as varlab and the mean of `Field` as `meanfield`. Supposing pipeline is the name of your data frame, the following R code will make the needed computations:

```
> i < - order(pipeline$Field)
> npipe < - pipeline [i,]
> ff < - gl (12, 9) [-108]
> meanfield < - unlist (lapply (split (npipe$Field, ff),mean))
> varlab < - unlist (lapply (split (npipe$Lab, ff), var))
```

Suppose we guess that the the variance in the response is linked to the predictor in the following way:

$$Var(Lab) = \alpha_0 Field^{\alpha_1}.$$

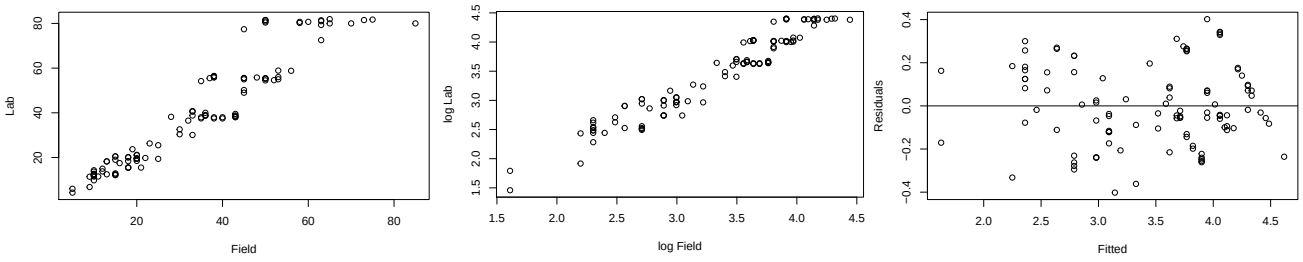
Regress log(`varlab`) on log(`meanfield`) to estimate α_0 and α_1. (You might choose to remove the last point.) Use this to determine appropriate weights in a WLS fit of Lab on Field. Show the regression summary.

```
i <- order(pipeline$Field)
npipe <- pipeline[i,]
ff <- gl(12, 9)[-108]
meanfield <- unlist(lapply(split(npipe$Field, ff), mean))
varlab <- unlist(lapply(split(npipe$Lab, ff), var))

df <- data.frame(varlab=varlab, meanfield=meanfield)
model <- lm(log(varlab)~log(meanfield), df)
w <- 1/ (exp(coef(model)[1]) * npipe$Field^coef(model)[2])

model <- lm(Lab~Field, npipe, weights=w)
summary(model)
```

```
##
## Call:
## lm(formula = Lab ~ Field, data = npipe, weights = w)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0826 -0.8102 -0.3189  0.6212  3.4429
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.49436    0.90707  -1.647    0.102
## Field        1.20828    0.03488  34.637   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.169 on 105 degrees of freedom
## Multiple R-squared:  0.9195, Adjusted R-squared:  0.9188
## F-statistic:  1200 on 1 and 105 DF,  p-value: < 2.2e-16
```

(c) An alternative to weighting is transformation. Find transformations on Lab and/or Field so that in the transformed scale the relationship is approximately linear with constant variance. You may restrict your choice of transformation to square root, log and inverse.

As we can see from the first plot, the variances of `Filed` and `Lab` are both getting bigger as the value of covariate increases. So first try do log transformation such that the extreme value of `Filed` and `Lab` will become less extreme (as in the second plot below). And from the plot of residuals vs fitted values, we can see that this transformed model is approximately linear with constant variance.




```
plot(pipeline$Field, pipeline$Lab, xlab="Field",ylab="Lab")
plot(log(pipeline$Field), log(pipeline$Lab), xlab="log Field",ylab="log Lab")
model <- lm(log(Lab)~log(Field), pipeline)
plot(fitted(model),residuals(model),xlab="Fitted",ylab="Residuals")
abline(h=0)
```

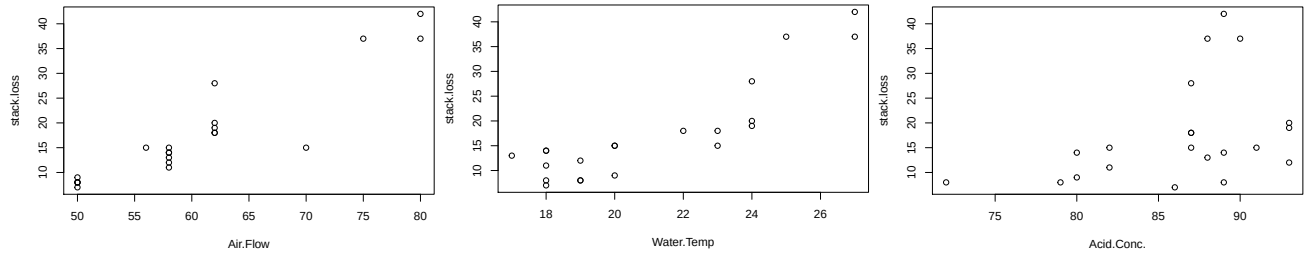
2. (Faraway 6.5) Using the `stackloss` data, fit a model with stack. loss as the response and the other three variables as predictors using the following methods:

- **Least squares**
- **Least absolute deviations**
- **Huber method**
- **Least trimmed squares**

Compare the results. Now use diagnostic methods to detect any outliers or influential points. Remove these points and then use least squares. Compare the results.

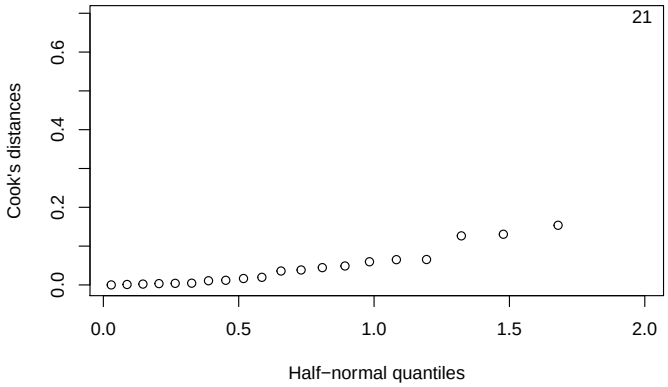
For these four methods, $\hat{\beta}_0$ and $\hat{\beta}_1$ are almost the same, while $\hat{\beta}_2$ of the LS model has largest value among four models. Also, $\hat{\beta}_2$ and $\hat{\beta}_3$ of the LTS model are smallest among four model, which may be conservative.

Checking for outliers:



From plots of response vs every predictor, we can see the point with `Acid.Conc.`<75 may be an outlier.

Checking for influential points:



Here, the 21th sample has extremely largest cook distance so it's an influential point.

After removing the outlier and the influential point, $\hat{\beta}_1$ of the LS model becomes smaller and similar to that of other model.

```
data(stackloss)
model_ls <- lm(stack.loss~., stackloss)
sumary(model_ls)
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -39.91967   11.89600 -3.3557   0.00375
## Air.Flow      0.71564    0.13486  5.3066 5.799e-05
## Water.Temp    1.29529    0.36802  3.5196   0.00263
## Acid.Conc.   -0.15212    0.15629 -0.9733   0.34405
##
## n = 21, p = 4, Residual SE = 3.24336, R-Squared = 0.91
```

```r
library(L1pack)
model_lad <- lad(stack.loss~., stackloss)
summary(model_lad)
```

```
## Call:
## lad(formula = stack.loss ~ ., data = stackloss)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.4812 -1.2174  0.0000  0.5275  7.6348
##
## Coefficients:
##             Estimate Std.Error Z value  p-value
## (Intercept) -39.6899  14.6995   -2.7001   0.0069
## Air.Flow      0.8319   0.1666    4.9921   0.0000
## Water.Temp    0.5739   0.4548    1.2620   0.2069
## Acid.Conc.   -0.0609   0.1931   -0.3152   0.7526
##
## Degrees of freedom: 21 total; 17 residual
## Scale estimate: 2.833893
## Log-likelihood: -50.15272 on 5 degrees of freedom
```

```r
require(MASS)
model_huber <- rlm(stack.loss~., stackloss)
summary(model_huber)
```

```
##
## Call: rlm(formula = stack.loss ~ ., data = stackloss)
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.91753 -1.73127  0.06187  1.54306  6.50163
##
## Coefficients:
##             Value    Std. Error t value
## (Intercept) -41.0265   9.8073    -4.1832
## Air.Flow      0.8294   0.1112     7.4597
## Water.Temp    0.9261   0.3034     3.0524
## Acid.Conc.   -0.1278   0.1289    -0.9922
##
## Residual standard error: 2.441 on 17 degrees of freedom
```

```r
model_lts <- ltsreg(stack.loss~., stackloss)
coef(model_lts)
```

```
##   (Intercept)      Air.Flow    Water.Temp    Acid.Conc.
## -3.429167e+01  7.142857e-01  3.571429e-01 -4.485978e-17
```

```r
# Outlier
plot(stack.loss ~ Air.Flow, stackloss)
plot(stack.loss ~ Water.Temp, stackloss)
plot(stack.loss ~ Acid.Conc., stackloss)

stackloss[stackloss$Acid.Conc.<75,]
```

```
##    Air.Flow Water.Temp Acid.Conc. stack.loss
## 17       50         19         72          8
```

```r
# Influencial points
cook <- cooks.distance(model_ls)
halfnorm(cook,1,ylab="Cook's distances")

# Refit ls model
model_ls <- lm(stack.loss~., stackloss[-c(17,21),])
sumary(model_ls)
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -41.57059   12.47322 -3.3328  0.004543
## Air.Flow      0.88589    0.12299  7.2028 3.059e-06
## Water.Temp    0.83155    0.33920  2.4515  0.026962
## Acid.Conc.   -0.13280    0.15855 -0.8376  0.415392
##
## n = 19, p = 4, Residual SE = 2.64677, R-Squared = 0.95
```

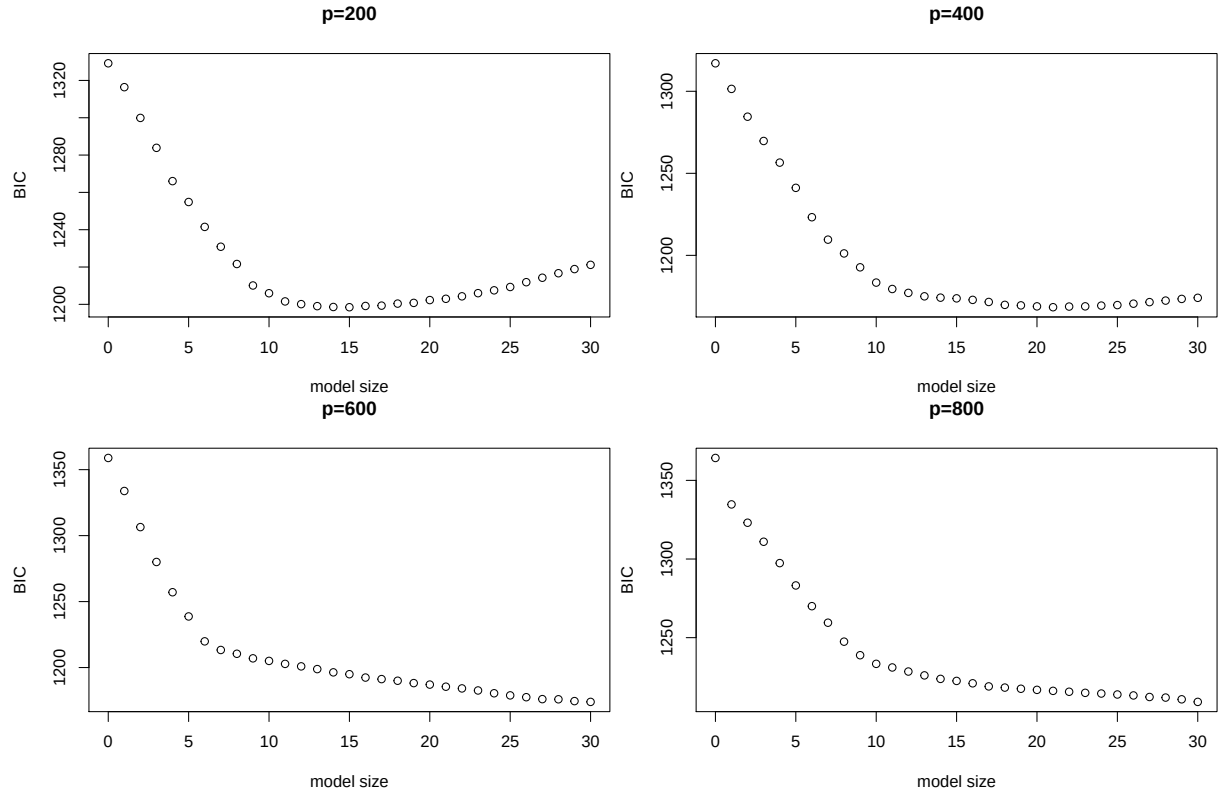
3. Multiple testing simulation. First generate data:

- **Generate $X \in \mathbb{R}^{n \times p}$ with i.i.d. $N(0,1)$ entries, then normalize the columns to have unit norm.**
- **Let the true coefficient vector be $\beta = (\underbrace{5, \ldots, 5}_{10 \text{ times}}, \underbrace{0, \ldots, 0}_{p-10 \text{ times}})$.**
- **Generate $Y = X\beta +$ (noise) where the noise values are i.i.d. $N(0,1)$.**

Run this simulation with $n = 400$ and try each value $p = 200, 400, 600, 800$. You may want to run both parts of the simulation a few times to get a clear picture of what the results typically look like, since these plots may be fairly noisy, especially for the second part.

(a) First, run forward stepwise starting with a model of size 0 (intercept only) and up to a model of size 30. (You can either implement forward selection "by hand", or use pre-existing R code as long as you're certain it's doing the same thing.) Evaluate the BIC at each model size, and plot BIC against model size. Repeat this for each choice of p, the total number of available covariates. Compare the resulting plots you see. How does BIC perform when p is smaller—does it do a good job of picking an appropriate model? What goes wrong as p increases?

For small p, at first the BIC decreases as the model size increases and then increases as the model size increases. So it does a good job of picking an appropriate model with model size being 10 ($p = 200$). While for large p, as the model size increases, the BIC keeps decreases, which does not serve as a good indicator of model picking. BIC will tend to select saturated models as p gets too large.




```
n_simulation <- 100
n <- 400
for (p in c(1:4)*200){
    X <- matrix(rnorm(n*p), n, p)
```

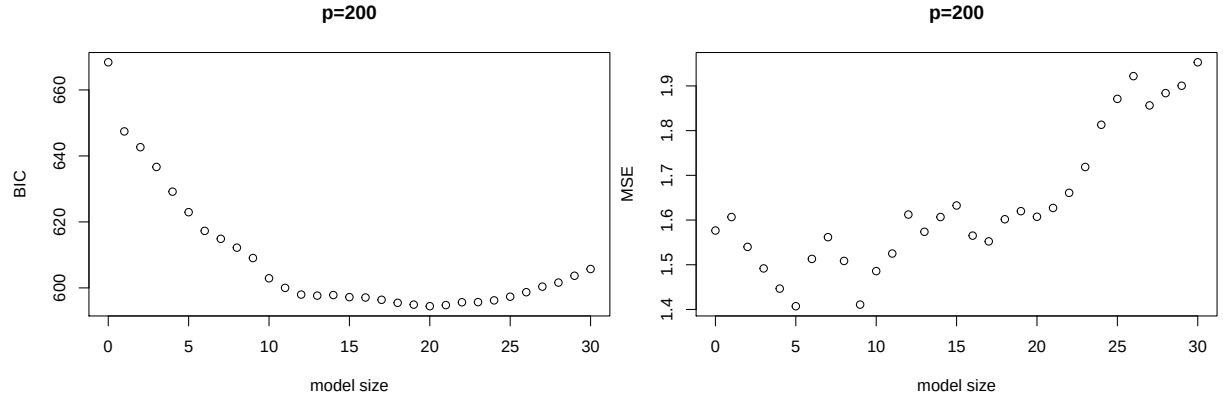
```
    X <- apply(X, 2, function(x)return(x/sqrt(sum(x^2))))
    beta <- matrix(c(rep(5,10), rep(0,p-10)), p, 1)
    e <- matrix(rnorm(n), n, 1)
    Y <- X%*%beta + e

    S <- 1:p
    S_in_order <- rep(0,30)
    BIC_list <- c(BIC(lm(Y~1)))

for(step in 1:30){
        BIC_list_temp <- c()
    for(i in 1:length(S)){
            S_temp <- S_in_order[1:step-1]
            S_temp[step] <- S[i]
            BIC_list_temp[i] <- BIC(lm(Y~X[,S_temp]))
        }
        add_ind <- S[which.min(BIC_list_temp)]
        S_in_order[step] <- add_ind
        S = setdiff(S, add_ind)
        BIC_list[step+1] <- min(BIC_list_temp)
    }

plot(0:30, BIC_list,
     xlab='model size', ylab='BIC', main=sprintf('p=%d',p))
}
```

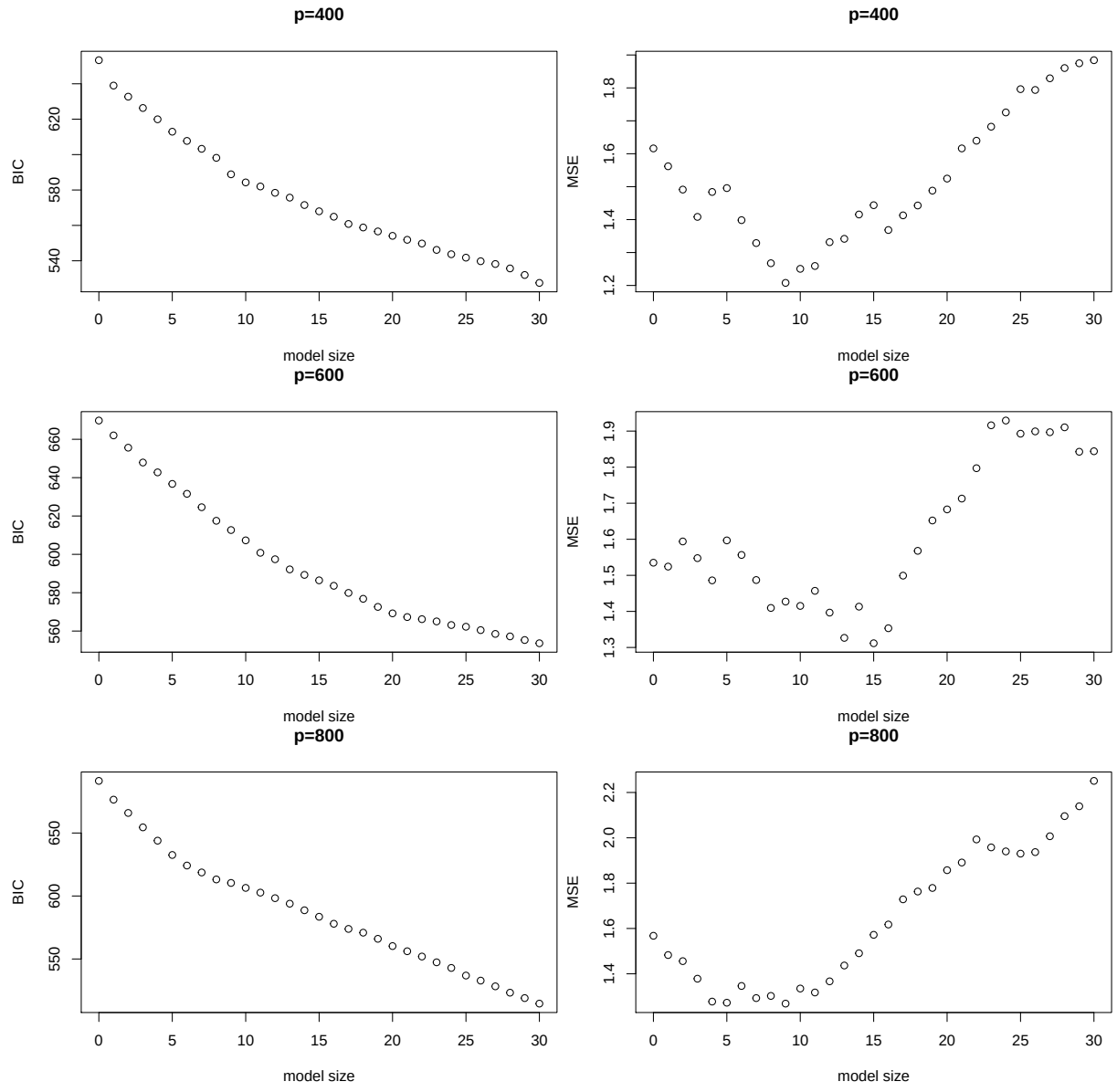
(b) Next, we'll do this again but we'll use a held-out validation set to test the model. Split your data at random into 200 training points and 200 validation points. Run forward stepwise on the 200 training points to obtain a model of size 0, a model of size 1, ..., a model of size 30. Then evaluate the prediction error of each model on the validation set (i.e. using the selected subset & fitted coefficients $\hat{\beta}$ from the training set). Plot prediction error against model size. What do you see? How do these results compare to BIC?

(1) When p is small. As model size increases, BIC first decreases and the increases. The prediction error is the same. Both of them serve as an indicator of model picking.

(2) When p is large. The prediction error has similar performance as when p is small, while the BIC fails to be an indicator of model picking.

BIC criterion tends select larger model due to multiple testing issues, while selection based on held out prediction error gives a much more correct result than the number of selected parameters is around 10.






```
n <- 400
for (p in c(1:4)*200){
    X <- matrix(rnorm(n*p), n, p)
    X <- apply(X, 2, function(x)return(x/sqrt(sum(x^2))))
    beta <- matrix(c(rep(5,10), rep(0,p-10)), p, 1)
    e <- matrix(rnorm(n), n, 1)
    Y <- X%*%beta + e

    test_ind <- sample(1:n, n/2)

    S <- 1:p
    S_in_order <- rep(0,30)
    BIC_list <- c(BIC(lm(Y[-test_ind]~1)))
    MSE_list <- c(var(lm(Y[-test_ind]~1)$residuals))

    for(step in 1:30){
```

```
        BIC_list_temp <- c()
        for(i in 1:length(S)){
            S_temp <- S_in_order[1:step-1]
            S_temp[step] <- S[i]
            BIC_list_temp[i] <- BIC(lm(Y[-test_ind]~X[-test_ind,S_temp]))
        }
        add_ind <- S[which.min(BIC_list_temp)]
        S_in_order[step] <- add_ind
        S = setdiff(S, add_ind)

        model <- lm(Y[-test_ind]~X[-test_ind,S_in_order])
        BIC_list[step+1] <- min(BIC_list_temp)
        MSE_list[step+1] <- var(Y[test_ind]-predict(model, data.frame(X[test_ind,S_in_order])))
    }

    plot(0:30, BIC_list,
         xlab='model size', ylab='BIC', main=sprintf('p=%d',p))
    plot(0:30, MSE_list,
         xlab='model size', ylab='MSE', main=sprintf('p=%d',p))
}
```

4. Consider the following iteratively reweighted least squares (IRLS) algorithm:

- **Solve least squares (weights $w_i = 1$) to get $\hat{\beta}$.**
- **Update the weights by setting $w_i = 1/\sqrt{|Y_i - X_i^\top \hat{\beta}|}$.**
- **Solve WLS to get a new $\hat{\beta}$.**
- **Iterate the last two steps until convergence**

(a) What is the M-estimator that this IRLS procedure is trying to solve? That is, the procedure above is designed to minimize

$$\sum_{i=1}^{n} l(Y_i - X_i^\top \beta)$$

for what loss function l?

Taking derivative with respect to β,

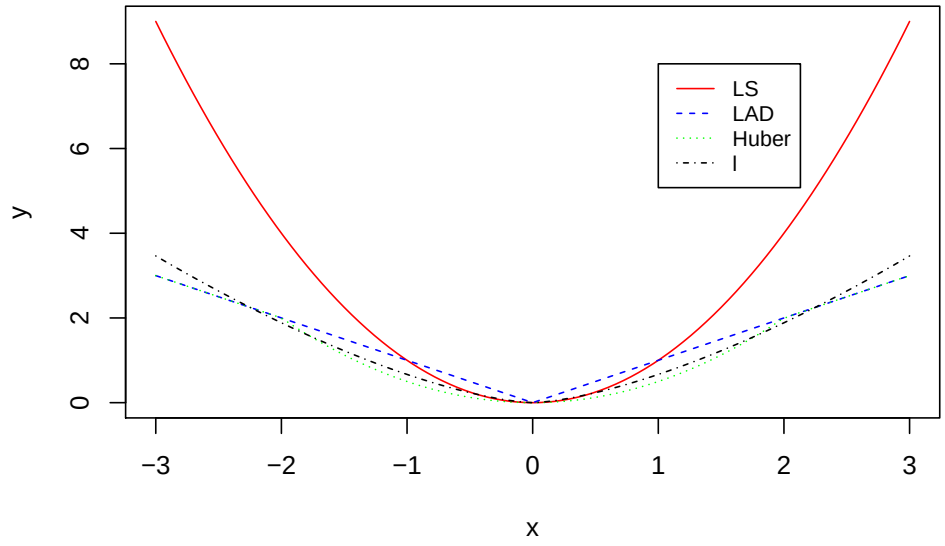
$$\sum_{i=1}^{n} l'(Y_i - X_i^\top \beta) X_i \xlongequal{u_i = Y_i - X_i^\top \beta} \sum_{i=1}^{n} \frac{l'(u_i)}{u_i}(Y_i - X_i^\top \beta) X_i$$
$$\xlongequal{w(u_i) = \frac{l'(u_i)}{u_i}} \sum_{i=1}^{n} w(u_i)(Y_i - X_i^\top \beta) X_i$$

Since $w(u_i) = \frac{1}{\sqrt{u_i}}$, we have $l(u_i) = \int w(u_i) u_i \mathrm{d}u_i = \int u_i^{\frac{1}{2}} \mathrm{d}u_i = \frac{2}{3} u_i^{\frac{3}{2}}$, i.e. $l(u) = \frac{2}{3}|u|^{\frac{3}{2}}$.

(b) Using your answer from above, how would this loss l compared to existing options, specifically least squares & LAD & Huber, in terms of robustness to outliers?

$$l_{LS}(u) = u^2$$
$$l_{LAD}(u) = |u|$$
$$l_{Huber}(u) = \begin{cases} |u| & , u > c \\ \frac{u^2}{c} & , u \le c \end{cases}$$

The robustness to outliers of l is better than l_{LS}, but worse than l_{LAD} and l_{Huber}.




```r
curve(x^2, from=-3, to=3, , xlab="x", ylab="y", col="red")
curve(abs(x), from=-3, to=3, add = TRUE, col = "blue", lty=2)
huber <- function(x,c=2){
    z<-x;
    z[abs(x)>c]<-abs(x[abs(x)>c]);
    z[abs(x)<=c]<-(x[abs(x)<=c])^2/c
    return(z)
}
curve(huber(x), from=-3, to=3, add = TRUE, col = "green", lty=3)
curve(abs(x)^(3/2)*2/3, from=-3, to=3, add = TRUE, col = "black", lty=4)
legend(1, 8, legend=c("LS", "LAD", "Huber", "l"),
       col=c("red", "blue", "green", "black"), lty=1:4, cex=0.8)
```