Lab 7 STA 206

Nov. 2019

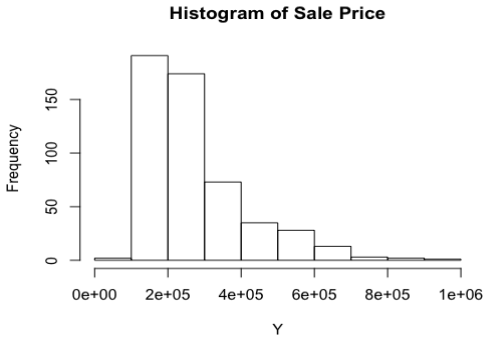
Model Selection in Multiple Regression

When we have a lot of predictor variables, instead of using them all, we would prefer choosing a few subsets of them that give us the best models based on a certain criterion. In this section, we will learn different methods of model selection and some commonly used selection criteria, AIC, BIC, Mallow_cp, adjusted R^2. For AIC and BIC, the smaller is the better. For adjusted R^2, the larger is the better. For Mallow_cp, the good model should be close or below the line $C_p = p$.
We will use the dataset in Appendix C.7(page 1353). The response variable is the residential home sales prices in a mid-western city. And the potential explanatory variables are some characteristics of the home and surrounding property. Six of these explanatory variables are quantitative and five of them are categorical.
A histogram of the response variable shows it is extremely skewed to the right, so we will take a log transformation on the home sale prices to make it more normally distributed. We will only use half of the data for model building and the other half for validation. Use the following commands to read in and explore and transform the data:

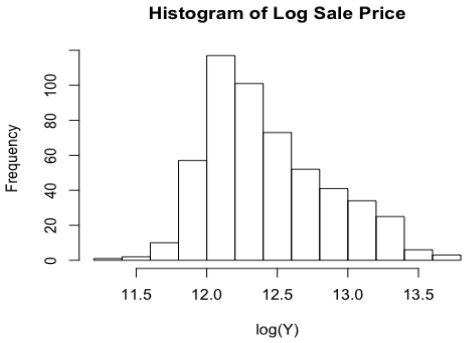
```
data = read.table('realestate.txt')
Y = data[, 2]
hist(Y, main="Histogram of Sale Price")
hist(log(Y), main="Histogram of Log Sale Price")
y = log(Y)


#categorical predictor variables
X_c = data[, c(6,8,10,11,13)]
colnames(X_c) = c("AC","pool","quality","style","hw")
#quantitative predictor variables
X_q = data[, c(3,4,5,7,9,12)]
colnames(X_q) = c("sqf","bedrm","bathrm","garage","year","lotsz")

#transformed dataset
d = cbind(y,X_q,X_c)
head(d)
```



(a)



(b)

```
#split data into two halves (training and validation)
set.seed(100)
n = nrow(d)/2
ind = sample(1:(2*n), n, replace=FALSE)
train = d[ind, ]  #training set
valid = d[-ind, ]  #validation set
```

Best Subsets Regression

For illustration purpose, we will only consider 4 quantitative X variables: Finished square feet, number of bedrooms, number of bathrooms, and garage size. Depending on inclusion and exclusion of these variables, we can construct $2^4 = 16$ models including the model with no predictor variable (intercept only, referred to as the none-model). We will use the function "regsubsets" in package "leaps" to get a summary for all 16 models.

```
install.packages(leaps) # install this package if you don't have it
library(leaps)
sub_set = regsubsets(y~sqf+bedrm+bathrm+garage, data=train, nbest=6,
nvmax=4, method="exhaustive")
sum_sub = summary(sub_set)

p.m = as.integer(rownames(sum_sub$which))+1
#number of coefficients in each model: p
ssto = sum((train$y-mean(train$y))^2)
sse = (1-sum_sub$rsq)*ssto
```

```
aic = n*log(sse/n)+2*p.m
bic = n*log(sse/n)+log(n)*p.m

res_sub = cbind(sum_sub$which, sse, sum_sub$rsq, sum_sub$adjr2,
sum_sub$cp, bic, aic)
```

In the "regsubsets" function, the option "nbest=6" tells R to find the best 6 models for each model size, and "nvmax=4" means each model can contain up to 4 X variables. Below is a summary of the output.

```
> sum_sub
Subset selection object
Call: regsubsets.formula(y ~ sqf + bedrm + bathrm + garage, data = train,
    nbest = 6, nvmax = 4, method = "exhaustive")
4 Variables  (and intercept)
       Forced in Forced out
sqf        FALSE      FALSE
bedrm      FALSE      FALSE
bathrm     FALSE      FALSE
garage     FALSE      FALSE
6 subsets of each size up to 4
Selection Algorithm: exhaustive
         sqf bedrm bathrm garage
1  ( 1 ) "*" " "   " "    " "
1  ( 2 ) " " " "   "*"    " "
1  ( 3 ) " " " "   " "    "*"
1  ( 4 ) " " "*"   " "    " "
2  ( 1 ) "*" " "   " "    "*"
2  ( 2 ) "*" " "   "*"    " "
2  ( 3 ) "*" "*"   " "    " "
2  ( 4 ) " " " "   "*"    "*"
2  ( 5 ) " " "*"   "*"    " "
2  ( 6 ) " " "*"   " "    "*"
3  ( 1 ) "*" " "   "*"    "*"
3  ( 2 ) "*" "*"   " "    "*"
3  ( 3 ) "*" "*"   "*"    " "
3  ( 4 ) " " "*"   "*"    "*"
4  ( 1 ) "*" "*"   "*"    "*"
```

The numbers in parenthesis are the ranks of the models based on SSE. The model

containing "finished square feet" is ranked number 1 (i.e., smallest SSE) among the models with only one predictor variable. The model containing "finished square feet" and "garage" is ranked number 1 among the models with two predictor variables. The "regsubsets" function does not fit the model with no X variable, so we will have to do it manually.

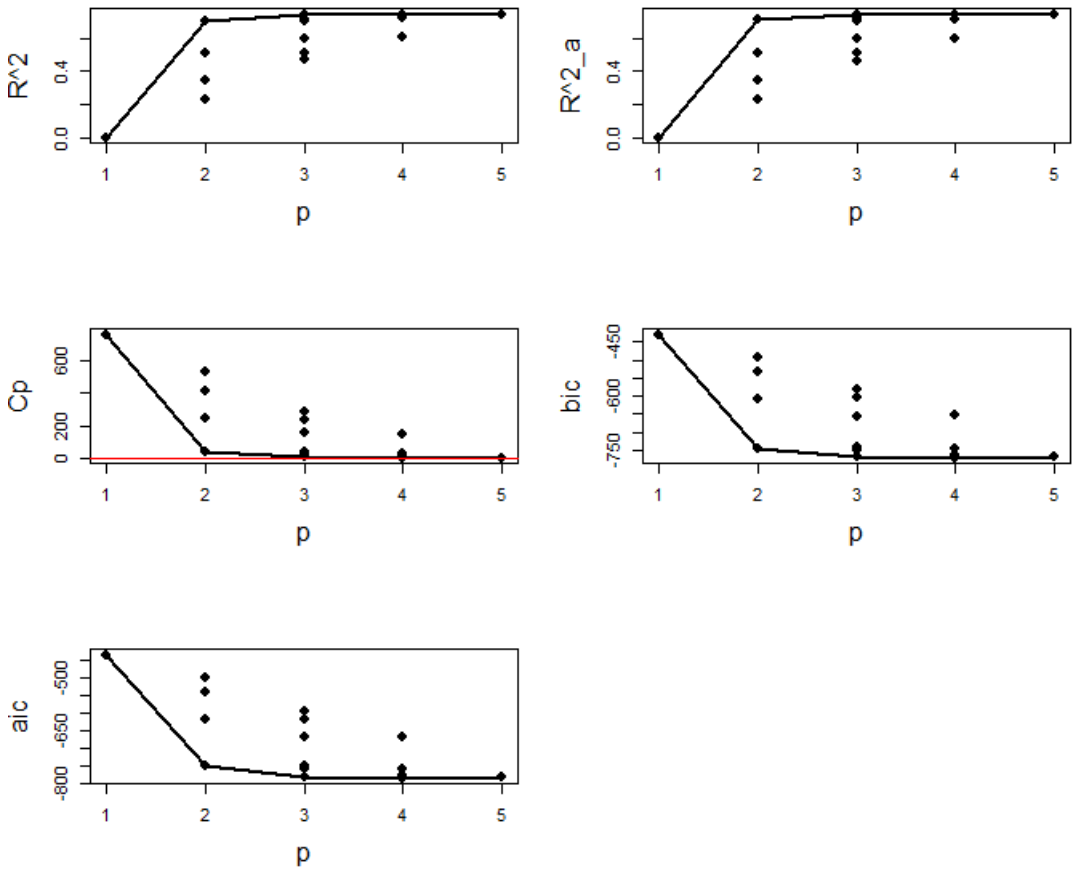
```
#model with intercept only
#need to calculate manually
fit1 = lm(y~1, data=train)
full = lm(y~sqf+bedrm+bathrm+garage, data=train)
sse1 = sum(fit1$residuals^2)
p = 1 #only one parameter
c1 = sse1/summary(full)$sigma^2 - (n-2*p)
aic1 = n*log(sse1/n)+2*p
bic1 = n*log(sse1/n)+log(n)*p
none = c(1,0,0,0,0,sse1,0,0,c1,bic1,aic1)  #model summary for intercept model

#combine the results
res_sub = rbind(none,res_sub)
colnames(res_sub)=c(colnames(sum_sub$which),"sse", "R^2", "R^2_a", "Cp",
"bic", "aic")
round(res_sub, 2)
```

```
> round(res_sub, 2)
     (Intercept) sqf bedrm bathrm garage   sse  R^2 R^2_a     Cp     bic     aic
none           1   0     0      0      0 49.36 0.00  0.00 756.77 -429.08 -432.65
1              1   1     0      0      0 14.39 0.71  0.71  39.12 -745.24 -752.36
1              1   0     0      1      0 24.20 0.51  0.51 240.88 -609.62 -616.75
1              1   0     0      0      1 32.29 0.35  0.34 407.44 -534.30 -541.43
1              1   0     1      0      0 38.06 0.23  0.23 526.16 -491.40 -498.53
2              1   1     0      0      1 12.79 0.74  0.74   8.28 -770.36 -781.05
2              1   1     0      1      0 13.88 0.72  0.72  30.52 -749.19 -759.88
2              1   1     1      0      0 14.39 0.71  0.71  41.03 -739.75 -750.45
2              1   0     0      1      1 19.83 0.60  0.60 153.14 -655.94 -666.63
2              1   0     1      1      0 24.01 0.51  0.51 239.07 -606.06 -616.76
2              1   0     1      0      1 26.18 0.47  0.47 283.70 -583.49 -594.19
3              1   1     0      1      1 12.44 0.75  0.75   3.02 -772.09 -786.35
3              1   1     1      0      1 12.77 0.74  0.74   9.69 -765.37 -779.63
3              1   1     1      1      0 13.85 0.72  0.72  31.90 -744.19 -758.45
```

```
3              1    0      1       1       1 19.53 0.60  0.60 148.92 -654.38 -668.64
4              1    1      1       1       1 12.44 0.75  0.74   5.00 -766.54 -784.37
```

We can see that the models are sorted (ranked) according to SSE within each model size.
We can option the plots of the criteria (R code omitted). A solid line connects the values of the best model for each model size.



For the C_p plot, good models should be nearby or below the red diagonal line $C_p = p$, indicating small model bias, and should have small C_p values, indicating a simple model (so less model variance).
One other criteria is the $Press_p$ using formula $Press_p = \sum_{i=1}^{n} \frac{(Y_i - \hat{Y}_i)^2}{(1-h_{ii})^2}$, where h_{ii} is the ith diagonal element of the hat matrix H (using all n cases). For example, we can get the $Press_p$ values for the null-model and the full model.

```
PRESS_none = sum((fit1$residuals/(1-influence(fit1)$hat))^2)
PRESS_full = sum((full$residuals/(1-influence(full)$hat))^2)
```

The function "influence(model)$hat" gives us the diagonal of the "hat matrix" of a regression model. Models with small $Press_p$ values are considered to have good predictive ability.

Stepwise Regression

We can also try to find the "best model" sequentially using stepwise regression procedures.

- Forward selection.
  Start from a simple model (normally the intercept only model), and add variables based on AIC or BIC (the variable that gives the biggest improvement (i.e. most decrease) in AIC/BIC is added at each step). Repeat the process until the AIC /BIC can not be improved by adding a variable.
- Backward elimination.
  Start from a full model and delete variables one by one based on a criterion (AIC or BIC). Repeat the process until AIC or BIC can not be improved anymore.
- Forward or Backward stepwise selection.
  At each step, evaluate all possible additions and deletions of a single variable, and choose the action that improves the criterion the most (it may be an addition or a deletion). Repeat until no improvement can be made.

```
#forward selection based on AIC
none_mod = lm(y~1, data=train)
full_mod = lm(y~sqf+bedrm+bathrm+garage+factor(AC)+factor(pool)
+factor(quality)+factor(style)+factor(hw), data=train)
library(MASS)
stepAIC(none_mod, scope=list(upper=full_mod), direction="forward", k=2)
#backward elimination based on AIC
stepAIC(full_mod, direction="backward", k=2)
#forward stepwise selection based on AIC
stepAIC(none_mod, scope=list(upper=full_mod), direction="both", k=2)

#selection based on BIC
stepAIC(none_mod, scope=list(upper=full_mod), direction="forward", k=log(n))
stepAIC(full_mod, direction="backward", k=log(n))
stepAIC(none_mod, scope=list(upper=full_mod), direction="both", k=log(n))
```

The "scope" option tells R to search the pool of models from the null model, but does not go beyond the full model.
Here direction="forward" means forward selection. Other options are direction="backward" (backward elimination) or direction="both" (forward or backward stepwise selection depending whether you start with a none-model or a full-model).
In the full model definition, instead we can also write `"y~."` , which means the model with all first order terms. If we wrote `"y~.^2"`, it means the model with all first order terms and all 2-way interactions. **Note: We should first transform the type of all the categorical variables into "factor".**
By default, the step function selects models based on AIC, if we add the option "k=log(n)", it will select models based on BIC.
If we want to always keep a couple variables, say X1 and X2, in all the models being considered, we need to specify "scope=list(lower= X1+X2)".
Below we show partial output of the forward stepwise selection results (based on BIC):

```
> stepAIC(none_mod, scope=list(upper=full_mod), direction="both", k=log(n))
Start:  AIC=-429.08
y ~ 1

                  Df Sum of Sq    RSS     AIC
+ sqf              1    34.972 14.391 -745.24
+ factor(quality)  2    32.026 17.336 -691.06
+ bathrm           1    25.168 24.195 -609.62
+ garage           1    17.074 32.289 -534.30
+ bedrm            1    11.304 38.058 -491.40
+ factor(style)    6    11.480 37.883 -464.79
+ factor(AC)       1     6.807 42.556 -462.25
+ factor(pool)     1     2.017 47.346 -434.41
<none>                         49.363 -429.08
+ factor(hw)       1     0.011 49.352 -423.58

Step:  AIC=-745.24
y ~ sqf

                  Df Sum of Sq    RSS     AIC
+ factor(quality)  2     4.272 10.119 -826.03
+ garage           1     1.596 12.794 -770.36
+ factor(AC)       1     0.855 13.535 -755.66
```

```
+ bathrm           1     0.515 13.875 -749.19
<none>                        14.391 -745.24
+ factor(hw)       1     0.062 14.328 -740.80
+ factor(pool)     1     0.030 14.361 -740.21
+ bedrm            1     0.005 14.386 -739.75
+ factor(style)    6     1.157 13.234 -733.72
- sqf              1    34.972 49.363 -429.08

Step:  AIC=-826.03
y ~ sqf + factor(quality)

                  Df Sum of Sq     RSS     AIC
+ garage           1    0.2741  9.8446 -827.63
<none>                         10.1187 -826.03
+ factor(AC)       1    0.1558  9.9629 -824.51
+ bathrm           1    0.0736 10.0452 -822.37
+ bedrm            1    0.0387 10.0800 -821.46
+ factor(hw)       1    0.0153 10.1034 -820.86
+ factor(pool)     1    0.0014 10.1173 -820.50
+ factor(style)    6    0.7190  9.3998 -811.88
- factor(quality)  2    4.2718 14.3905 -745.24
- sqf              1    7.2176 17.3364 -691.06

Step:  AIC=-827.63
y ~ sqf + factor(quality) + garage

                  Df Sum of Sq     RSS     AIC
<none>                         9.8446 -827.63
- garage           1    0.2741 10.1187 -826.03
+ factor(AC)       1    0.0964  9.7483 -824.63
+ bathrm           1    0.0599  9.7848 -823.66
+ bedrm            1    0.0462  9.7984 -823.29
+ factor(hw)       1    0.0113  9.8334 -822.36
+ factor(pool)     1    0.0001  9.8446 -822.07
+ factor(style)    6    0.6737  9.1710 -812.74
- factor(quality)  2    2.9497 12.7943 -770.36
- sqf              1    6.5887 16.4333 -699.46
```

```
Call:
lm(formula = y ~ sqf + factor(quality) + garage, data = train)

Coefficients:
     (Intercept)          sqf  factor(quality)2  factor(quality)3        garage
      11.9293886    0.0003155        -0.2942863        -0.5174687     0.0617522
```

Model Validation

The best model based on forward selection, backward elimination and forward or Backward stepwise selection with BIC is "Log(price) $\sim$ sqf + garage + factor(quality)". The validation data is used to re-run this model.

```
train1 = lm(y ~ sqf + garage + factor(quality), data = train)
valid1 = lm(y ~ sqf + garage + factor(quality), data = valid)

mod_sum = cbind(coef(summary(train1))[,1], coef(summary(valid1))[,1],
coef(summary(train1))[,2], coef(summary(valid1))[,2])
colnames(mod_sum) = c("Train Est","Valid Est","Train s.e.","Valid s.e.")
```

And we have the following comparison:

```
> mod_sum
                     Train Est     Valid Est   Train s.e.   Valid s.e.
(Intercept)      11.9293885569 11.9220216825 1.022414e-01 9.428707e-02
sqf               0.0003154758  0.0002980272 2.410161e-05 2.544639e-05
garage            0.0617522060  0.1006410044 2.312996e-02 2.282240e-02
factor(quality)2 -0.2942863333 -0.3811186850 4.630435e-02 4.559969e-02
factor(quality)3 -0.5174686868 -0.5498294372 6.027920e-02 5.701472e-02
```

Most of the estimated coefficients as well as their standard errors agree quite closely on the two data sets, which implies the consistency in parameter estimation.
We can also examine the SSE and adjusted R squares using both the training data and validation data.

```
sse_t = sum(train1$residuals^2)
sse_v = sum(valid1$residuals^2)
Radj_t = summary(train1)$adj.r.squared
Radj_v = summary(valid1)$adj.r.squared
```

```
train_sum = c(sse_t,Radj_t)
valid_sum = c(sse_v,Radj_v)
criteria = rbind(train_sum,valid_sum)
colnames(criteria) = c("SSE","R2_adj")
```

```
> criteria
               SSE    R2_adj
train_sum 9.844643 0.7974493
valid_sum 9.332345 0.8000322
```

The SSE and adjusted R squares are also very close.

Now what we'd like to do is find the SSE/n under the training model, and compare it to the $MSPE_v$ when we apply our training model to the validation data. The idea here is that if SSE/n is close to $MSPE_v$, then there is no severe overfitting by the model. If $MSPE_v$ is much larger than SSE/n, then there is some evidence of overfitting with the training data, and so we should use $MSPE_v$ as an indicator of the predictive ability of the model.

Now what we have to do is to use the estimated coefficients from the training fit, and apply them to get fitted values for the validation set.

```
#Get MSPE_v from new data
newdata = valid[, -1]
y.hat = predict(train1, newdata)
```

Now that we have the fitted values for the validation set (using the estimated coefficients from the training set), we can find $MSPE_v$. We have that

$$MSPE_v = \frac{\sum_{j=1}^{m}(Y_j - \hat{Y}_j)^2}{m},$$

Where Y_j is the j^{th} observation from the validation set, $\hat{Y}_j$ is the j^{th} fitted value, and m is the number of observations in the validation set. In R, we can find it this way:

```
MSPE = mean((valid$y - y.hat)^2)
MSPE
[1] 0.03765169
sse_t/n
[1] 0.03771894
```

We conclude that since $MSPE_v$ is close to SSE/n, there is no severe overfitting in the model.

Model Diagnostics

Outlying Y Observations

Now we fit a new model using only the variables sqf and garage. Once we have fit our model, we need to check if there are any outliers in our data. As discussed in lecture, Some (but not necessarily all) outlying cases may have an unduly strong influence on the fitted regression function.
Here we discuss how to find the studentized deleted residuals, which can be used to identify outlying Y observations. Deleted residuals are simply residuals found by deleting the i^{th} observation when calculating the i^{th} fitted value so that $d_i = Y_i - \hat{Y}_{i(i)}$. Note that this is what is used in defining Press_p. Studentized deleted residuals are then the deleted residuals divided by their standard errors:

$$t_i = \frac{d_i}{s\{d_i\}}.$$

In R, using the MASS package, we can find them using the function studres():

```
#studentized deleted residuals
fit = lm(y ~ sqf + garage, data=train)
stu.res.del = studres(fit)
head(sort(abs(stu.res.del), decreasing=TRUE))
     161      108      203      100       55       87
3.463841 3.042926 2.944603 2.868386 2.820887 2.774177
```

This code gives us the 6 largest studentized deleted residuals. We might be interested if these residuals exceed Bonferroni's threshold, which is at level α,

$$t(1 - \frac{\alpha}{2n}; n - p - 1).$$

In R this is

```
qt(1-.1/(2*n), n-3-1) #Bonferroni's Threshold (alpha=0.1, n=sample size, p=3)
[1] 3.599024
```

for $\alpha = 0.1$. In this case Bonferroni's threshold does not count any of the observations as outliers, but as stated in the lecture, Bonferroni tends to be conservative and lacks power.

Outlying X observations

Outlying X observations are identified through examining the diagonal elements h_{ii} of the hat matrix, a.k.a., leverage. A large value of h_{ii} indicates that the X values of the ith case is far away from the center of the X observations.
In practise, if $h_{ii} > \frac{2p}{n}$, then the ith case is identified as outlying with regard to its X values.
In R we write

```
h = as.vector(influence(fit)$hat)
p=3
index.X = which(h>(2*p/n))
index.X #17 outliers
[1]   1  11  17  25  50  54  68  93 104 115 160 174 210 225 240 245 250
```

In terms of X observations, the leverage method has detected 17 observations as outliers.

Cook's distance

Cook's distance measures the aggregate influence of the ith case on all n fitted values:

$$D_i := \frac{\sum_{j=1}^{n}(\widehat{Y}_j - \widehat{Y}_{j(i)})^2}{p \times MSE} = \frac{e_i^2}{p \times MSE}\frac{h_{ii}}{(1-h_{ii})^2} = \frac{r_i^2}{p}\frac{h_{ii}}{(1-h_{ii})}.$$

Here $\widehat{Y}_j$ is the fitted value for the jth case when all cases are used in the fitting and $\widehat{Y}_{j(i)}$ is the fitted value for the jth case when the ith case is excluded in the fitting.
When interpreting Cook's distance, we compare it with $\frac{4}{n-p}$. In practice, $D_i > \frac{4}{n-p}$ is often used as an indicator for being a potential influential case.
The R code is given below:

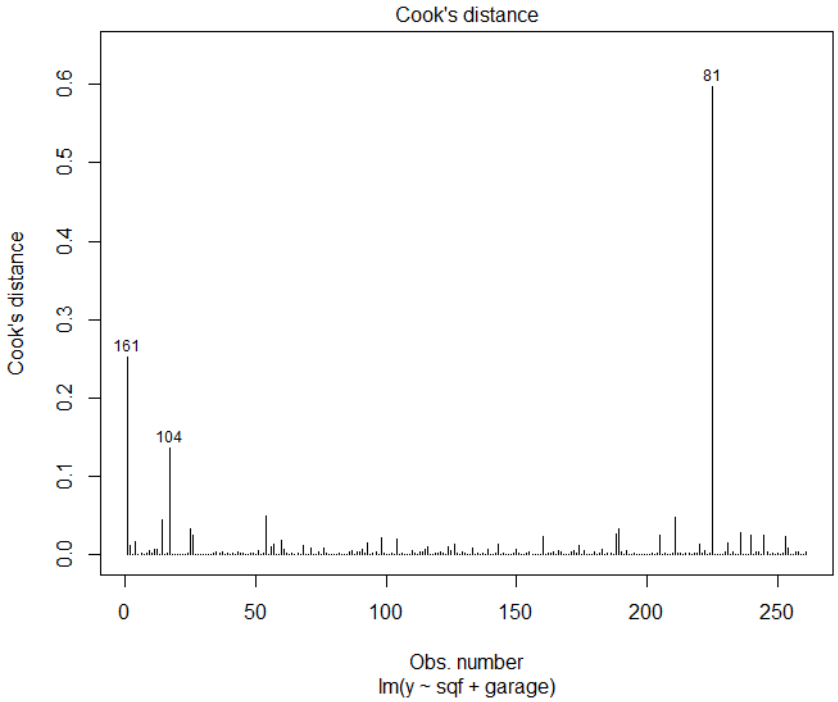
```
res = fit$residuals
mse = anova(fit)["Residuals", 3]
cook.d = res^2*h/(p*mse*(1-h)^2)

sort(cook.d[index.X], decreasing = TRUE)
sort(cook.d[index.X], decreasing = TRUE) > 4/(n-p)
```

We check the Cook's distance for the 17 outliers detected by leverage. We can see that the Cook's distances of the cases 81, 161, 104, 129, 210, 78, 219, 254 and 152 have the values larger than $\frac{4}{n-p}$, indicating that they may potentially have some

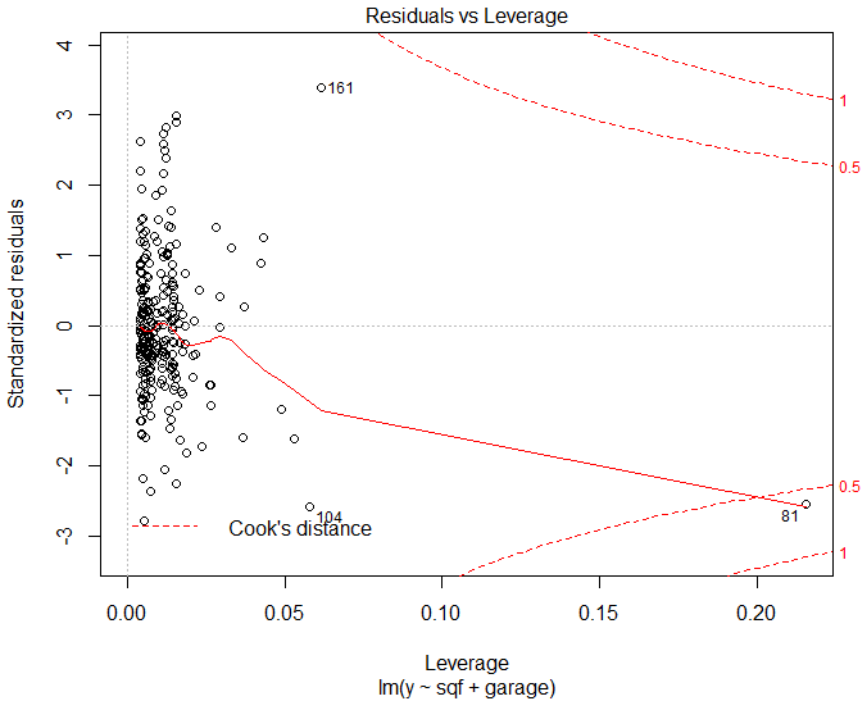
influence on the fitted values.
We can also check the Cook's distance by the Cook's distance plot:

```
plot(fit, which=4)
```



Moreover, we can check the Residuals vs. Leverage plot:

```
plot(fit, which=5)
```



Here we have the studentized residuals plotted against their leverage, which are the respective diagonal elements h_{ii} of the hat matrix. The leverage is a measure of how far away the X values of a case is from the center of the X values. It can be used to identify cases that are outlying in X values. If for a case both residual and leverage are large, i.e., the case tends to be outlying in both Y and X, then it could be an influential outlier which requires further investigation. The contour for a measure of influence (Cook's distance) is also provided in this plot. In these plots, the potential outliers also have a number next to them- this is their case index. This way, we can look at the plots and determine which cases are influential outliers.