[HW5]

[Arjun Bhan]

[4-21-2021]

```r
library(MASS)
?Boston
```

```
## starting httpd help server ... done
```

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

Exercise 1:

```r
#3th degree polynomial
lm.fit3=lm(medv ~ poly(lstat, degree=3), data=Boston)
summary(lm.fit3)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, degree = 3), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5441  -3.7122  -0.5145   2.4846  26.4153
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               22.5328     0.2399  93.937  < 2e-16 ***
## poly(lstat, degree = 3)1 -152.4595     5.3958 -28.255  < 2e-16 ***
## poly(lstat, degree = 3)2   64.2272     5.3958  11.903  < 2e-16 ***
## poly(lstat, degree = 3)3  -27.0511     5.3958  -5.013 7.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.396 on 502 degrees of freedom
## Multiple R-squared:  0.6578, Adjusted R-squared:  0.6558
## F-statistic: 321.7 on 3 and 502 DF,  p-value: < 2.2e-16
```

```
#For the 3th degree polynomial we reject the null hypothesis that H0: beta_j=0 for j=1,2,3 becau
se all pvalues are less than 5%

#4th degree polynomial
lm.fit4=lm(medv ~ poly(lstat, degree=4), data=Boston)
summary(lm.fit4)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, degree = 4), data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.563  -3.180  -0.632   2.283  27.181
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               22.5328     0.2347  95.995  < 2e-16 ***
## poly(lstat, degree = 4)1 -152.4595     5.2801 -28.874  < 2e-16 ***
## poly(lstat, degree = 4)2   64.2272     5.2801  12.164  < 2e-16 ***
## poly(lstat, degree = 4)3  -27.0511     5.2801  -5.123 4.29e-07 ***
## poly(lstat, degree = 4)4   25.4517     5.2801   4.820 1.90e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.28 on 501 degrees of freedom
## Multiple R-squared:  0.673,  Adjusted R-squared:  0.6704
## F-statistic: 257.8 on 4 and 501 DF,  p-value: < 2.2e-16
```

```
#For the 4th degree polynomial we reject the null hypothesis that H0: beta_j=0 for j=1,2,3,4 bec
ause all pvalues are less than 5%

#6th degree polynomial
lm.fit6=lm(medv ~ poly(lstat, degree=6),data=Boston)
summary(lm.fit6)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, degree = 6), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.7317  -3.1571  -0.6941   2.0756  26.8994
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               22.5328     0.2317  97.252  < 2e-16 ***
## poly(lstat, degree = 6)1 -152.4595     5.2119 -29.252  < 2e-16 ***
## poly(lstat, degree = 6)2   64.2272     5.2119  12.323  < 2e-16 ***
## poly(lstat, degree = 6)3  -27.0511     5.2119  -5.190 3.06e-07 ***
## poly(lstat, degree = 6)4   25.4517     5.2119   4.883 1.41e-06 ***
## poly(lstat, degree = 6)5  -19.2524     5.2119  -3.694 0.000245 ***
## poly(lstat, degree = 6)6    6.5088     5.2119   1.249 0.212313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.212 on 499 degrees of freedom
## Multiple R-squared:  0.6827, Adjusted R-squared:  0.6789
## F-statistic: 178.9 on 6 and 499 DF,  p-value: < 2.2e-16
```

```
#For the 6th degree polynomial we reject the H0: that beta_j=0 for j=1,2,3,4,5 because all pvalu
es are less than 5%. We fail to reject j=6 because the pvalue is 0.212313 which is greater than
 .05.
```

Exercise 2:

```
set.seed(9)
num_obs = nrow(Boston) #get the number of rows
train_index = sample(num_obs, size = trunc(0.80 * num_obs)) #get 80% random rows
train_data = Boston[train_index, ] #train data
test_data = Boston[-train_index, ] #test data: all except every thing in the train data

#Root mean square error function
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
#Define the model complexity
get_complexity = function(model) {
  length(coef(model)) - 1 #return the number of variables (predictors)
}
#Get the RMSE for a model
get_rmse = function(model, data, response) {
  rmse(actual = data[, response],
       predicted = predict(model, data))
}
lm.fit1=lm(medv~lstat, data=train_data)
lm.fit2=lm(medv~lstat+I(lstat^2), data=train_data)
lm.fit3=lm(medv~lstat+I(lstat^2)+I(lstat^3), data=train_data)
lm.fit4=lm(medv~lstat+I(lstat^2)+I(lstat^3)+I(lstat^4), data=train_data)
lm.fit5=lm(medv~lstat+I(lstat^2)+I(lstat^3)+I(lstat^4)+I(lstat^5), data=train_data)
lm.fit6=lm(medv~lstat+I(lstat^2)+I(lstat^3)+I(lstat^4)+I(lstat^5)+I(lstat^6), data=train_data)
model_list = list(lm.fit1, lm.fit2, lm.fit3, lm.fit4, lm.fit5, lm.fit6)

#get the train error from the list of models
train_rmse = sapply(model_list, get_rmse, data = train_data, response = "medv")

#get the test error from the list of models
test_rmse = sapply(model_list, get_rmse, data = test_data, response = "medv")

#get the model complexity
model_complexity = sapply(model_list, get_complexity)

#Find the R^2 values for all models
R.Square=c(summary(lm.fit1)$r.squared,
           summary(lm.fit2)$r.squared,
           summary(lm.fit3)$r.squared,
           summary(lm.fit4)$r.squared,
           summary(lm.fit5)$r.squared,
           summary(lm.fit6)$r.squared)
```

```
#Create a data frame to list models, train error, test error, R^2 value, number of variables
data.frame(Model=c("fit_1", "fit_2", "fit_3", "fit_4", "fit_5","fit_6"),
           Train_RMSE=train_rmse,
           Test_RMSE=test_rmse,
           R_squared=R.Square,
           Number_predictors=model_complexity
            )
```

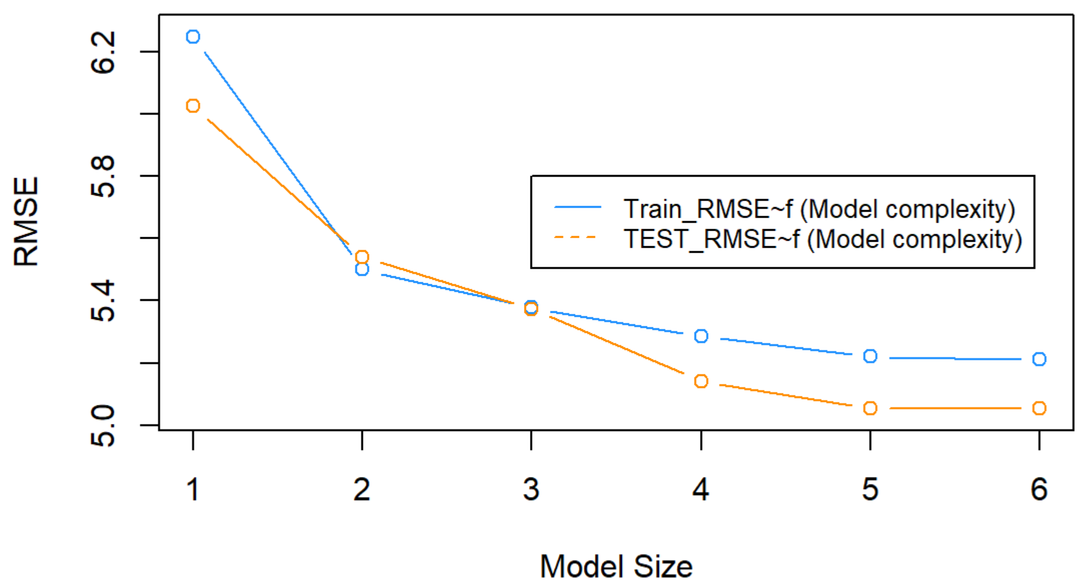
```
##   Model Train_RMSE Test_RMSE R_squared Number_predictors
## 1 fit_1   6.248410  6.025851 0.5397676                 1
## 2 fit_2   5.499806  5.540612 0.6434397                 2
## 3 fit_3   5.377728  5.372651 0.6590930                 3
## 4 fit_4   5.286163  5.141826 0.6706033                 4
## 5 fit_5   5.220380  5.054113 0.6787506                 5
## 6 fit_6   5.210863  5.055352 0.6799207                 6
```

```
#Plot train rmse as a function of model complexity
plot(model_complexity, train_rmse, type = "b",
     ylim = c(min(c(train_rmse, test_rmse)) - 0.02,
              max(c(train_rmse, test_rmse)) + 0.02),
     col = "dodgerblue",
     xlab = "Model Size",
     ylab = "RMSE")

#add test rmse as a function of model complexity into the previous graph

lines(model_complexity, test_rmse, type = "b", col = "darkorange")

#add legend into the graph
legend(3, 5.8,
       legend=c("Train_RMSE~f (Model complexity)", "TEST_RMSE~f (Model complexity)"),
       col=c("dodgerblue", "darkorange"),
       lty=1:2, cex=0.8)
```




```
#fit_5 has the smallest test error and therefore the best model.
```

Exercise 3:

```
mod1=lm(crim~zn+indus+chas+nox+rm+age+dis+rad+tax+ptratio+black+lstat+medv, data=Boston)
summary(mod1)
```

```
##
## Call:
## lm(formula = crim ~ zn + indus + chas + nox + rm + age + dis +
##     rad + tax + ptratio + black + lstat + medv, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.924 -2.120 -0.353  1.019 75.051
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.033228   7.234903   2.354 0.018949 *
## zn            0.044855   0.018734   2.394 0.017025 *
## indus        -0.063855   0.083407  -0.766 0.444294
## chas         -0.749134   1.180147  -0.635 0.525867
## nox         -10.313535   5.275536  -1.955 0.051152 .
## rm            0.430131   0.612830   0.702 0.483089
## age           0.001452   0.017925   0.081 0.935488
## dis          -0.987176   0.281817  -3.503 0.000502 ***
## rad           0.588209   0.088049   6.680 6.46e-11 ***
## tax          -0.003780   0.005156  -0.733 0.463793
## ptratio      -0.271081   0.186450  -1.454 0.146611
## black        -0.007538   0.003673  -2.052 0.040702 *
## lstat         0.126211   0.075725   1.667 0.096208 .
## medv         -0.198887   0.060516  -3.287 0.001087 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.439 on 492 degrees of freedom
## Multiple R-squared:  0.454,  Adjusted R-squared:  0.4396
## F-statistic: 31.47 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
#For the linear regression we reject the H0:βi=0 for zn,dis,rad, black and medv because the pval
ue is less than .05. We fail to reject the null hypothesis for indus, chas,nox,rm,age,tax,ptrati
o and lstat because the pvalue is greater than .05.
```

Exercise 4:

```r
set.seed(2000)
num_obs = nrow(Boston) #get the number of rows
train_index = sample(num_obs, size = trunc(0.80 * num_obs)) #get 80% random rows
train_data = Boston[train_index, ] #train data
test_data = Boston[-train_index, ] #test data: all except every thing in the train data

#Root mean square error function
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
#Define the model complexity
get_complexity = function(model) {
  length(coef(model)) - 1 #return the number of variables (predictors)
}
#Get the RMSE for a model
get_rmse = function(model, data, response) {
  rmse(actual = data[, response],
       predicted = predict(model, data))
}
mod1=lm(crim~rad+dis+zn+medv, data=train_data)
mod2=lm(crim~medv+I(medv^2)+I(medv^3), data=train_data)
mod3=lm(crim~medv*dis*tax, data=train_data)
model_list = list(mod1, mod2, mod3)


#get the train error from the list of models
train_rmse = sapply(model_list, get_rmse, data = train_data, response = "crim")

#get the test error from the list of models
test_rmse = sapply(model_list, get_rmse, data = test_data, response = "crim")

#get the model complexity
model_complexity = sapply(model_list, get_complexity)

#Find the R^2 values for all models
R.Square=c(summary(mod1)$r.squared,
           summary(mod2)$r.squared,
           summary(mod3)$r.squared)
```

```r
data.frame(Model=c("mod1", "mod2", "mod3"),
          Train_RMSE=train_rmse,
           Test_RMSE=test_rmse,
           R_squared=R.Square,
           Number_predictors=model_complexity
            )
```

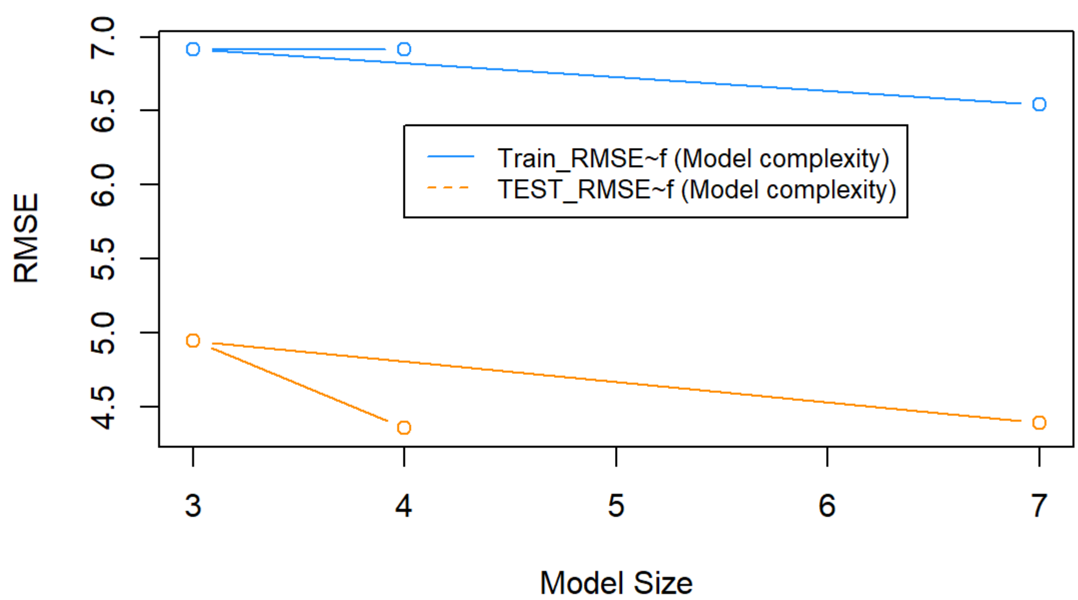
```
##   Model Train_RMSE Test_RMSE R_squared Number_predictors
## 1  mod1   6.912684  4.357042 0.4154678                 4
## 2  mod2   6.914092  4.948434 0.4152298                 3
## 3  mod3   6.544526  4.394230 0.4760722                 7
```

```
#Plot train rmse as a function of model complexity
plot(model_complexity, train_rmse, type = "b",
     ylim = c(min(c(train_rmse, test_rmse)) - 0.02,
              max(c(train_rmse, test_rmse)) + 0.02),
     col = "dodgerblue",
     xlab = "Model Size",
     ylab = "RMSE")

#add test rmse as a function of model complexity into the previous graph

lines(model_complexity, test_rmse, type = "b", col = "darkorange")

#add legend into the graph
legend(4, 6.4,
       legend=c("Train_RMSE~f (Model complexity)", "TEST_RMSE~f (Model complexity)"),
       col=c("dodgerblue", "darkorange"),
       lty=1:2, cex=0.8)
```




```
#mod1 has lowest test error and therefore the best model.
```