Lab #2

Purpose: In this lab, we perform a linear regression analysis.

1. Load Data Lab2Data.csv to object Datlab2.

```
rm(list = ls())

Datlab2=read.csv(file.choose(),header=T)

names(Datlab2)

attach(Datlab2)

contrasts(x4)
```

There are four input variables ($x_1 \sim x_4$) and one response variable (y) in this data. Variable x_4 is a categorical variable with two possible values. We should create a dummy variable that takes on two possible numerical values. The contrasts() function returns how R codes this dummy variable.

turn the categorical data => dummy variable

2. Multiple linear regression

We first fit the entire data with all input variables into a multiple linear regression model M1. We can use the lm() function.

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_0 + \epsilon$$

```
M1=lm(y~x1+x2+x3+x4,data= Datlab2)
```

We can use the function summary () to obtain the coefficients, their associated p-values, the R^2, and the F-Statistics.

```
summary(M1)
```

We can find that only one variable looks significant in this model. If we generate a residual plot by plotting the residuals with the predicted value, we could see that there is a strong pattern in the residuals. This indicates the existence of non-linearity. Thus, the current model may be misleading.

```
plot(predict(M1), residuals(M1))
```

We can plot the response variable to each predictor so as to determine which of them needs a non-linear transformation. Here we illustrate an example for y and x_1. We plot y and x_1 along with the least squares regression line using the plot()function. The plot suggests a polynomial relation between y and x_1. To examine the order of this polynomial relation, we use the function poly() within lm() to produce a fifth-order polynomial model M2. The result suggests the potential necessity of including a quadratic term in the model.

```
plot(x1,y,col="red")

M2=lm(y~poly(x1,5))
```

$\hat{y} = x_1 + x_1^2 + x_1^3 + x_1^4 + x_1^5$

x_1^2 is significant

```
summary(M2)
```

We can also check if the model should include an interaction effect. For example, to test an interaction effect between x_1 and x_2, we can include y~x1*x2 or y~x1:x2 in the lm() function. The syntax x1*x2 in the function lm() simultaneously includes x_1, x_2 and $x_1 * x_2$. That is, lm(y~x1*x2) fits the following model:

$\hat{y}$

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 (x_1 * x_2) + \beta_0 + \epsilon$$

```
M3=lm(y~x1*x2)
```

```
summary(M3)
```

The result suggests that the interaction effect is insignificant.

3. Hold-out, AIC, BIC, Adjusted R^2

Now we try different models and select a better one. We develop models using a training data set and assess the model performance using a test data set. We first use the sample() function to randomly split the original data set into one training set and one test set. The function nrow() counts the total number of rows in Datlab2. Sometimes we want to reproduce the exact same sampling results; we can use the set.seed() function. To use this function, you need to set a seed, which is an arbitrary integer, e.g. 1, 2, 3, 4 ....

We randomly select 80% of the observations for training (Datlab2.train) and the remaining in the test set (Datlab2.test). And the vector y.test stores the original value of y in the testing set.

```
set.seed(1)
```

```
train=sample(nrow(Datlab2),nrow(Datlab2)*0.8) #index for training data set
```

[1]
[9]
[17]
⋮
index we can pick up

with the index

```
Datlab2.train=Datlab2[train,] #training data set
```

from train

80%

```
Datlab2.test=Datlab2[-train,] #test data set
```

20%

vector with 40 observations

```
y.test=y[-train]  #the response in the test set
```

only have rows (only 1 column)

The first model of choice M4 contains all of the input variables and one quadratic term. The syntax I(x1^2) represent x_1 square. The second model M5 contains only two input variables. We learn the two models only using the training data set.

```
M4train=lm(y~I(x1^2)+x1+x2+x3+x4,data=Datlab2.train)
```

```
M5train =lm(y~I(x1^2)+x3+x4,data=Datlab2.train)
```

X_1, X_2 exclude

To compare the model performance, we calculate the MSE of each model using the testing data set. The function predict() predicts the value of the response variable for each observation in test data. y.predictM4 stores the predicted value of each observation in the test data set using model M4, and y.predictM5 stores the predicted value of each observation in the test data set using model M5. The mean() function here is to calculate the average prediction deviation, i.e. MSE, in the entire test data.

```
y.predictM4=predict(M4train,Datlab2.test)
```

where do u want to apply

```
M4MSE=mean((y.test-y.predictM4)^2)
```

```
y.predictM5=predict(M5train,Datlab2.test)
```

```
M5MSE=mean((y.test-y.predictM5)^2)
```

better cuz smaller.

In addition to using MSE, the model performance can be assessed using functions BIC(), AIC() and adjusted R^2 in summary().

prefer smaller value

```
AIC(M4train)
```

```
BIC(M4train)

summary(M4train)

AIC(M5train)

BIC(M5train)

summary(M5train)
```

Given the results from MSE, AIC, BIC, and adjusted R^2, the model M5 is a better one.

4. Cross-Validation

200/25 = 40 ×8 = 320

Now we illustrate how to compare the models using cross-validation. We use 5-fold cross validation along with MSE in this case. Thus, we first create a matrix to store the accuracy results associated with each fold for each model. We set the initial values for this matrix as zero.

set a vector

```
set.seed(1)

k=5

M4CVMSE=rep(0,k)

M5CVMSE=rep(0,k)
```

We use the sample() function to split the original data set into five folds.

```
folds=sample(1:k,nrow(Datlab2),replace=TRUE)
```

→ equal change to be picked up

How many we want to pick up (200)

In each round, we use four-fold to train the model, and one-fold to test the model. Then save testing MSE for M4 and M5 in M4CVMSE and M5CVMSE respectively.

folds expect fo the testing fold

```
for(j in 1:k)

{
    M4CV=lm(y~I(x1^2)+x1+x2+x3+x4,data=Datlab2[folds!=j,])
    M4CVMSE [j]=mean((y-predict(M4CV,Datlab2))[folds==j]^2)
}

for(j in 1:k)

{
    M5CV=lm(y~I(x1^2)+x3+x4,data=Datlab2[folds!=j,])
    M5CVMSE [j]=mean((y-predict(M5CV,Datlab2))[folds==j]^2)
}
```

Finally, calculate the cross-validation MSE for each model

```
MeanM4MSE=mean(M4CVMSE) ###CVMSE-M4###

MeanM5MSE=mean(M5CVMSE) ###CVMSE-M5###
```

smaller →

Consistent with the evaluation using hold-out, M5 is a better model.

5. Model Diagnostics

When the final model structure is decided, we can learn the final model coefficients using the entire data set.

```
M5=lm(y~I(x1^2)+x3+x4,data=Datlab2)
```

entire dataset

```
summary(M5)
```

coef() returns all coefficients values and confint() shows their associated confidence interval.

```
coef(M5)
```

```
confint(M5)
```

double - check

We generate some diagnostic plots for the model M5. Residual plots can be used to identify the existence of non-linearity. The function rstudent() returns the studentized residuals, which can be used to identity outliers. Given the diagnostics results, the model seems to fit the data pretty well.

```
plot(predict(M5), residuals(M5))
```

not any > 3
< 3

```
plot(predict(M5), rstudent(M5))
```

This concludes lab #2.