MAST90083_A1

QINGTAN

03/09/2022

Question 1

(1)

We load the dataset firstly:

```r
library(ISLR)
data("Hitters")
```

Then we delete NA rows, and we get the dimension of new_Hitters

```r
new_Hitters <- subset(Hitters, Hitters$Salary != "NA")
dim(new_Hitters)
```

```
## [1] 263  20
```

(2)

Firstly, we create the variable x and the variable y:

```r
x = model.matrix(Salary~., new_Hitters)[,-1]
y = new_Hitters$Salary
```

Then we get 100 random values:

```r
power_num <- seq(10, -2, length=100)
ridge_lambda <- 10^power_num
```

Then we use glmnet to get the ridge model and the coefficients for largest lambda and smallest lambda value:

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-3
```

```
ridge_model <- glmnet(x, y, alpha=0, lambda=ridge_lambda)
largest_lam <- coef(ridge_model)[, 1]
smallest_lam <- coef(ridge_model)[, 100]
```

Here is the coefficient values for the smallest lambda (10ˆ-2)

```
smallest_lam
```

```
##   (Intercept)        AtBat         Hits        HmRun         Runs
##  164.11321606  -1.97386151   7.37772270   3.93660219  -2.19873625
##           RBI        Walks        Years       CAtBat        CHits
##   -0.91623008   6.20037718  -3.71403424  -0.17510063   0.21132772
##        CHmRun        CRuns         CRBI       CWalks      LeagueN
##    0.05629004   1.36605490   0.70965516  -0.79582173  63.40493257
##     DivisionW      PutOuts      Assists       Errors   NewLeagueN
## -117.08243713   0.28202541   0.37318482  -3.42400281 -25.99081928
```

Here is the coefficient values for the smallest lambda (10ˆ10)

```
largest_lam
```

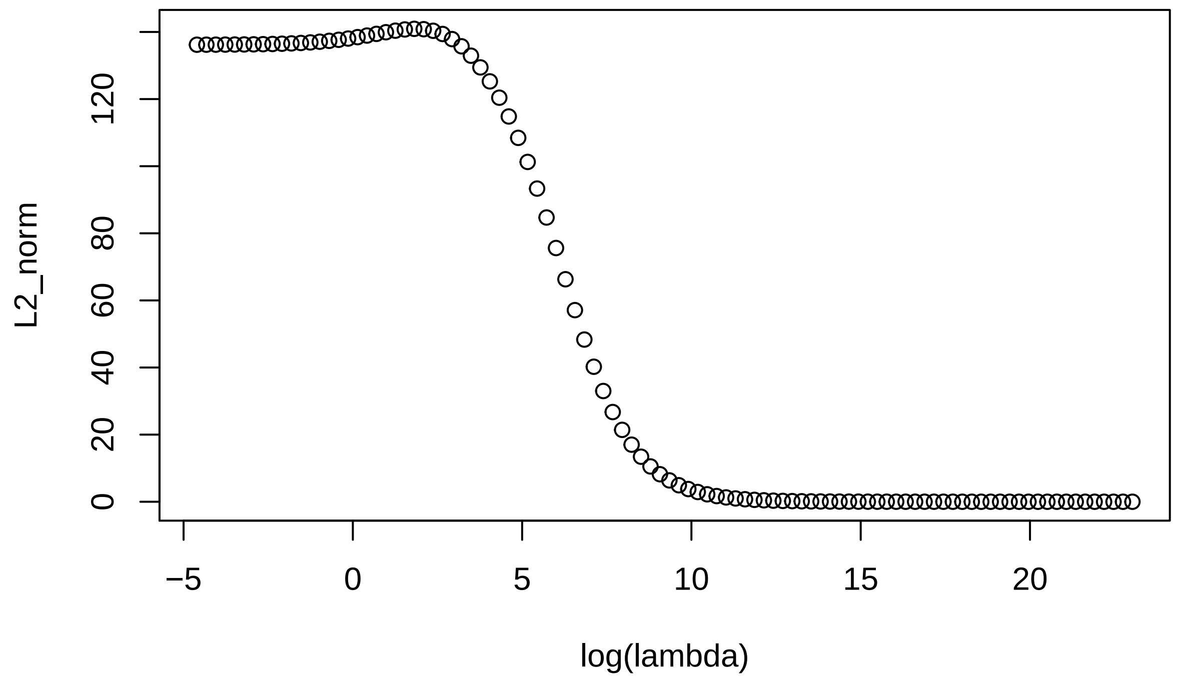
```
##   (Intercept)        AtBat         Hits        HmRun         Runs
##  5.359257e+02  5.443467e-08  1.974589e-07  7.956523e-07  3.339178e-07
##           RBI        Walks        Years       CAtBat        CHits
##  3.527222e-07  4.151323e-07  1.697711e-06  4.673743e-09  1.720071e-08
##        CHmRun        CRuns         CRBI       CWalks      LeagueN
##  1.297171e-07  3.450846e-08  3.561348e-08  3.767877e-08 -5.800263e-07
##     DivisionW      PutOuts      Assists       Errors   NewLeagueN
## -7.807263e-06  2.180288e-08  3.561198e-09 -1.660460e-08 -1.152288e-07
```

We can see that for the largest lambda, the coefficient values are more close to zero.

(3)

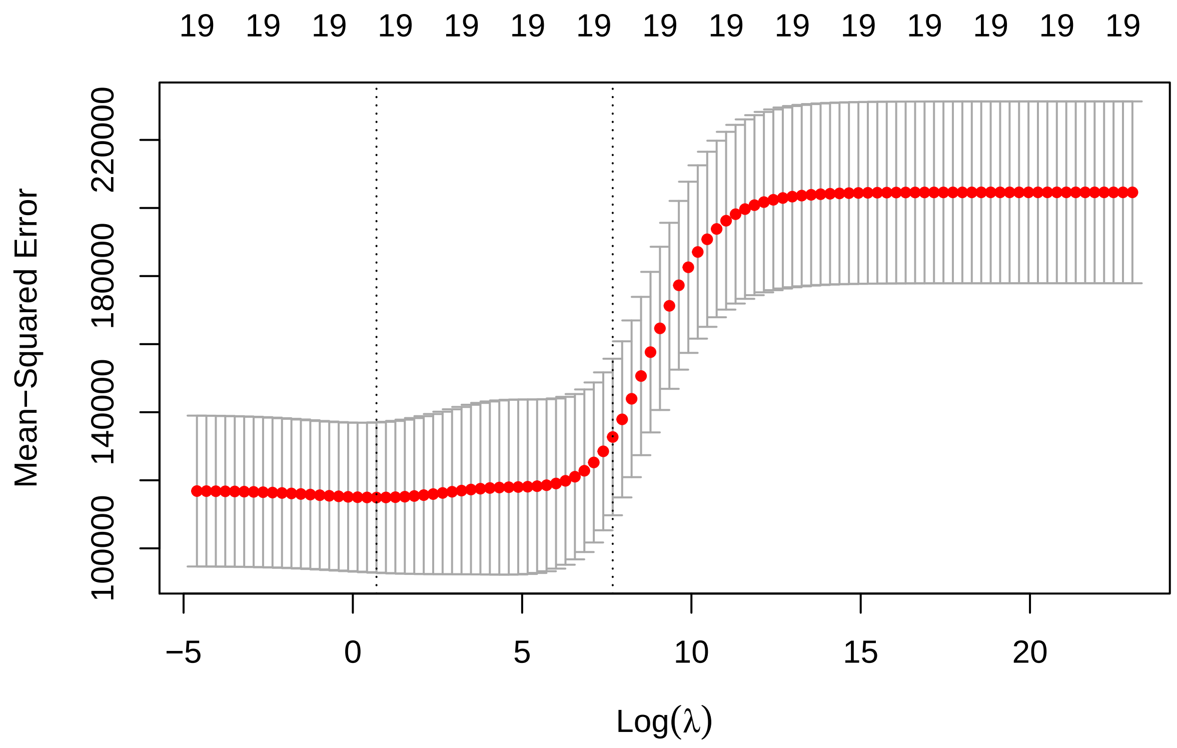
for l2_norm plot:

```
l2_norm = c(rep(0, 100))
for (i in 1:100){
  l2_norm[i] <- sqrt(sum(coef(ridge_model)[, i][-1]^2))
}
plot(log(ridge_lambda), l2_norm, xlab = 'log(lambda)', ylab = 'L2_norm')
```



for MSE plot, we shold use cv.glmnet to get the MSE value:

```
cv_ridge_model = cv.glmnet(x, y, alpha=0, lambda=ridge_lambda)
plot(cv_ridge_model)
```



Conclusion: From these two plots, we can see that from the L2_norm value, it is hard for us to find a smallest value, because values are all close to zero when log({lambda}) > 15, and it is much easier to find a smallest value in MSE plot, this value is located between 5 and 7.5. So it is more reasonable for us to get a good result by using Mean Square Error plot.

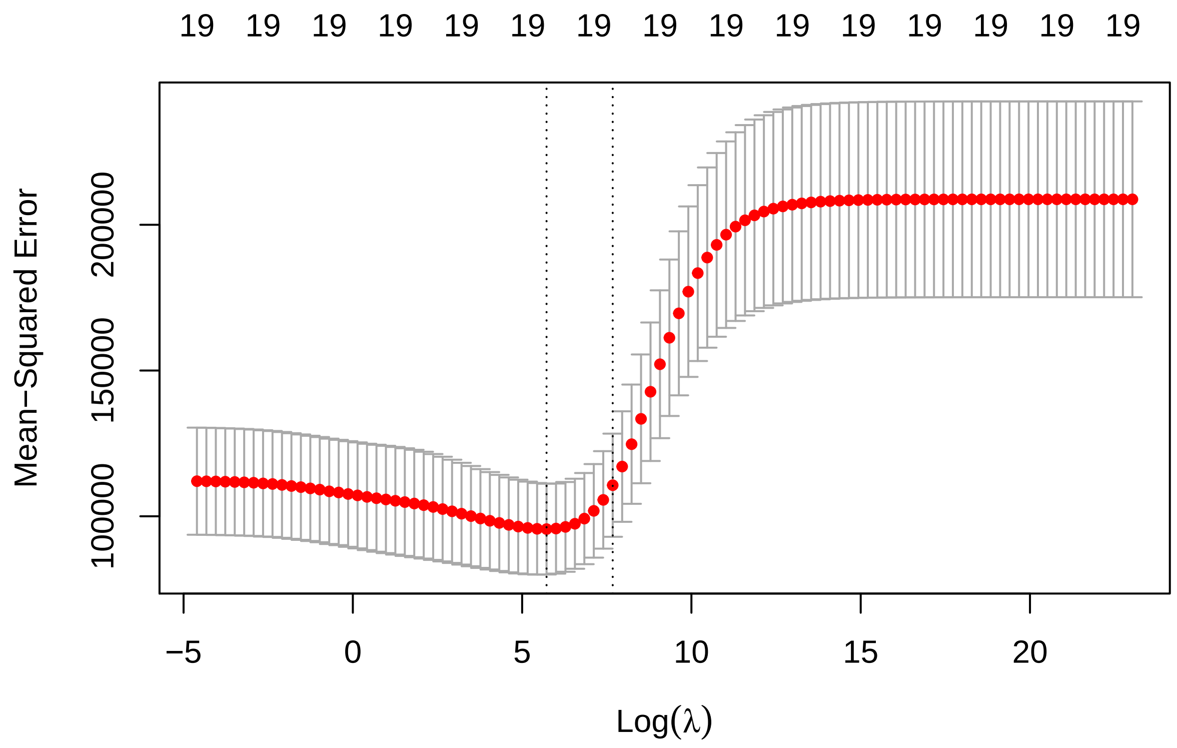
(4)

Firstly, we set seed and choose training set and testing set:

```r
set.seed(10)
sample_chosen = sample(1:nrow(x), 131)
training_set_1 = new_Hitters[sample_chosen, ]
testing_set_1 = new_Hitters[-sample_chosen, ]
training_set = model.matrix(Salary~., training_set_1)[,-1]
testing_set = model.matrix(Salary~., testing_set_1)[,-1]
training_target = training_set_1$Salary
testing_target = testing_set_1$Salary
```

Then we can use cv.glmnet to get the MSE plot:

```r
cv_ridge_model_1 =cv.glmnet(training_set,training_target,nfolds=10,alpha=0,lambda=ridge_lambda)
plot(cv_ridge_model_1)
```



Then we will find the best value for lambda:

```
min_mse_lambda = cv_ridge_model_1$lambda.min
min_mse_lambda
```

```
## [1] 305.3856
```

Then we use this best lambda to get MSE:

```
best_lam_model = glmnet(training_set, training_target, alpha=0, lambda=min_mse_lambda)
ridge_predict = predict(best_lam_model, newx=testing_set)
MSE_best = sum((ridge_predict - testing_target)^2) / length(testing_target)
MSE_best
```

```
## [1] 143219.1
```

So the best lambda is 305.3856, we get the value of MSE is 143219.1

Then we use the total dataset to check the coefficient values.

```
total_sets_model = glmnet(x, y, alpha=0, lambda = min_mse_lambda)
coef(total_sets_model)
```

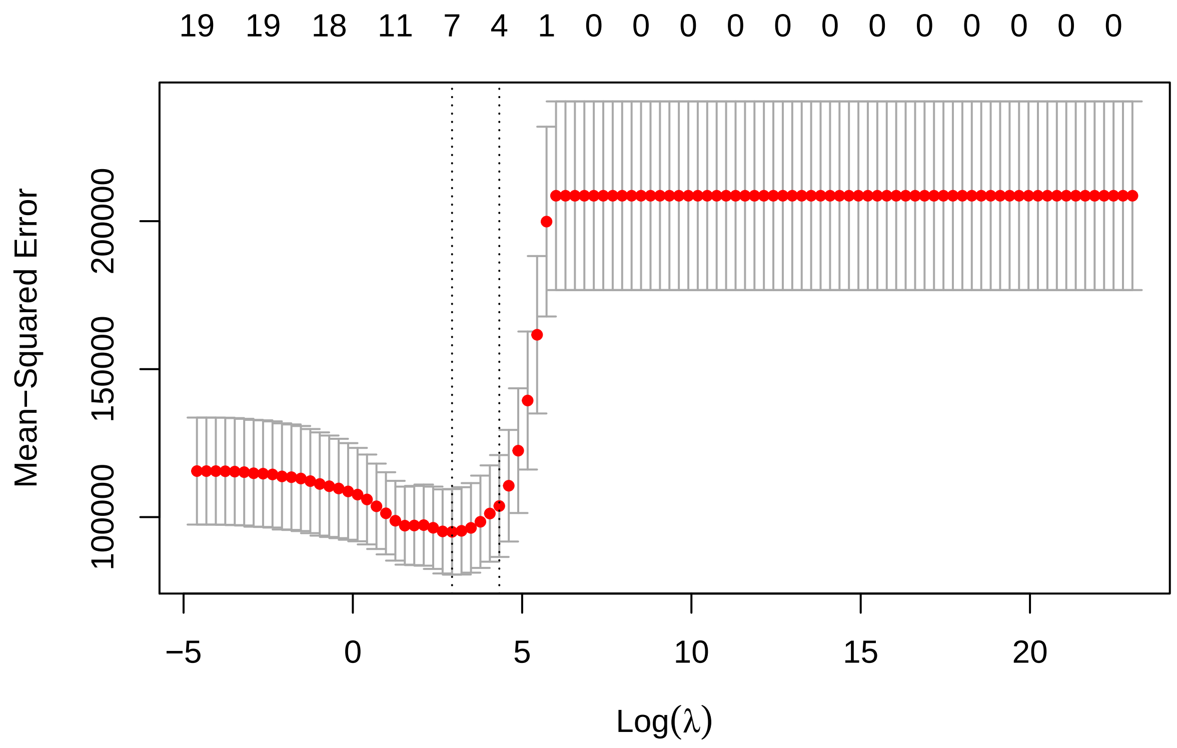
```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  13.82836042
## AtBat         0.07185690
## Hits          0.87923686
## HmRun         0.53787861
## Runs          1.07209934
## RBI           0.87940924
## Walks         1.65046005
## Years         1.20215837
## CAtBat        0.01135929
## CHits         0.05850049
## CHmRun        0.41307137
## CRuns         0.11642426
## CRBI          0.12329787
## CWalks        0.05014536
## LeagueN      22.83614180
## DivisionW   -81.02283953
## PutOuts       0.17012394
## Assists       0.03105466
## Errors       -1.42556422
## NewLeagueN    8.95856315
```

Conclusion: The coefficient values are like linear regression coefficient values, all of them are not close to zero, the number of variables didn't change.

(5)

Firstly, we get the MSE plot of lasso:

```
set.seed(10)
lasso_model = cv.glmnet(training_set, training_target, alpha=1, nfolds=10, lambda=ridge_lambda)
plot(lasso_model)
```



Then we find the best lambda for lasso:

```
min_lasso_lambda = lasso_model$lambda.min
min_lasso_lambda
```

```
## [1] 18.73817
```

Then we use this best lambda to get MSE value:

```
best_lasso_model = glmnet(training_set, training_target, alpha=1, lambda=min_lasso_lambda)
lasso_predict = predict(best_lasso_model, newx=testing_set)
MSE_lasso_best = sum((lasso_predict - testing_target)^2) / length(testing_target)
MSE_lasso_best
```

```
## [1] 142291.7
```

We can see that the best value for lambda is 18.73817, we get the smallest MSE, which is 142291.7

Then we use the total dataset to check the coefficient values.

```
total_lasso_model = glmnet(x, y, alpha=1, lambda = min_lasso_lambda)
coef(total_lasso_model)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)  24.6396990
## AtBat          .
## Hits          1.8499834
## HmRun          .
## Runs           .
## RBI            .
## Walks         2.1959762
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         .
## CRuns         0.2058514
## CRBI          0.4095910
## CWalks         .
## LeagueN        .
## DivisionW   -99.9733911
## PutOuts       0.2158910
## Assists        .
## Errors         .
## NewLeagueN     .
```

Conclusion: From the coefficient values, we can see that using lasso can ignore some useless variables, because most of them are zero, which is different from ridge regression.

Question 2

(1)

Write by hand.

(2)

Write by hand.

(3)

Write by hand.

(4)

Here we generate M1 nodes firstly:

```r
err_M1 <- rnorm(100, 0, 1)
M1 <- c()
for (i in 1:100){
  if (i <= 5){
    M1[i] <- 0
  }
  if (i > 5){
    M1[i] <- 0.434*M1[i-1]+0.217*M1[i-2]+0.145*M1[i-3]+0.108*M1[i-4]+0.087*M1[i-5]+err_M1[i]
  }
}
```

Then we generate M2 nodes:

```
err_M2 <- rnorm(100, 0, 1)
M2 <- c()
for (i in 1:100){
  if (i <= 2){
    M2[i] <- 0
  }
  if (i > 2){
    M2[i] <- 0.682*M2[i-1]+0.346*M2[i-2]+err_M2[i]
  }
}
```

(5)

Here we consider M1 firstly, we should use 'ar' function:

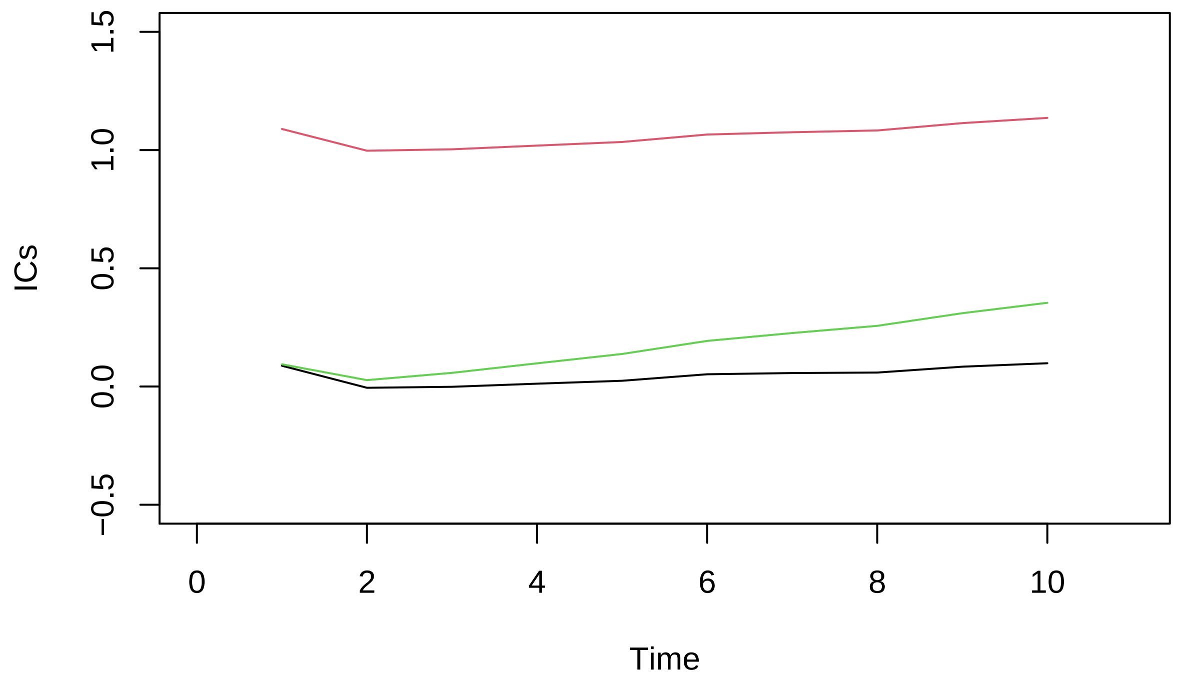
```
library(forecast)
```

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

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
M1_IC1 <- numeric(10)
M1_IC2 <- numeric(10)
M1_IC3 <- numeric(10)
for (i in 1:10){
  next_one = ar(M1, aic=F, i)
  T_num <- 100 - i
  res <- na.omit(M1 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  M1_IC1[i] = log(sig_2) + 2*(i+1)/T_num
  M1_IC2[i] = log(sig_2) + (T_num+i)/(T_num-i-2)
  M1_IC3[i] = log(sig_2) + log(T_num)*i/T_num
}

plot(-1, -1, xlim = c(0, 11), ylim = c(-0.5, 1.5), xlab = "Time",ylab = "ICs")
lines(M1_IC1,col=1)
lines(M1_IC2,col=2)
lines(M1_IC3,col=3)
```

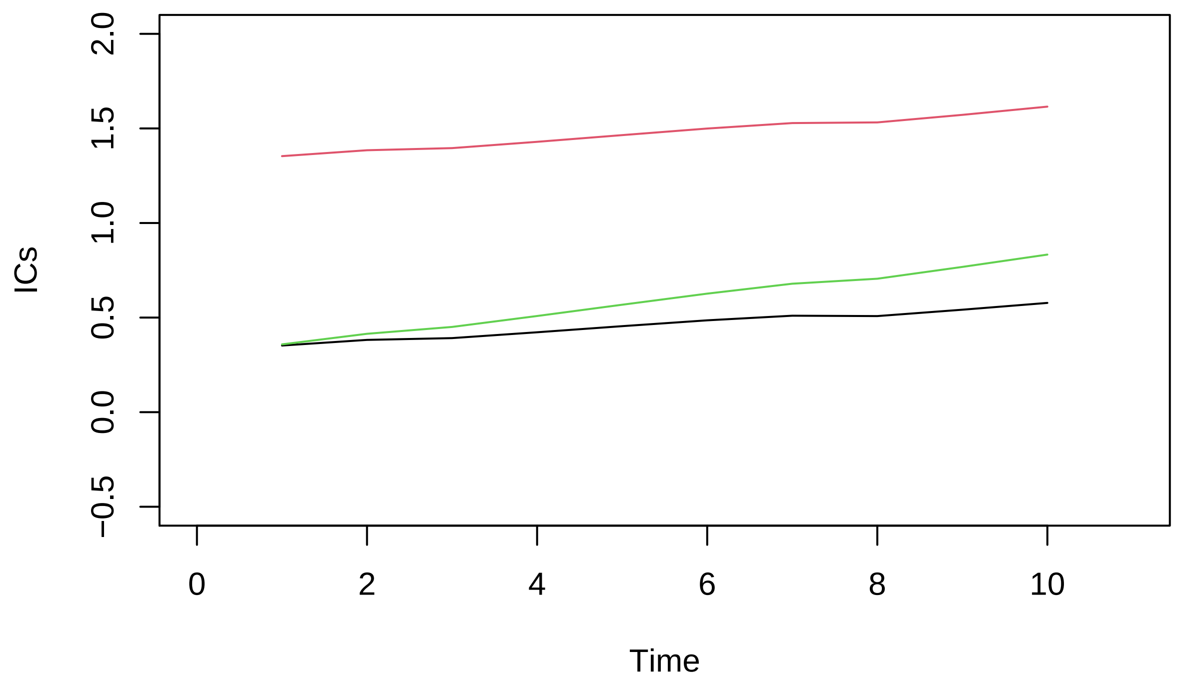


In this plot, black line is IC1, red line is IC2, green line is IC3.

Then we consider model M2:

```
M2_IC1 <- numeric(10)
M2_IC2 <- numeric(10)
M2_IC3 <- numeric(10)
for (i in 1:10){
  next_one = ar(M2, aic=F, i)
  T_num <- 100 - i
  res <- na.omit(M2 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  M2_IC1[i] = log(sig_2) + 2*(i+1)/T_num
  M2_IC2[i] = log(sig_2) + (T_num+i)/(T_num-i-2)
  M2_IC3[i] = log(sig_2) + log(T_num)*i/T_num
}

plot(-1,-1, xlim = c(0, 11), ylim = c(-0.5, 2), xlab = "Time",ylab = "ICs")
lines(M2_IC1,col=1)
lines(M2_IC2,col=2)
lines(M2_IC3,col=3)
```



In this plot, black line is IC1, red line is IC2, green line is IC3.

(6)

We consider model M1:

```
IC1_num <- c()
IC2_num <- c()
IC3_num <- c()

for (j in 1:1000){
  err_M1 <- rnorm(100, 0, 1)
M1 <- c()
for (i in 1:100){
  if (i <= 5){
    M1[i] <- 0
  }
  if (i > 5){
    M1[i] <- 0.434*M1[i-1]+0.217*M1[i-2]+0.145*M1[i-3]+0.108*M1[i-4]+0.087*M1[i-5]+err_M1[i]
  }
}
  M1_IC1 <- numeric(10)
M1_IC2 <- numeric(10)
M1_IC3 <- numeric(10)
for (k in 1:10){
  next_one = ar(M1, aic=F, k)
  T_num <- 100 - k
  res <- na.omit(M1 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  # p = i
  M1_IC1[k] = log(sig_2) + 2*(k+1)/T_num
  M1_IC2[k] = log(sig_2) + (T_num+k)/(T_num-k-2)
  M1_IC3[k] = log(sig_2) + log(T_num)*k/T_num
}
i1<- which(M1_IC1 == min(M1_IC1))
i2<- which(M1_IC2 == min(M1_IC2))
i3<- which(M1_IC3 == min(M1_IC3))
IC1_num[j] <- i1
IC2_num[j] <- i2
IC3_num[j] <- i3
}
```

Then we get a table for frequency of IC1 numbers:

```
table(IC1_num)
```

```
## IC1_num
##   1   2   3   4   5   6   7   8   9  10
##  43 326 368 126  32  32  22  18  17  16
```

Then we get a table for frequency of IC2 numbers:

```
table(IC2_num)
```

```
## IC2_num
##   1   2   3   4   5   6   7   8   9  10
##  46 353 386 122  31  26  11  11   7   7
```

Then we get a table for frequency of IC2 numbers:

```
table(IC3_num)
```

```
## IC3_num
##   1   2   3   4   5   6   7
## 157 562 244  31   2   1   3
```

(7)

Here we consider model1, change the samples from 100 to 15:

```
IC1_num <- c()
IC2_num <- c()
IC3_num <- c()

for (j in 1:1000){
  err_M1 <- rnorm(100, 0, 1)
M1 <- c()
for (i in 1:15){
  if (i <= 5){
    M1[i] <- 0
  }
  if (i > 5){
    M1[i] <- 0.434*M1[i-1]+0.217*M1[i-2]+0.145*M1[i-3]+0.108*M1[i-4]+0.087*M1[i-5]+err_M1[i]
  }
}
  M1_IC1 <- numeric(10)
M1_IC2 <- numeric(10)
M1_IC3 <- numeric(10)
for (k in 1:10){
  next_one = ar(M1, aic=F, k)
  T_num <- 15 - k
  res <- na.omit(M1 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  # p = i
  M1_IC1[k] = log(sig_2) + 2*(k+1)/T_num
  M1_IC2[k] = log(sig_2) + (T_num+k)/(T_num-k-2)
  M1_IC3[k] = log(sig_2) + log(T_num)*k/T_num
}
i1<- which(M1_IC1 == min(M1_IC1))
i2<- which(M1_IC2 == min(M1_IC2))
i3<- which(M1_IC3 == min(M1_IC3))
IC1_num[j] <- i1
IC2_num[j] <- i2
IC3_num[j] <- i3
}
```

Then we get a table for frequency of IC1 numbers:

```
table(IC1_num)
```

```
## IC1_num
##   1   2   3   4   6   7   8   9
## 865  83  22   6  12   5   3   4
```

Then we get a table for frequency of IC2 numbers:

```
table(IC2_num)
```

```
## IC2_num
##    7
## 1000
```

Then we get a table for frequency of IC3 numbers:

```
table(IC3_num)
```

```
## IC3_num
##   1   2   3   4   6   7   8   9  10
## 878  71  17   3  12   6   6   4   3
```

(8)

Firstly, we consider n=100 for M2:

```r
IC1_num <- numeric(1000)
IC2_num <- numeric(1000)
IC3_num <- numeric(1000)

for (j in 1:1000){
  err_M2 <- rnorm(100, 0, 1)
M2 <- c()
for (i in 1:100){
  if (i <= 2){
    M2[i] <- 0
  }
  if (i > 2){
    M2[i] <- 0.682*M2[i-1]+0.346*M2[i-2]+err_M2[i]
  }
}
M2_IC1 <- numeric(10)
M2_IC2 <- numeric(10)
M2_IC3 <- numeric(10)
for (k in 1:10){
  next_one = ar(M2, aic=F, k)
  T_num <- 100 - k
  res <- na.omit(M2 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  M2_IC1[k] = log(sig_2) + 2*(k+1)/T_num
  M2_IC2[k] = log(sig_2) + (T_num+k)/(T_num-k-2)
  M2_IC3[k] = log(sig_2) + log(T_num)*k/T_num
}
i1<- which(M2_IC1 == min(M2_IC1))
i2<- which(M2_IC2 == min(M2_IC2))
i3<- which(M2_IC3 == min(M2_IC3))
IC1_num[j] <- i1
IC2_num[j] <- i2
IC3_num[j] <- i3
}
```

Then we get a table for frequency of IC1 numbers:

```r
table(IC1_num)
```

```
## IC1_num
##   1   2   3   4   5   6   7   8   9
## 402 423  83  40  17  12  12   5   6
```

Then we get a table for frequency of IC2 numbers:

```r
table(IC2_num)
```

```
## IC2_num
##   1   2   3   4   5   6   7   8   9
## 414 439  79  31  14  10   8   3   2
```

Then we get a table for frequency of IC3 numbers:

```
table(IC3_num)
```

```
## IC3_num
##   1   2   3   4   5
## 602 375  20   2   1
```

Firstly, we consider n=15 for M2:

```
IC1_num <- numeric(1000)
IC2_num <- numeric(1000)
IC3_num <- numeric(1000)

for (j in 1:1000){
  err_M2 <- rnorm(100, 0, 1)
M2 <- c()
for (i in 1:15){
  if (i <= 2){
    M2[i] <- 0
  }
  if (i > 2){
    M2[i] <- 0.682*M2[i-1]+0.346*M2[i-2]+err_M2[i]
  }
}
M2_IC1 <- numeric(10)
M2_IC2 <- numeric(10)
M2_IC3 <- numeric(10)
for (k in 1:10){
  next_one = ar(M2, aic=F, k)
  T_num <- 15 - k
  res <- na.omit(M2 - fitted(next_one))
  sig_2 <- sum(res^2) / T_num
  M2_IC1[k] = log(sig_2) + 2*(k+1)/T_num
  M2_IC2[k] = log(sig_2) + (T_num+k)/(T_num-k-2)
  M2_IC3[k] = log(sig_2) + log(T_num)*k/T_num
}
i1<- which(M2_IC1 == min(M2_IC1))
i2<- which(M2_IC2 == min(M2_IC2))
i3<- which(M2_IC3 == min(M2_IC3))
IC1_num[j] <- i1
IC2_num[j] <- i2
IC3_num[j] <- i3
}
```

Then we get a table for frequency of IC3 numbers:

```
table(IC1_num)
```

```
## IC1_num
##   1   2   3   4   5   6   7   8   9
## 827  64  56  28  11   9   3   1   1
```

Then we get a table for frequency of IC3 numbers:

```
table(IC2_num)
```

```
## IC2_num
##    7
## 1000
```

Then we get a table for frequency of IC3 numbers:

```
table(IC3_num)
```

```
## IC3_num
##   1   2   3   4   5   6   7   8   9  10
## 863  52  33  22   9   8   5   3   3   2
```

(9)

For n=100 in M1 model, we can get the largest frequency numbers when p=2 or p=3 for IC1, IC2 and IC3. For n=100 in M2 model, we can get the largest frequency numbers when p=1 or p=3 for IC1, IC2 and IC3. For n=15 in M1 model, we can get the largest frequency numbers when p=1 for IC1 and IC3, but we can only get the largest frequency numbers when p=7 for IC2. For n=15 in M2 model, we can get the largest frequency numbers when p=1 for IC1 and IC3, but we can only get the largest frequency numbers when p=7 for IC2.

(10)

Write by hand.

(11)

Firstly, we need consider n=25. For M1, we know p=5, n=25, so for IC1:

```
n <- 25
p=5
IC1 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-p)/i)*exp(2*(n+1)*i/((n-p)*(n-p-i))) - (n-p-i)/i
  IC1[i] <- 1 - pf(f_num, i, n-p-i)
}
IC1
```

```
## [1] 6.199304e-02 2.877511e-02 1.256454e-02 5.237465e-03 2.074740e-03
## [6] 7.763016e-04 2.728411e-04 8.967531e-05
```

For M1, we know p=5, n=25, so for IC2:

```r
n <- 25
p=5
IC2 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-5)/i)*exp(2*n*i/((n-2*p-2)*(n-2*p-2*i-2))) - (n-p-i)/i
  IC2[i] <- 1 - pf(f_num, i, n-p-i)
}
IC2
```

```
## [1] 6.425828e-03 1.767892e-04 6.552481e-07 2.979617e-11 0.000000e+00
## [6] 0.000000e+00 1.000000e+00 1.000000e+00
```

For M1, we know p=5, n=25, so for IC3:

```r
n <- 25
p=5
IC3 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-5)/i)*exp(((n-p)*(p+i)*log(n-p-i)-(n-p-i)*p*log(n-p))/((n-p-i)*(n-p))) - (n-p-i)/i
  IC3[i] <- 1 - pf(f_num, i, n-p-i)
}
IC3
```

```
## [1] 3.813045e-02 1.324483e-02 4.554776e-03 1.575617e-03 5.504635e-04
## [6] 1.954053e-04 7.115851e-05 2.694694e-05
```

Then we make a table using these results:

```r
overfit_table <- matrix(0, nrow=3, ncol=8)
for (i in 1:8){
  overfit_table[1,i] <- IC1[i]
  overfit_table[2,i] <- IC2[i]
  overfit_table[3,i] <- IC3[i]
}
overfit_table
```

```
##             [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 0.061993039 0.0287751060 1.256454e-02 5.237465e-03 0.0020747400
## [2,] 0.006425828 0.0001767892 6.552481e-07 2.979617e-11 0.0000000000
## [3,] 0.038130449 0.0132448275 4.554776e-03 1.575617e-03 0.0005504635
##              [,6]         [,7]         [,8]
## [1,] 0.0007763016 2.728411e-04 8.967531e-05
## [2,] 0.0000000000 1.000000e+00 1.000000e+00
## [3,] 0.0001954053 7.115851e-05 2.694694e-05
```

Secondly, we will consider n=100, p=5. For M1, we know p=5, n=100, so for IC1:

```r
n <- 100
p=5
IC1 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-p)/i)*exp(2*(n+1)*i/((n-p)*(n-p-i))) - (n-p-i)/i
  IC1[i] <- 1 - pf(f_num, i, n-p-i)
}
IC1
```

```
## [1] 0.078075092 0.044346796 0.024630175 0.013673498 0.007593488 0.004212596
## [7] 0.002331189 0.001285272
```

For M1, we know p=5, n=100, so for IC2:

```
n <- 100
p=5
IC2 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-5)/i)*exp(2*n*i/((n-2*p-2)*(n-2*p-2*i-2))) - (n-p-i)/i
  IC2[i] <- 1 - pf(f_num, i, n-p-i)
}
IC2
```

```
## [1] 0.0628526753 0.0300277659 0.0137227425 0.0060971939 0.0026256214
## [6] 0.0010903217 0.0004343283 0.0001650842
```

For M1, we know p=5, n=100, so for IC3:

```
n <- 100
p=5
IC3 <- c(rep(0, 8))
for (i in 1:8){
  f_num = ((n-5)/i)*exp(((n-p)*(p+i)*log(n-p-i)-(n-p-i)*p*log(n-p))/((n-p-i)*(n-p))) - (n-p-i)/i
  IC3[i] <- 1 - pf(f_num, i, n-p-i)
}
IC3
```

```
## [1] 1.701753e-02 3.317702e-03 6.529802e-04 1.298958e-04 2.596594e-05
## [6] 5.194280e-06 1.036975e-06 2.062236e-07
```

Then we make a table using these results:

```
overfit_table <- matrix(0, nrow=3, ncol=8)
for (i in 1:8){
  overfit_table[1,i] <- IC1[i]
  overfit_table[2,i] <- IC2[i]
  overfit_table[3,i] <- IC3[i]
}
overfit_table
```

```
##            [,1]        [,2]         [,3]         [,4]         [,5]         [,6]
## [1,] 0.07807509 0.044346796 0.0246301746 0.0136734985 7.593488e-03 4.212596e-03
## [2,] 0.06285268 0.030027766 0.0137227425 0.0060971939 2.625621e-03 1.090322e-03
## [3,] 0.01701753 0.003317702 0.0006529802 0.0001298958 2.596594e-05 5.194280e-06
##              [,7]         [,8]
## [1,] 2.331189e-03 1.285272e-03
## [2,] 4.343283e-04 1.650842e-04
## [3,] 1.036975e-06 2.062236e-07
```

(12)

Firstly, if we keep the sample number to be the same (to be 25 or 100), just added more parameters to the model, then the probability will be smaller after adding parameters than before.

Secondly, if we added same number of parameters to two sample groups, the first group has 25 samples, the second group has 100 samples. Then the probabily of 25 samples will be larger than probability of 100 samples.

(13)

Write by hand.

(14)

Here is the conclusion: If our sample numbers is infinity, the overfitting probability we get from IC1 and IC2 are the same, also the overfitting probability is really low for IC3. No matter you use IC1, IC2 or IC3, you will get a conclusion that there is no overfitting problem for your model. So if the sample size is really large, you can avoid the overfitting problem.