Statistical Learning Problem Set 1

Stavrou Athanasios (159917)

March 28, 2018

1 Exercise

"In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

a Generate a predictor X of length $n = 100$ distributed uniformly, as well as a noise vector ϵ of length $n = 100$ distributed normally."

To do that, I use the following code:

```
f = function(x,j) {
  x ^ j
}

set.seed(165)
k=10
n=100
dmatrix=matrix(data=NA, nrow=n, ncol=k)

x = runif(100, min = 0, max = 1)
eps = rnorm(100, mean = 0, sd = 0.5)
for(j in 1:k){
  dmatrix[,j] = f(x,j)
}
```

First, I create a function to fill the design matrix, so that I will not have to do it manually. I create a matrix "dmatrix" and fill it with NA values. I generate the vector x, uniformly distributed from 0 to 1 and then create the error terms, eps, normally distributed with mean 0 and standard deviation 0.5. After I have created my variables, I run the function that I created earlier so that I fill dmatrix with x in the powers of i, $i \in [1, 10]$. I will need this matrix for later uses.

b "Generate a response vector Y of length n = 100 according to the model

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_4 x^4 + \epsilon$$

where β_0, β_1, β_2, and β_4 are constants of your choice."

I used the values 50, 0.8, 0.5 and 0.3 for β_0, β_1, β_2, and β_4 respectively. I also created a dataframe (datf) for later uses, containing the response variable Y and the matrix x, generated on the previous question (a).

```
y = 50 + 0.8*f(x,1) + 0.5*f(x,2) + 0.3*f(x,4) + eps
datf = data.frame(y,dmatrix)
```

c "Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors $x, x^2, \ldots, x^{10}$. What is the best model obtained according to C_p, BIC and R^2_{adj}? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained."

```
library(leaps)

#########################################exhaustive#############################################

bsub = regsubsets(y~., data = datf, nvmax=10)
bsub_summary = summary(bsub)
```

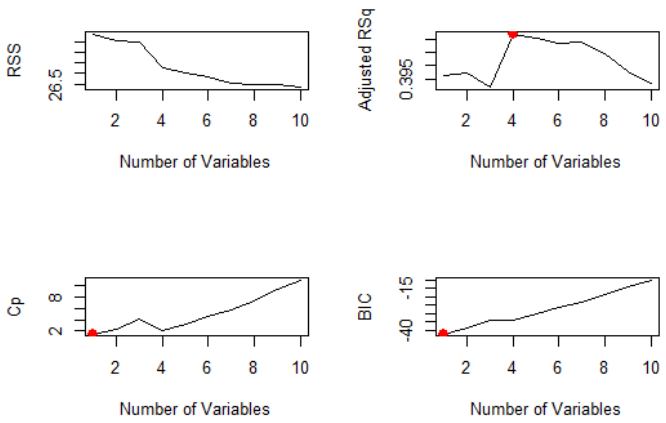
For this question, I firstly run the library I am going to need for regsubsets() function (leaps). Then, I run regsubsets of y on all the predictors, and get the summary as well as the R^2_{adj}. From Figure 1, I see that the best model for every size is to get the next x in the design matrix (e.g. for the best model for 3 variables the 3 first variables, for the best model for 4 variables the first 4 variables and so on). The best model regarding R^2_{adj} though, is the model with 4 variables, as I see that it is the highest.

Figure 1: Best subset, exhaustive

```
1 subsets of each size up to 10
Selection Algorithm: exhaustive
          X1  X2  X3  X4  X5  X6  X7  X8  X9  X10
1  ( 1 )  "*" " " " " " " " " " " " " " " " " " "
2  ( 1 )  "*" "*" " " " " " " " " " " " " " " " "
3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " " "
4  ( 1 )  "*" "*" "*" "*" " " " " " " " " " " " "
5  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " "
6  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " " " " "
7  ( 1 )  "*" "*" "*" "*" "*" "*" "*" " " " " " "
8  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " " "
9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" "*" " "
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
> bsub_summary$adjr2
 [1] 0.3963650 0.3971837 0.3919766 0.4119696 0.4104807
 0.4083570 0.4086383 0.4042787 0.3977709 0.3932691
>
```

To get the needed graphs, I run code to get the R^2_{adj} for every model, as well as C_p and BIC. I put the 4 plots (with R^2 included) into a single window and point out the maximum/minimum (depending on the criterion) points (Figure 2).

Figure 2: The four plots in 1 window $(R^2, R^2_{adj}, C_p, BIC)$



According to R^2_{adj}, the best model is the one with 4 variables (x, x^2, x^3 and x^4), while according to C_p and BIC the best model is the one with 1 variable (x). The coefficients of these 2 model (1 and 4 variable models) are:

```
> coef(bsub, 4)
(Intercept)          X1          X2          X3          X4
  50.487323   -7.229875   33.865520  -48.367683   23.087428
> coef(bsub, 1)
(Intercept)          X1
  49.915788    1.580719
>
```

The code used for the plots is the following:

```
###########plots###############################
par(mfrow = c(2,2))
plot(bsub_summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
plot(bsub_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type =
    "l")

adj_r2_max = which.max(bsub_summary$adjr2)
adj_r2_max

points(adj_r2_max, bsub_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

points(adj_r2_max, bsub_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

plot(bsub_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
cp_min = which.min(bsub_summary$cp)
cp_min
points(cp_min, bsub_summary$cp[cp_min], col = "red", cex = 2, pch = 20)

plot(bsub_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
bic_min = which.min(bsub_summary$bic)
bic_min
points(bic_min, bsub_summary$bic[bic_min], col = "red", cex = 2, pch = 20)

coef(bsub, 4)
coef(bsub, 1)
```

d "Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?"

For this question, I use the exact same code, only that on regsubsets function I insert the option "method = forward", so that it calculated the forward stepwise selection. This time the results change. I get that for the same amount of variables the variables themselves vary (e.g. the best model for 2 variables is x and x^{10}). This can be shown by the fact that my graphs are different too. According to R^2_{adj}, the best model is the one with 5 variables (x, x^2, x^4, x^9 and x^{10}), while according to C_p and BIC the best model remains the one with 1 variable (x). The coefficients of these 2 model (1 and 5 variable models) are:

```
> coef(bsub_fwd, 5)
(Intercept)          X1          X2          X4          X9         X10
  50.378798   -4.077309   12.894421  -13.752109   22.649899  -16.191214
> coef(bsub_fwd, 1)
(Intercept)          X1
  49.915788    1.580719
>
```

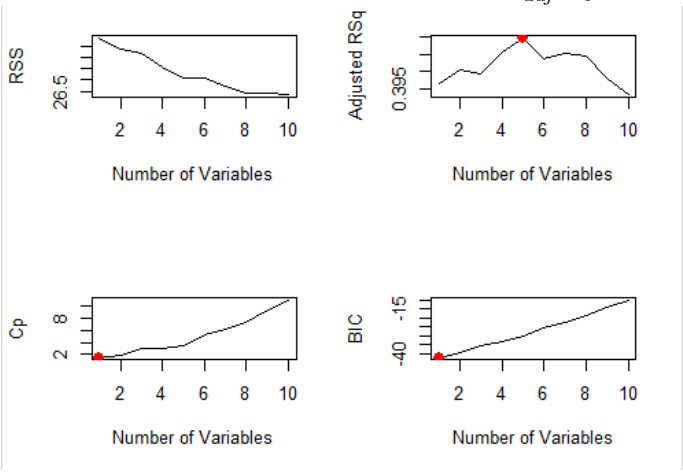
```
bsub_fwd = regsubsets(y~., datf,nvmax=10, method = "forward")
bsub_summary2 =summary(bsub_fwd)
bsub_summary2
bsub_summary2$adjr2
```

Figure 3: Best subset, forward

```
1 subsets of each size up to 10
Selection Algorithm: forward
          X1  X2  X3  X4  X5  X6  X7  X8  X9  X10
1  ( 1 )  "*" " " " " " " " " " " " " " " " " " "
2  ( 1 )  "*" " " " " " " " " " " " " " " " " "*"
3  ( 1 )  "*" " " " " " " " " " " " " " "*" "*"
4  ( 1 )  "*" "*" " " " " " " " " " " " "*" "*"
5  ( 1 )  "*" "*" " " "*" " " " " " " " "*" "*"
6  ( 1 )  "*" "*" "*" "*" " " " " " " " "*" "*"
7  ( 1 )  "*" "*" "*" "*" "*" " " " " " " "*" "*"
8  ( 1 )  "*" "*" "*" "*" "*" " " " " "*" "*" "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" " " "*" "*" "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
> bsub_summary2$adjr2
 [1] 0.3963650 0.4004253 0.3991120 0.4055594 0.4098289
 0.4037727 0.4052841 0.4045119 0.3979055 0.3932691
>
```

The code for the plots remains exactly the same.

Figure 4: The four plots in 1 window $(R^2, R^2_{adj}, C_p, BIC)$



To get the backward stepwise I use the same commands but with "method = backward" option. The results change once again, the best model according to R^2_{adj} is the one with 4 variables (x^2, x^3, x^4 and x^5), while the best model according to C_p and BIC is the one with 3 variables (x^3, x^4 and x^5). The respective coefficients are:

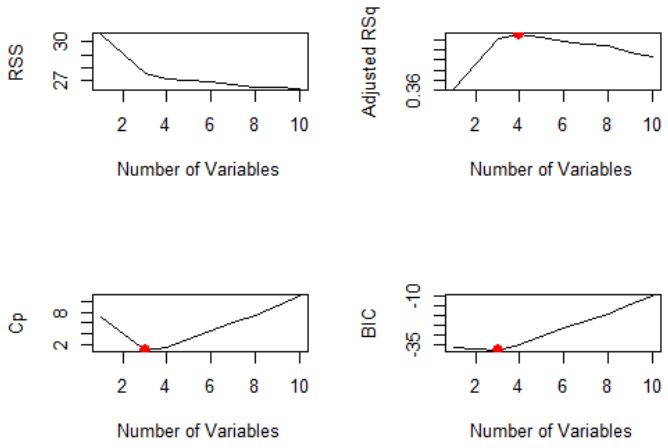
```
> coef(bsub_bwd, 4)
(Intercept)          X2          X3          X4          X5
   50.21097   -13.43125    72.17088  -108.08691    51.12930
> coef(bsub_bwd, 3)
(Intercept)          X3          X4          X5
   50.06406    19.13925   -38.00776    20.67474
>
```

```
bsub_fwd = regsubsets(y~., datf,nvmax=10, method = "backward")
bsub_summary2 =summary(bsub_fwd)
bsub_summary2
bsub_summary2$adjr2
```

Figure 5: Best subset, backward

```
1 subsets of each size up to 10
Selection Algorithm: backward
          X1  X2  X3  X4  X5  X6  X7  X8  X9  X10
1  ( 1 )  " " " " "*" " " " " " " " " " " " " "
2  ( 1 )  " " " " "*" "*" " " " " " " " " " " "
3  ( 1 )  " " " " "*" "*" "*" " " " " " " " " "
4  ( 1 )  " " "*" "*" "*" "*" " " " " " " " " "
5  ( 1 )  " " "*" "*" "*" "*" " " " " " " "*" "
6  ( 1 )  " " "*" "*" "*" "*" " " " " " " "*" "*"
7  ( 1 )  "*" "*" "*" "*" "*" " " " " " " "*" "
8  ( 1 )  "*" "*" "*" "*" "*" " " " " " " "*" "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " "*" "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
> bsub_summary3$rss
 [1] 30.54928 29.05653 27.52793 27.09381 26.93240
26.84991 26.65753 26.44976 26.44931 26.35684
>
```

Figure 6: Best subset, backward



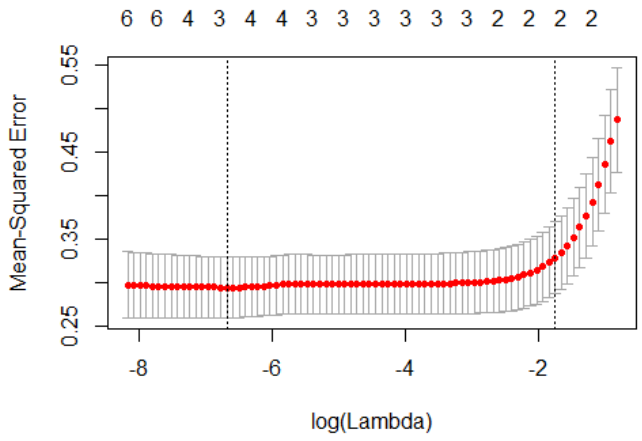
The code for the plots remains unchanged. In conclusion, the results differ from the previous question, as I choose completely different models (depending on the model/the criterion I want), not only different variables but different number of variables too.

e "Now fit a lasso model to the simulated data, again using $x, x^2, \ldots, x^{10}$ as predictors. Use cross-validation to select the optimal value of λ. Create plots of the crossvalidation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained."

I firstly call the library glmnet. Then, use the cv.glmnet command, to get all λ and choose the minimum. I make predictions for the minimum value of λ and get the MSE and RMSE (with 2 different ways).

```
library(glmnet)

fit_lasso_cv = cv.glmnet(dmatrix, y, alpha = 1)
plot(fit_lasso_cv)
coef(fit_lasso_cv, s = "lambda.min")

pred=predict(fit_lasso_cv, dmatrix, s = "lambda.min")
mean((y - pred) ^ 2)
sqrt(fit_lasso_cv$cvm)
sqrt(fit_lasso_cv$cvm[fit_lasso_cv$lambda == fit_lasso_cv$lambda.min])
```

Figure 7: MSE-log(λ)



The MSE - λ graph is illustrated by Figure 7. The coefficients for lambda.min are:

```
11 x 1 sparse Matrix of class "dgCMatrix"
                     1
(Intercept) 50.089643
V1           .
V2           2.749988
V3           .
V4           .
V5          -2.945412
V6           .
V7           .
V8           .
V9           .
V10          2.008834
> pred=predict(fit_lasso_cv, dmatrix, s = "lambda.min")
> mean((y - pred) ^ 2)
[1] 0.2760614
>
```

The variables chosen for lambda.min are different than the 3 models I got from regsubsets (all 3 methods). If we put high λ we get high MSE, because the penalty of the coefficients is very high.

f "Now generate a response vector Y according to the model

$$Y = \beta_0 + \beta_7 x^7 + eps$$

and perform best subset selection and the lasso. Discuss the results obtained."

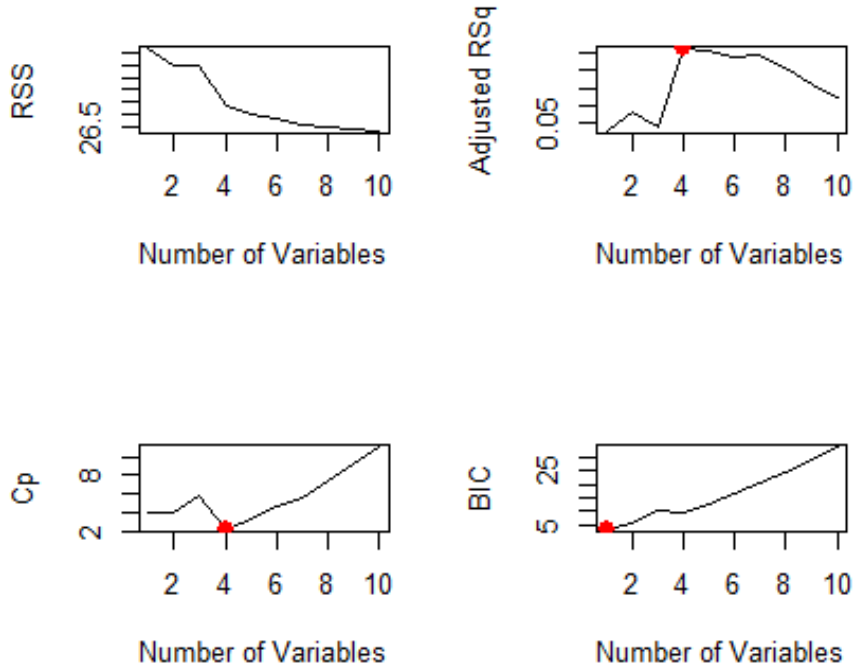
After creating the new response value, I run regsubsets and lasso on the data, in the same way I did on the previous questions. With regsubsets I get new coefficients and new plots:

```
> coef(bsub2, 4)
(Intercept)          X1          X2          X3          X4
  50.501844   -8.421735   35.809721  -53.745163   26.782638
> coef(bsub2, 1)
(Intercept)          X1
 49.9304224   0.4713663
>
```

Figure 8: NEW exhaustive regsubsets

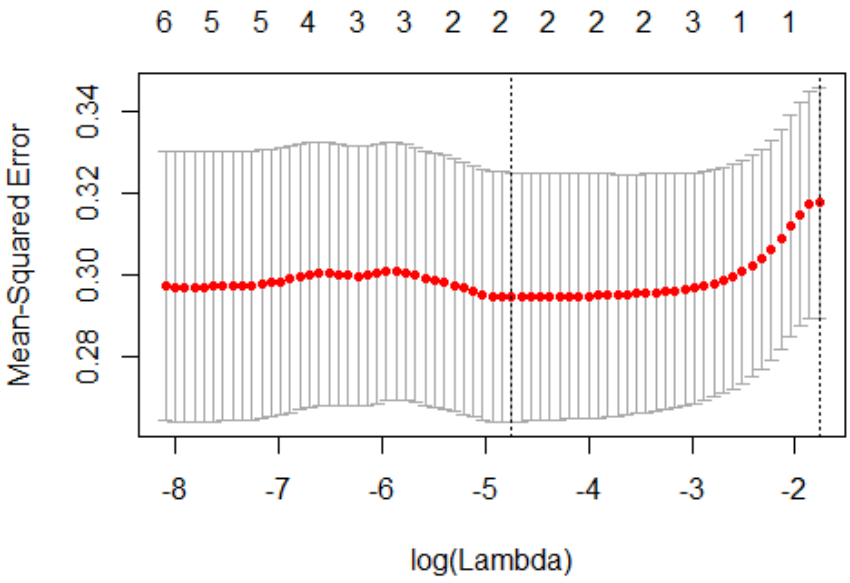
```
1 subsets of each size up to 10
Selection Algorithm: exhaustive
          X1  X2  X3  X4  X5  X6  X7  X8  X9  X10
1  ( 1 )  "*" " " " " " " " " " " " " " " " " " "
2  ( 1 )  "*" "*" " " " " " " " " " " " " " " " "
3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " " "
4  ( 1 )  "*" "*" "*" "*" " " " " " " " " " " " "
5  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " "
6  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " " " " "
7  ( 1 )  "*" "*" "*" "*" "*" "*" "*" " " " " " "
8  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " " "
9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" "*" " "
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
> bsub2_summary$adjr2
 [1] 0.04552595 0.05631502 0.04735594 0.09207947 0.09142677
0.08795600 0.08840864 0.08168831 0.07165644 0.06471690
>
```

Figure 9: Best model according to R^2_{adj}, C_p and BIC



According to R^2_{adj} and C_p, the best model is the one with 4 variables (x, x^2, x^3 and x^4) and the best model according to BIC is the one with 1 variable (x). With lasso, I get different coefficients and the new MSE-λ plot:

```
11 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 50.089643
V1           .
V2           2.749988
V3           .
V4           .
V5          -2.945412
V6           .
V7           .
V8           .
V9           .
V10          2.008834
> pred=predict(fit_lasso_cv, dmatrix, s = "lambda.min")
> mean((y - pred) ^ 2)
[1] 0.2760614
>
```



2 Exercise

"In this exercise, we will predict the times that some student raises his hand in class ("raisedhands" variable) using the other features in the College data set. You can find information about the dataset in the Attributes.txt file. The College.csv file contains the data. Use the read.csv() function to read the data set.

1. Check whether there are any missing values. Clean the data set, while trying to maintain as much information as possible."

   After reading and inspecting my data, I see that most of my NAs are coming from the "LuckyNumber" variable, so I drop it. I also drop the variable "X", that just enumerates my observations (like a time variable, only that this is not needed). I see that the remaining NAs are in the "StudentAbsenceDays" variable, which is a dummy. Since the NAs are not that many, I replace with the prevailing value, which is "Under-7". This cleans the dataset, with no NA values left.

```
MyData <- read.csv(file="C:/Users/Trainwreck/Documents/R/College.csv", header=TRUE,
    sep=",")
sum(is.na.data.frame(MyData))
summary(MyData$StudentAbsenceDays)

for(j in 1:nrow(MyData)){
  MyData$StudentAbsenceDays[is.na(MyData$StudentAbsenceDays)] = "Under-7"
}
summary(MyData$StudentAbsenceDays)

sum(is.na.data.frame(MyData$LuckyNumber))
df = MyData[ ,-c(1,19) ]
sum(is.na.data.frame(df))
```

2. "Report the dimensions of your clean data set."

```
dim(df)
```

This gives a dataframe 480x17.

3. "Split the data set into a training set and a test set with a 70%-30% ratio."

```
set.seed(60456)
indices = 1:nrow(df)
trainIndex = sample(x = indices,size = floor(0.7*nrow(df)),replace = F)

training = df[trainIndex,]
testing = df[-trainIndex,]

dim(training)
dim(testing)
```

The dimensions of training and testing are 336x17 and 144x17 respectively, which verifies that my split is done rightly.

4. " Fit a linear model using least squares on the training set, and report the test error obtained."

For every string variable, R creates a dummy for every value of that variable. If I simply run a linear regression of "raisedhands" on the rest variables, most people have the same "Nationality" and "PlaceOfBirth", so I have multicollinearity problem. To deal with that, I drop the Variable "Nationality", as well as "GradeID".

```
df = df[ ,-c(2,5) ]
training = training[,-c(2,5)]
testing = testing[,-c(2,5)]

myfit = lm(raisedhands ~ ., training)
test_mse = mean((testing$raisedhands - predict(myfit,testing))^2)
```

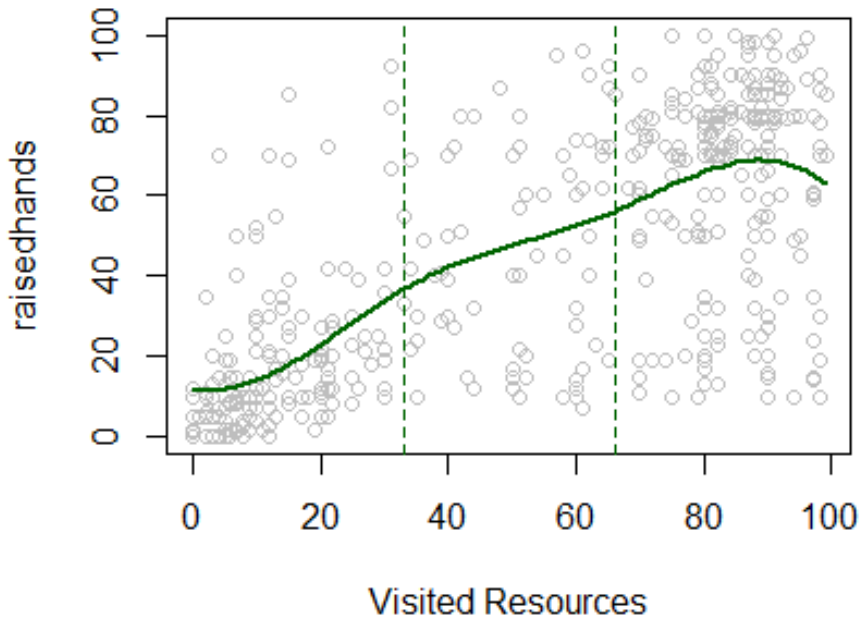
The test MSE I get is 393,604.

5. "Fit a 3^{rd} degree polynomial that predicts raisedhands using only the VisitedResources feature on the training test and report the test error obtained. Repeat the process using a cubic spline. Create a scatter plot and then add lines for the two models. What do you observe? Did the spline improve the prediction error?"

   First, I call the library "splines". Then, I fit the regression on the training data. I get the test MSE the same way I did before. The test MSE I get from this fit is 425,81. Then, I run a new fit, splfit, with knots on 33 and 66. The test MSE I get from this fit is 419,44 which is smaller than the previous fit. This is to be expected, as the data I have are split into 3 parts, so the fit is better. I conclude that the spline did improve the test MSE. I plot it and get the results on Figure 10. The knots I chose is on 1 and 2 thirds of the total visited resources, and by the Figure it SEEMS like a logical choice. The code is the following:

```
library(splines)

fit_3 = lm(raisedhands ~ poly(VisITedResources, degree = 3), data = training)
test_mse3 = mean((testing$raisedhands - predict(fit_3,testing))^2)

vrlims<-range(df$VisITedResources)

vr.grid<-seq(from=vrlims[1], to = vrlims[2])

splfit<-lm(raisedhands ~ bs(VisITedResources,knots = c(33,66)),data = training)
summary(splfit)

plot(df$VisITedResources,df$raisedhands,col="grey",xlab="Visited
    Resources",ylab="raisedhands")
points(vr.grid,predict(splfit,newdata =
    list(VisITedResources=vr.grid)),col="darkgreen",lwd=2,type="l")
abline(v=c(33,66),lty=2,col="darkgreen")

spline_test_mse = mean((testing$raisedhands - predict(splfit,testing))^2)
```

Figure 10: Spline Fit



6. "Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained."

   For this question I use the glmnet library. To get the lasso I use the command cv.glmnet with option $alpha = 0$. The cv.glmnet gives many results, but I want the predictions for lambda.min using the testing data. The ridge test MSE for the minimum λ is 371,85.

```
library(glmnet)

x1=model.matrix(raisedhands~.,training)[,-1]
x2=model.matrix(raisedhands~.,testing)[,-1]

fit_ridge_cv = cv.glmnet(x1, training$raisedhands, alpha = 0)
ridgepred=predict(fit_ridge_cv, x2, s = "lambda.min")
ridge_mse = mean((testing$raisedhands - ridgepred) ^ 2)
```

7. "Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates."

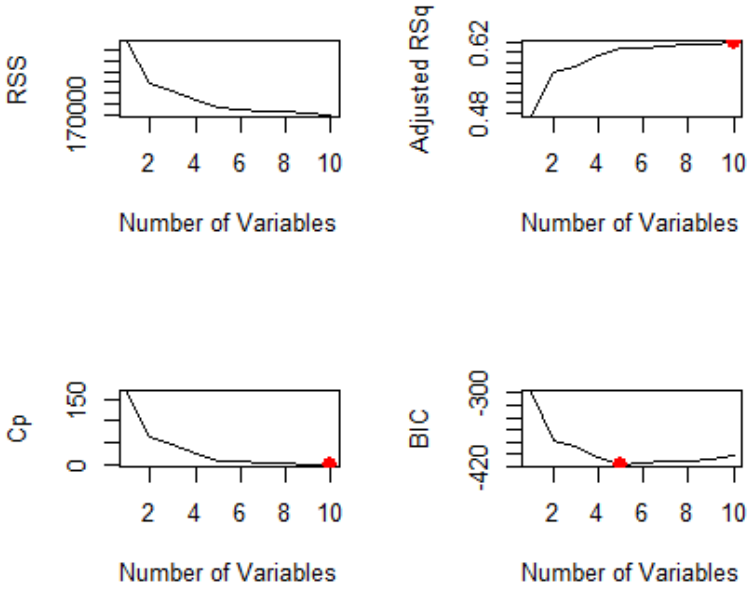
   The same procedure as before, but with option alpha=1.

```
fit_lasso_cv = cv.glmnet(x1, training$raisedhands, alpha = 1)
lassopred=predict(fit_lasso_cv, x2, s = "lambda.min")
lasso_mse=mean((testing$raisedhands - lassopred) ^ 2)

newfit=coef(fit_lasso_cv, s = "lambda.min")
length(newfit@x)
```

   To get the number of coefficients that are not zero for the minimum λ I run a new regression, newfit and check the length of x, where glmnet stores the coefficients.

8. "Fit a model of your choice and provide arguments for that choice. Some criteria could be interpretability, predictive accuracy, cost or domain knowledge."

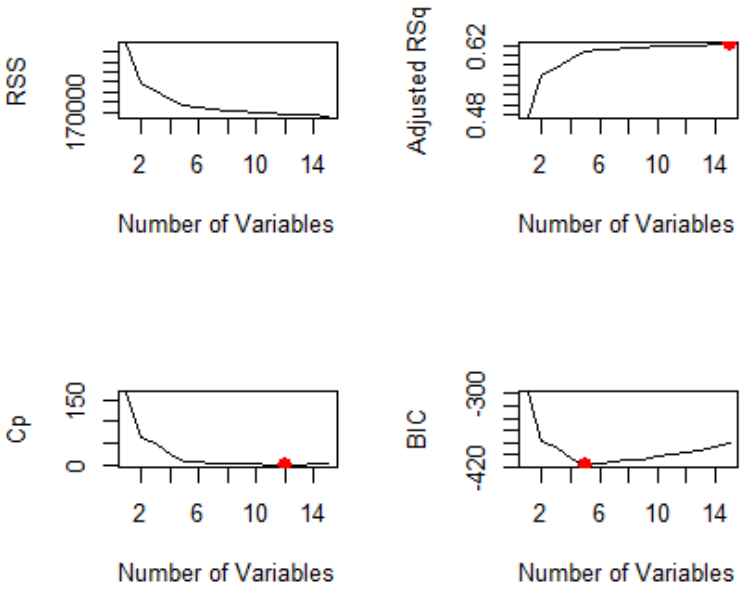
   For this question, I will run regsubsets() function for the exhaustive, forward and backward stepwise methods. I will choose the best model after looking at R^2_{adj}, C_p, BIC as well as how costly the computation of the model is. For example, I won't choose the exhaustive model for $nvmax = ncol(df)$, as I have many variables and it takes time to compute. I set nvmax=10, so that I run some computations. I create the plots the same way I did on exercise 1, for R^2_{adj}, C_p and BIC. For the exhaustive method I get the following plots:



The optimal R^2_{adj}, C_p and BIC are 0.6172073, 1.611873 and -413.8095 respectively. Note that these criteria are not for the same model though, since the optimum BIC is on the model with 5 variables.

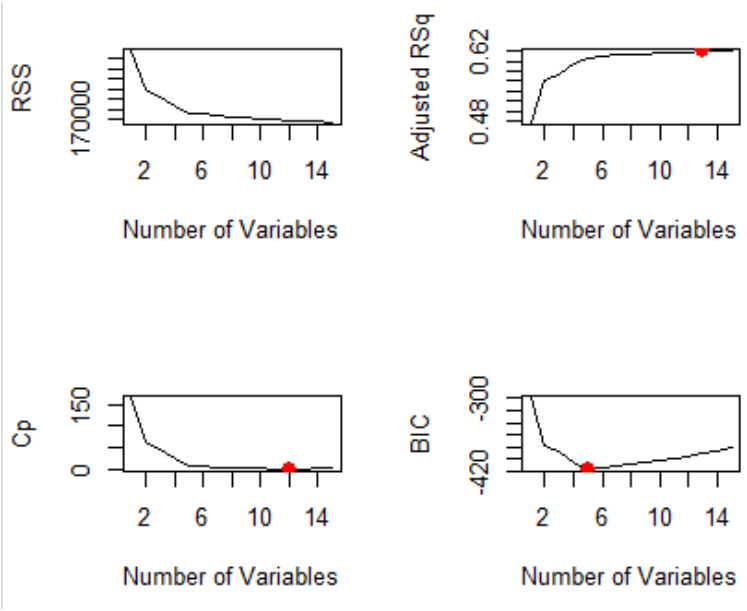
```
library(leaps)
regsub = regsubsets(raisedhands~., df, nvmax=10)
regsub_summary = summary(regsub)

###########plots##############################
par(mfrow = c(2,2))
plot(regsub_summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
plot(regsub_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type
    = "l")

adj_r2_max = which.max(regsub_summary$adjr2)
adj_r2_max

points(adj_r2_max, regsub_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

points(adj_r2_max, regsub_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

plot(regsub_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
cp_min = which.min(regsub_summary$cp)
cp_min
points(cp_min, regsub_summary$cp[cp_min], col = "red", cex = 2, pch = 20)

plot(regsub_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
bic_min = which.min(regsub_summary$bic)
bic_min
points(bic_min, regsub_summary$bic[bic_min], col = "red", cex = 2, pch = 20)

coef(regsub, 10)
coef(regsub, 5)
```

As done previously, to get the forward and backward stepwise regsubsets() I use the option method=forward and method=backward respectively. The code is the same and the plots for the forward stepwise are the following:



This time the best model according to R^2_{adj} is the one with 15 variables, according to C_p the one with 12 and according to BIC the one with 5 variables. The optimal R^2_{adj}, C_p and BIC are 0.6207165, 0.6770488 and -415.9972 respectively.

The plots for the backward stepwise are the following:



The best model according to R^2_{adj} is the one with 13 variables, according to C_p the one with 12 and according to BIC the one with 5 variables. The optimal R^2_{adj}, C_p and BIC are 0.6194647, 1.596838 and -415.9972 respectively.

To choose the best model, I have to take into account all characteristics of all methods/models. I leave the exhaustive method aside, due to the fact that it is computationally costly. Furthermore, the results I have presented are not even for the maximum number of variables, but for nvmax=10. With that in mind, I have to choose between 2 methods, backward and forward stepwise. Between the 2 methods, the best models of forward stepwise have better characteristics than the best models of the backward one (except the BIC, which are equal). For example, the R^2_{adj} of the best models for the forward and the backward methods are 0.6207165 and 0.6172073, and the C_p of the best models are 0.6770488 and 1.596838 respectively. Keeping that in mind I conclude that the best method is the forward stepwise one. From the forward stepwise method, I choose the best model to be the one with 12 variables, that is the one with the lowest C_p, but I take into consideration that even though the best model according to R^2_{adj} is the one with 15 variables, it is not dropping that sharply in the model with 12 variables. Furthermore, as I increase the number of variables the BIC increases too, which is not wanted.

To conclude with, I choose the model with 12 variables with the forward stepwise method which give me the following coefficients:

```
> coef(regsub_fwd, 12)
                  (Intercept)          PlaceofBirthIraq      PlaceofBirthlebanon
                  23.02548543                7.26629778                7.18272007
                TopicGeology                RelationMum          VisITedResources
                   7.05743105                4.95023784                0.31160750
ParentAnsweringSurveyYes                      ClassL                      ClassM
                  -4.56306934              -20.18713380              -10.26048977
     StageIDMiddleSchool                TopicFrench
                   5.68622482               -5.76580006
       AnnouncementsView                  Discussion
                   0.28667598                0.07076489

> |
```

3 Exercise

"Using the data you generated in Exercise 1.b create a dataframe that contains the design matrix and the response variable. You should avoid using ready functions for this exercise.

1. Define a function that takes as inputs some dataframe and the column index of the dataframe that belongs to the response variable y and computes the analytical solution of the least squares minimization problem using matrices."

   First, I create the same matrix I had before, ranging from x to x^{10}. Then, I create the dataframe and I create a second function, betahats, which takes as inputs the dataframe and the response variable and estimates the coefficients (linear regression coefficients).

```
f = function(x,j) {
  x ^ j
}

set.seed(165)
k=10
n=100
dmatrix=matrix(data=NA, nrow=n, ncol=k)

x = runif(100, min = 0, max = 1)
eps = rnorm(100, mean = 0, sd = 0.5)
for(j in 1:k){
  dmatrix[,j] = f(x,j)
}

y = 50 + 0.8*f(x,1) + 0.5*f(x,2) + 0.3*f(x,4) + eps
ones=matrix(data=1,nrow=n,ncol=1)
datf = data.frame(y,dmatrix)
desmat = model.matrix(y~.,data = datf)

betahats = function(dataframe, y){
  datmat=as.matrix(dataframe)
  datmaty=datmat[,y]
  datmatx=cbind(1,datmat[,-y])
  beta = round(solve(t(datmatx)%*%datmatx)%*%t(datmatx)%*%datmaty,digits=3)
  return(beta)
}

betas=betahats(datf,1)
betas
```

The results of this regression are:

```
> betas
             [,1]
          49.480
X1        43.418
X2      -836.431
X3      7456.439
X4    -38590.272
X5    126416.045
X6   -268182.147
X7    365412.117
X8   -307496.486
X9    145084.106
X10   -29304.551
>
```

2. "Repeat step (a), for the ridge solution. In this case the new function should take an additional input, the lambda parameter."

The 2 functions are pretty much the same, because the ridge regression coefficients are given by $(X'X - \lambda I)^{-1}X'Y$, while the simple linear regression's coefficients by $(X'X)^{-1}X'Y$. The function this time takes 1 input more, λ. In my example, I use $\lambda = 0.1$

```
ridgebetahats = function(dataframe, y, lambda){

  datmat=as.matrix(dataframe)
  datmaty=datmat[,y]
  datmatx=cbind(1,datmat[,-y])
  k=ncol(datmatx)
  I=diag(k)
  beta=round((solve(t(datmatx)%*%datmatx)+lambda*I)%*%t(datmatx)%*%datmaty,digits=3)
  return(beta)
}
ridgebetas=ridgebetahats(datf,1,0.1)
ridgebetas
```

The results of this regression are:

```
> ridgebetas=ridgebetahats(datf,1,0.01)
> ridgebetas
             [,1]
          100.240
X1         70.654
X2       -817.869
X3       7470.404
X4     -38579.105
X5     126425.378
X6    -268174.097
X7     365419.204
X8    -307490.141
X9     145089.878
X10    -29299.253
>
```

3. "Define a function that takes as inputs a vector that contains the beta estimates and the design matrix and returns the predicted values for the response variable."

   A simple function, as we know that $\hat{Y} = \hat{\beta}X$

```
predictions = function(betas, desmatrix){
  yhats=desmatrix%*%betas
  return(yhats)
}
yhats=predictions(betas,desmat)
```

4. "Define a function that takes as inputs the response and the predicted values for that response and computes the mean squared error."

```
mse = function(response, yhat){
  mse = mean((response - yhat) ^ 2)
  return(mse)
}

get_MSE=mse(y,yhats)
```

5. "Define a function that takes as inputs a dataframe, the number of CV folds k and using the above functions performs k-fold cross-validation, where in each iteration:
   a) computes the ols estimates and the rigde estimates
   b) predicts the y values for the two cases
   c) computes the mean squared error for the two cases and stores it in a matrix
   After finishing the CV process, the function prints the average MSE for the two cases."

   In this question, I have to use the functions created above.

```
my_cv = function(datf,k){
  #Reshuffling my data
  datf = datf[sample(nrow(datf)),]
  MSEmatrix=matrix(data=NA, nrow=k, ncol=2)
  MSEres=matrix(data=NA,nrow=1,ncol=2)
  folds = cut(seq(1,nrow(datf)),breaks=k,labels=FALSE)

  for(i in 1:k){
    testIndexes = which(folds==i,arr.ind=TRUE)
    testData = datf[testIndexes, ]
    trainData = datf[-testIndexes, ]
    testdesmat=model.matrix(y~.,testData)

    betasOLS=betahats(trainData,1)
    betasRIDGE=ridgebetahats(trainData,1,.1)

    olspred=predictions(betasOLS,testdesmat)
    ridgepred=predictions(betasRIDGE,testdesmat)

    getolsMSE=mse(testData$y,olspred)
    getridgeMSE=mse(testData$y,ridgepred)

    MSEmatrix[i,1]=getolsMSE
    MSEmatrix[i,2]=getridgeMSE
  }
  MSEres[1,1]=mean(MSEmatrix[,1])
  MSEres[1,2]=mean(MSEmatrix[,2])

   return(MSEres)
} c=my_cv(datf,10)
```

Firstly, I reshuffle my data (for extra pseudo-randomness, not much needed though). Then, I create a new matrix, MSEmatrix, which is the matrix that the MSE for the 2 cases will go in. It has 2 columns, one for the OLS and one for the Ridge regression. The MSEres matrix is a 1 by 2 matrix, which will have the result in it, i.e. the average MSE for the 2 cases. Then, I cut the folds into k breaks and create test and train data into the fold. I calculate the $\hat{\beta_{OLS}}$ and $\hat{\beta_{Ridge}}$ with the above functions. I make the predictions for each case and get their MSE. After I have done that, I put the MSE into the MSEmatrix. After k repetitions I get the mean of the 2 columns, which is my final answer, the average MSE for the two cases. The c variable contains the average MSE for the 2 cases for k=10.

Notice that in betaRIDGE where I call the ridgebetahats() function, I use as default $\lambda = 0.1$, as the exercise needed the question to only take as inputs the dataframe and k!