EAS508-HW4

Saish Mandavkar

2022-10-20

Lab Code Homework

5.3 Cross Validation Labs

5.3.1 Validation Set Approach

```
# Setting the seed and loading the data

library(ISLR2)
set.seed(1)
train <- sample(392,196)
```

```
# Fitting a linear regression on the train data using subset option

lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)
```

```
# Predicting the estimates for the 392 observations and calculate the MSE for 192 observations

mean((Auto$mpg - predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 23.26601
```

```
# Fitting cubic regression and calculating the MSE

lm.fit2 <- lm(mpg ~poly(horsepower, 2), data = Auto, subset = train)

mean((Auto$mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 18.71646
```

```
# Fitting uadratic regression and calculating the MSE

lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto, subset = train)

mean((Auto$mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
## [1] 18.79401
```

```r
# Using different seed and calculatiing the values for all the three regressions - will result into dif

set.seed(2)
train <- sample(392,196)

# Linear regression MSE

lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)

mean((Auto$mpg - predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 25.72651
```

```r
# Cubic regression MSE

lm.fit2 <- lm(mpg ~poly(horsepower, 2), data = Auto, subset = train)

mean((Auto$mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 20.43036
```

```r
# Quadratic regression MSE


lm.fit3 <- lm(mpg ~poly(horsepower, 3), data = Auto, subset = train)

mean((Auto$mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
## [1] 20.38533
```

5.3.2 Leave One-Out Cross-Validation

```r
# LOOCV using glm() package

glm.fit <- glm(mpg ~ horsepower, data = Auto)

coef(glm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```r
# LOOCV using normal lm() function

lm.fit <- lm(mpg ~ horsepower, data = Auto)

coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```r
# Cross-validation error using glm() package

library(boot)

glm.fit <- glm(mpg ~ horsepower, data = Auto)

cv.err <- cv.glm(Auto, glm.fit)

cv.err$delta
```

```
## [1] 24.23151 24.23114
```

```r
# Calculating CV error for for polynomial of order 1 to 10 using a for loop.

cv.error <- rep(0,10)

for (i in  1:10) {

  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]

}

cv.error
```

```
##  [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305 18.96115
##  [9] 19.06863 19.49093
```

5.3.3 k-Fold Cross Validation

```r
# Calculating k-fold CV error for for polynomial of order 1 to 10 with k = 10

set.seed(17)
cv.error.10 <- rep(0,10)

for (i in  1:10) {

  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error.10[i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]

}

cv.error.10
```

```
##  [1] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```

6.5.3 PCR and PLS Regression

```
# Creating model matrix for x and storing all salary values in y

# Omitting NA values

Hitters <- na.omit(Hitters)

x <- model.matrix(Salary ~ ., Hitters)[, -1]
y <- Hitters$Salary

# Creating train and test by setting the R seed

set.seed(1)
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]
```

```
# Applying PCR to Hitters data to predcit Salary

library(pls)
```

Principal Components Regression

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```
set.seed(2)
pcr.fit <- pcr(Salary ~., data = Hitters, scale = TRUE, validation = "CV")
```

```
# Checking summary of our fit

summary(pcr.fit)
```

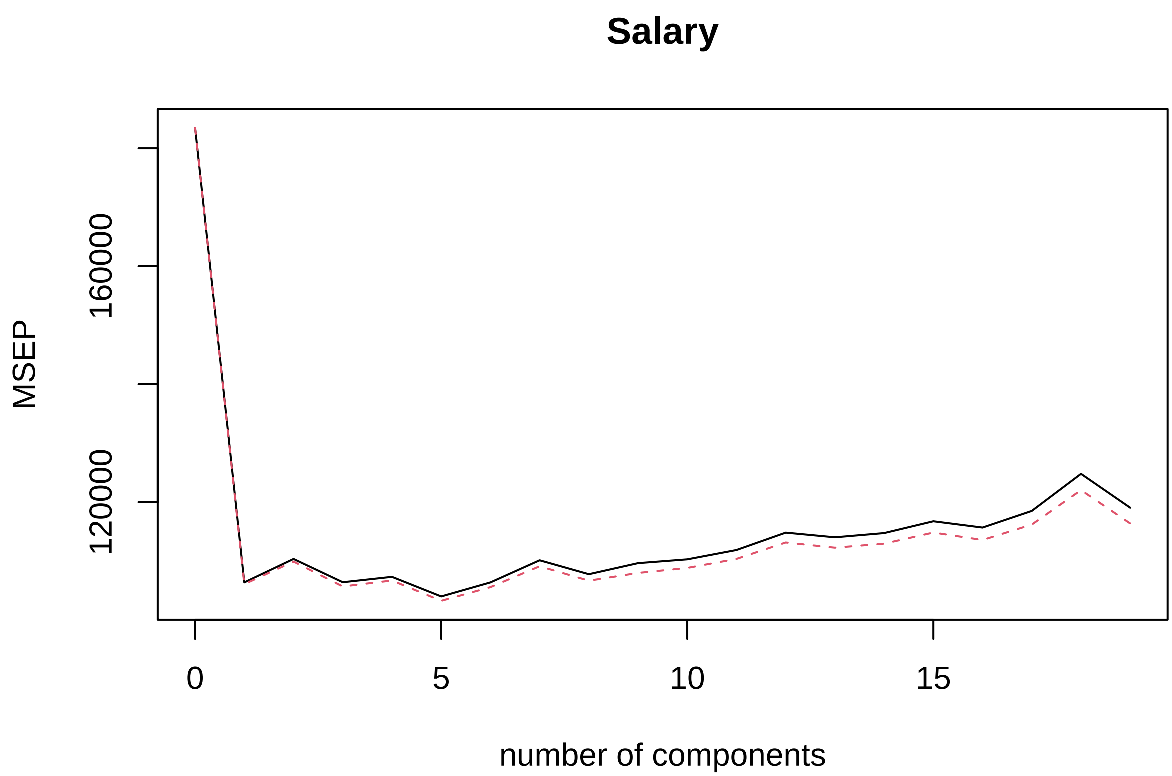
```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV             452    351.9    353.2    355.0    352.8    348.4    343.6
## adjCV          452    351.6    352.7    354.4    352.1    347.6    342.7
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       345.5    347.7    349.6     351.4     352.1     353.5     358.2
```

```
## adjCV    344.7    346.7    348.5     350.1     350.7     352.0     356.5
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        349.7     349.4     339.9     341.6     339.2     339.6
## adjCV     348.0     347.7     338.2     339.7     337.2     337.6
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26    94.96
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69    46.75
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         96.28     97.26     97.98     98.65     99.15     99.47     99.75
## Salary    46.86     47.76     47.82     47.85     48.10     50.40     50.55
##         16 comps  17 comps  18 comps  19 comps
## X          99.89     99.97     99.99    100.00
## Salary     53.01     53.85     54.61     54.61
```

```r
# Plotting cross-validation MSE

validationplot(pcr.fit, val.type = "MSEP")
```


```r
# Performing PCR on the training data using subset function and plotting the CV MSE

set.seed(1)

pcr.fit <- pcr(Salary ~., data = Hitters, subset = train, scale = TRUE,
```

```r
                validation = "CV")

validationplot(pcr.fit, val.type = "MSEP")
```




```r
# Find the lowest CV error when M = 5


pcr.pred <- predict(pcr.fit, x[test, ], ncomp = 5)

mean((pcr.pred - y.test)^2)
```

```
## [1] 142811.8
```

```r
# Fit PCR on complete dataset using M = 5 identified by CV

pcr.fit <- pcr(y~x, scale = TRUE, ncomp = 5)

summary(pcr.fit)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 5
## TRAINING: % variance explained
```

```
##    1 comps  2 comps  3 comps  4 comps  5 comps
## X    38.31    60.16    70.84    79.03    84.29
## y    40.63    41.58    42.17    43.22    44.90
```

```
# Implement PLS using plsr() function

set.seed(1)
pls.fit <- plsr(Salary ~ ., data = Hitters, subset = train, scale = TRUE,
                validation = "CV")

summary(pls.fit)
```

Partial Least Squares

```
## Data:    X dimension: 131 19
##  Y dimension: 131 1
## Fit method: kernelpls
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           428.3    325.5    329.9    328.8    339.0    338.9    340.1
## adjCV        428.3    325.0    328.2    327.2    336.6    336.1    336.6
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       339.0    347.1    346.4     343.4     341.5     345.4     356.4
## adjCV    336.2    343.4    342.8     340.2     338.3     341.8     351.1
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        348.4     349.1     350.0     344.2     344.5     345.0
## adjCV     344.2     345.0     345.9     340.4     340.6     341.1
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         39.13    48.80    60.09    75.07    78.58    81.12    88.21    90.71
## Salary    46.36    50.72    52.23    53.03    54.07    54.77    55.05    55.66
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         93.17     96.05     97.08     97.61     97.97     98.70     99.12
## Salary    55.95     56.12     56.47     56.68     57.37     57.76     58.08
##         16 comps  17 comps  18 comps  19 comps
## X          99.61     99.70     99.95    100.00
## Salary     58.17     58.49     58.56     58.62
```

```
# Evaluating the coresponding test set MSE

pls.pred <- predict(pls.fit, x[test, ], ncomp = 1)

mean((pls.pred - y.test)^2)
```

```
## [1] 151995.3
```

```r
# Fitting PLS on complete dataset when M = 1

pls.fit <- plsr(Salary ~ ., data = Hitters, scale = TRUE, ncomp = 1)

summary(pls.fit)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: kernelpls
## Number of components considered: 1
## TRAINING: % variance explained
##         1 comps
## X         38.08
## Salary    43.05
```

7.8.3 GAMs

```r
# Fit a GAM or predict wage using natural spline functions of years and age.

gam1 <- lm(wage ~ splines::ns(year, 4) + splines::ns(age, 5) + education, data = Wage)
```

```r
# Fit the model using smoothing splines

library(gam)
```

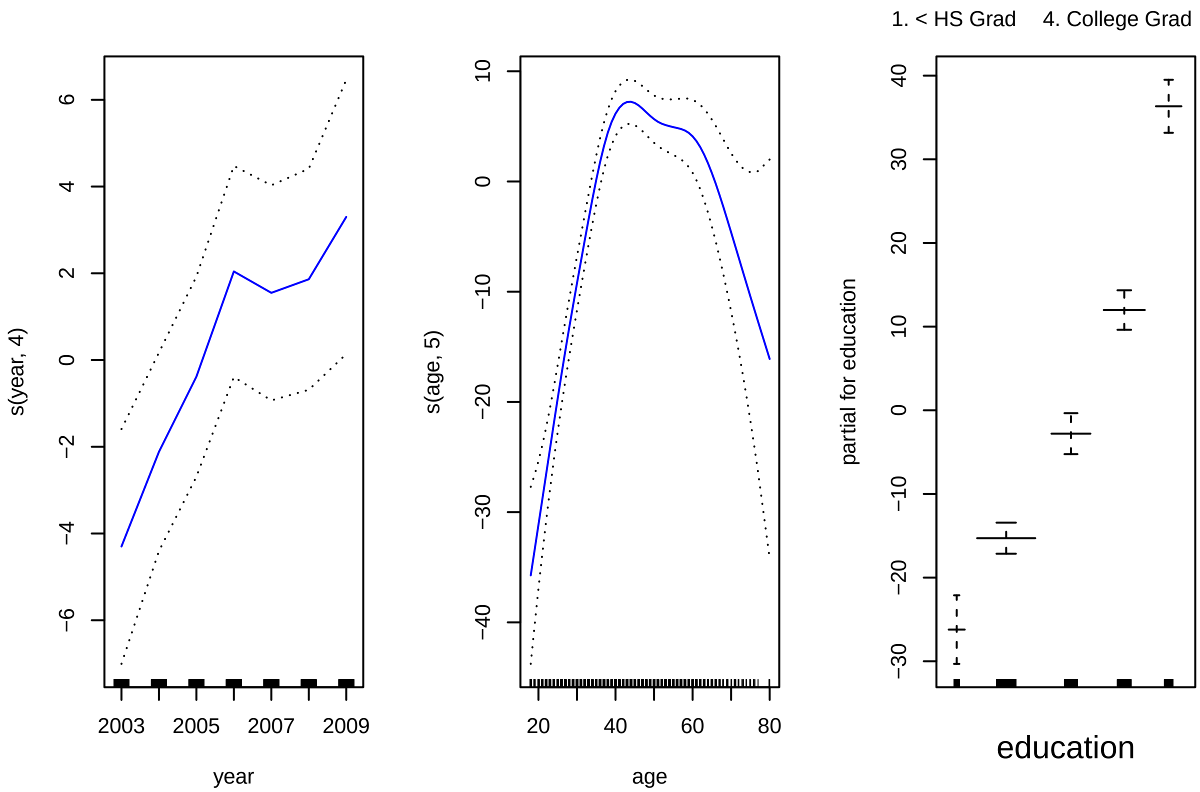
```
## Loading required package: splines
```

```
## Loading required package: foreach
```

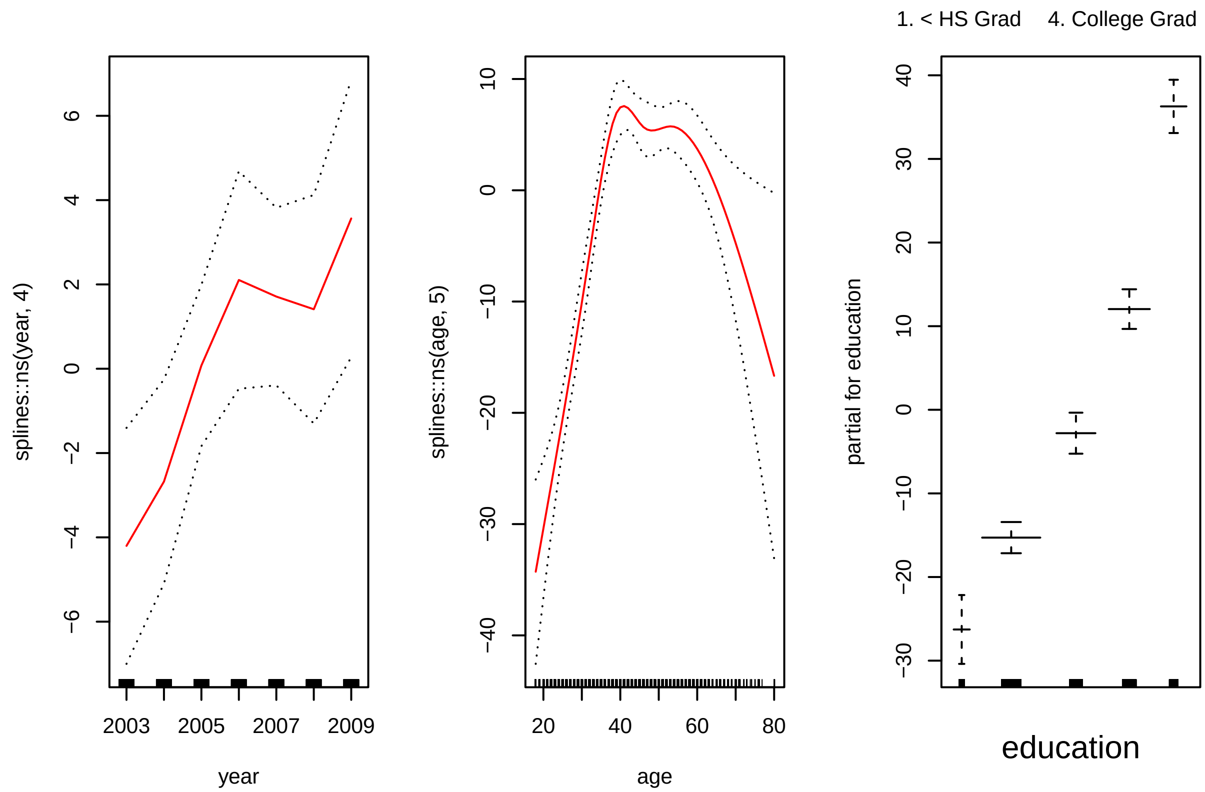
```
## Loaded gam 1.20.2
```

```r
gam.m3 <- gam(wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
```

```r
# Plot the model

par(mfrow = c(1,3))
plot(gam.m3, se = TRUE, col = "blue")
```




```r
# Plotting the GAM created using lm

par(mfrow = c(1,3))
plot.Gam(gam1, se = TRUE, col = "red")
```




```r
# Performing ANOVA test to determine the best model

gam.m1 <- gam(wage ~ s(age, 5) + education,  data = Wage)
gam.m2 <- gam(wage ~ year + s(age, 5) + education, data = Wage)

anova(gam.m1, gam.m2, gam.m3, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: wage ~ s(age, 5) + education
## Model 2: wage ~ year + s(age, 5) + education
## Model 3: wage ~ s(year, 4) + s(age, 5) + education
##   Resid. Df Resid. Dev Df Deviance       F    Pr(>F)
## 1      2990    3711731
## 2      2989    3693842  1  17889.2 14.4771 0.0001447 ***
## 3      2986    3689770  3   4071.1  1.0982 0.3485661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

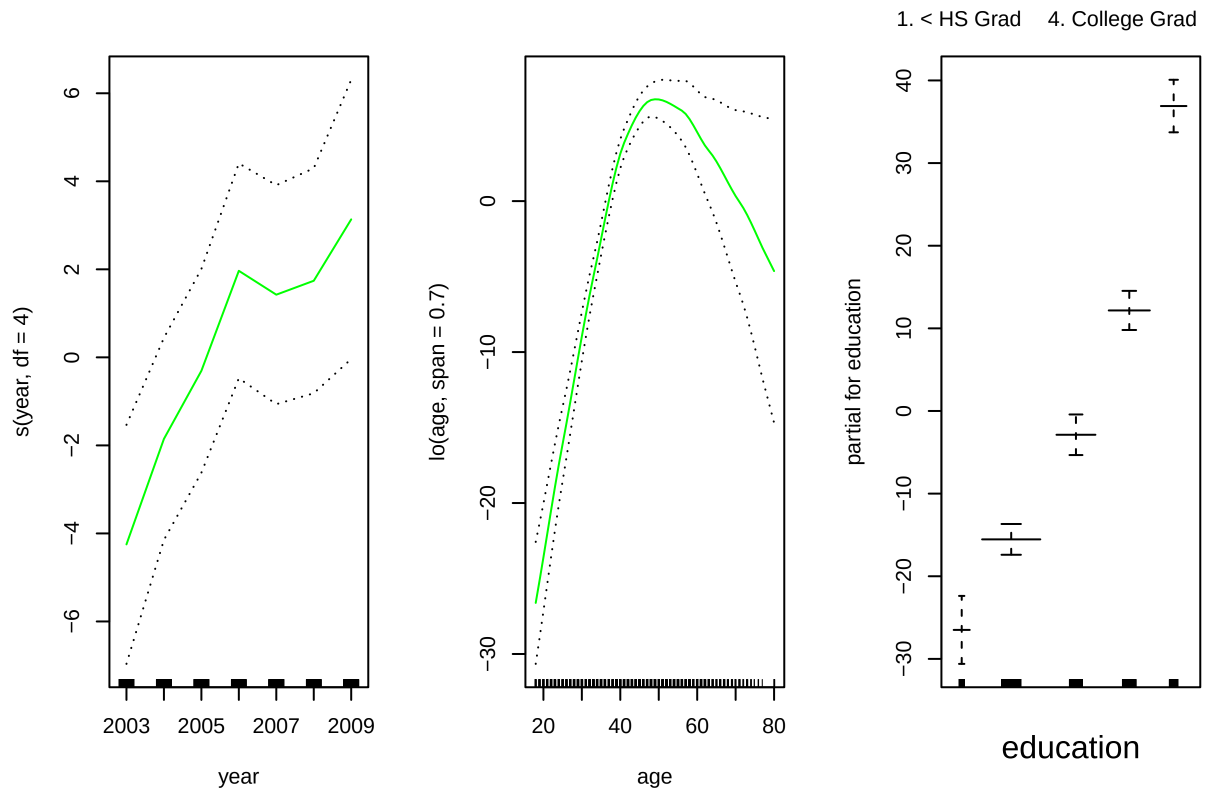
```r
# summary of gam.m3

summary(gam.m3)
```

```
##
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -119.43  -19.70   -3.33   14.17  213.48
##
## (Dispersion Parameter for gaussian family taken to be 1235.69)
##
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3689770 on 2986 degrees of freedom
## AIC: 29887.75
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq F value    Pr(>F)
## s(year, 4)     1   27162   27162  21.981 2.877e-06 ***
## s(age, 5)      1  195338  195338 158.081 < 2.2e-16 ***
## education      4 1069726  267432 216.423 < 2.2e-16 ***
## Residuals   2986 3689770    1236
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##              Npar Df Npar F  Pr(F)
## (Intercept)
## s(year, 4)         3  1.086 0.3537
## s(age, 5)          4 32.380 <2e-16 ***
## education
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Using the predict method for class GAM

preds <- predict(gam.m2, newdata = Wage)
```

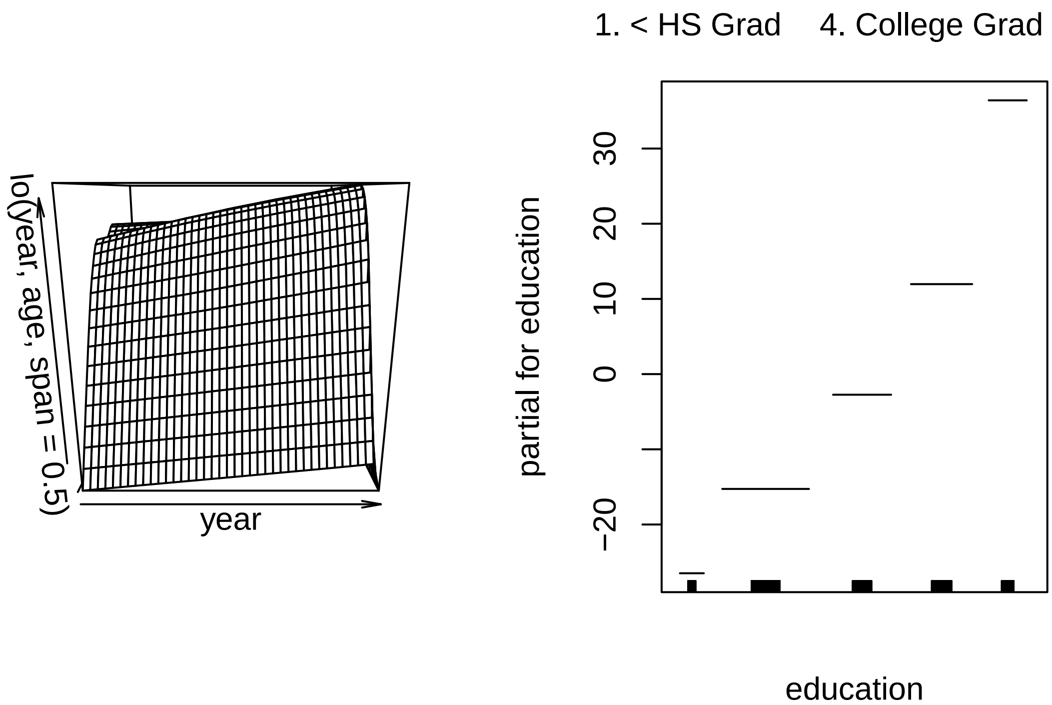
```r
# Using local regression fits in GAM using lo()

gam.lo <- gam(wage ~ s(year, df = 4) + lo(age, span = 0.7) + education, data = Wage)

par(mfrow = c(1,3))
plot.Gam(gam.lo, se = TRUE, col = "green")
```



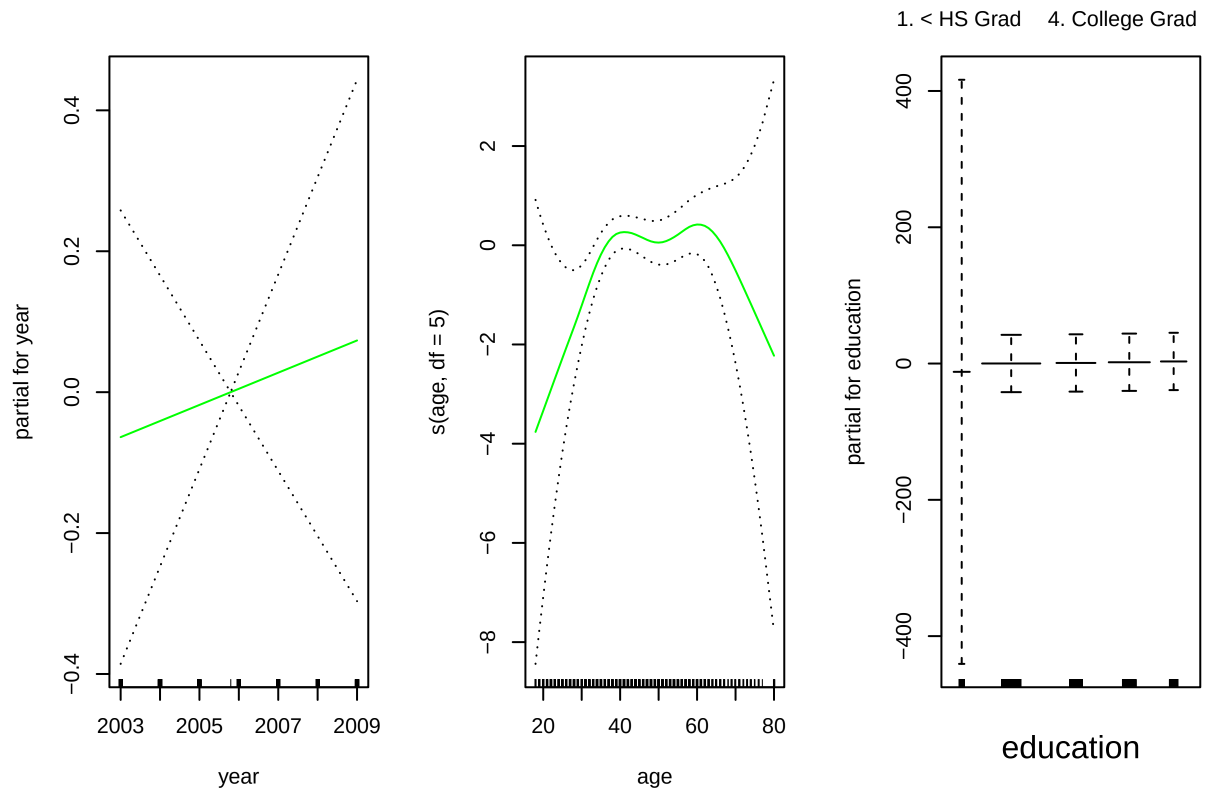

```
# Using lo() to create interactions before calling gam

gam.lo.i <- gam(wage ~ lo(year, age, span = 0.5) + education, data = Wage)
```

```
## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : liv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : lv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : liv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : lv
## too small. (Discovered by lowesd)
```

```
# Plotting the 2D surface using akima package

library(akima)
par(mfrow = c(1,2))
plot(gam.lo.i)
```



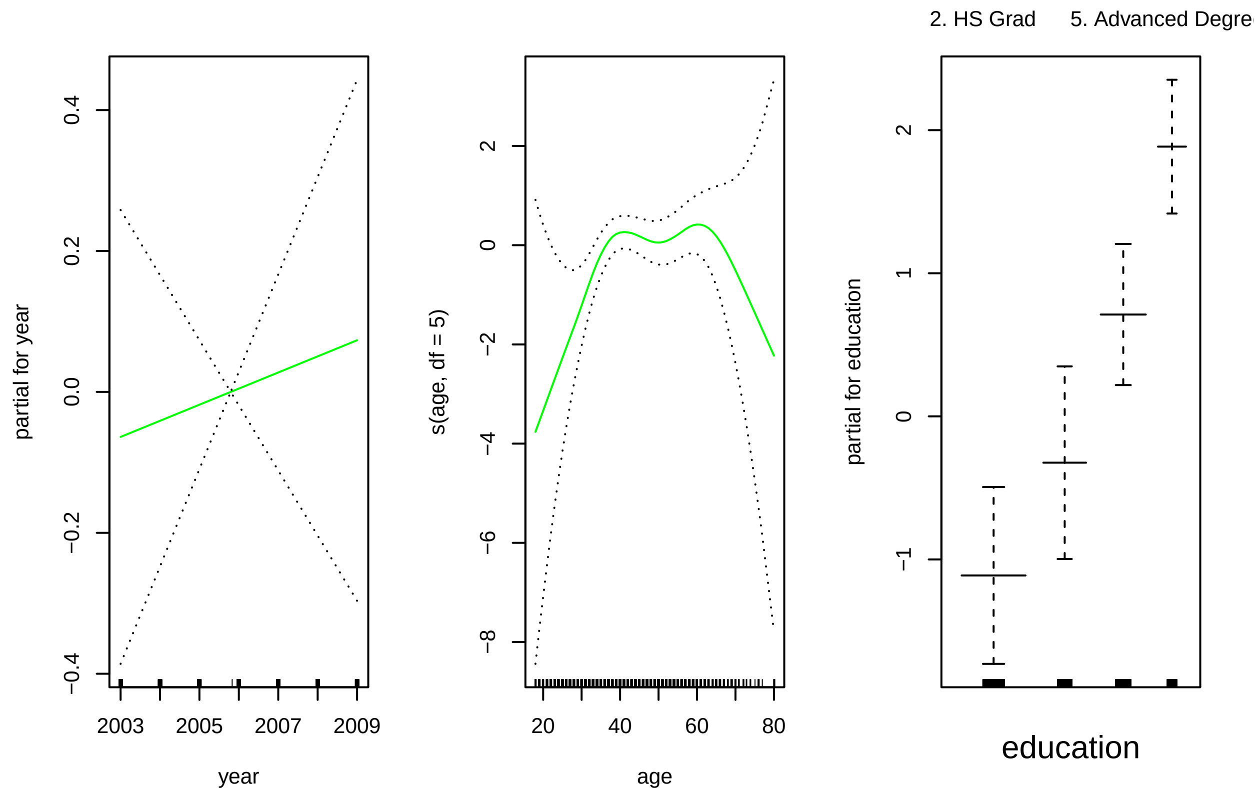

```r
# Fittinga logistic regression GAM using I() function

gam.lr <- gam(I(wage > 250) ~ year + s(age, df = 5) + education,
              family = binomial, data = Wage)

par(mfrow = c(1,3))
plot(gam.lr, se = TRUE, col = "green")
```




```r
# Creeate a table with high earnes in the < HS category

attach(Wage)
table(education, I(wage > 250))
```

```
##
## education            FALSE TRUE
##   1. < HS Grad         268    0
##   2. HS Grad           966    5
##   3. Some College      643    7
##   4. College Grad      663   22
##   5. Advanced Degree   381   45
```

```r
# Fitting a logsitic regression GAM by skipping the education category

gam.lr.s <- gam( I(wage > 250) ~ year + s(age, df = 5) + education,
                 family = binomial, subset = (education != "1. < HS Grad"))

par(mfrow = c(1,3))
plot(gam.lr.s, se = TRUE, col = "green")
```



5.4 Q8 - Perform cross-validation on a simulated data set.

(a) Generate a simulated data set as follows:

```
# Generate a simulated data set

set.seed(1)

x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

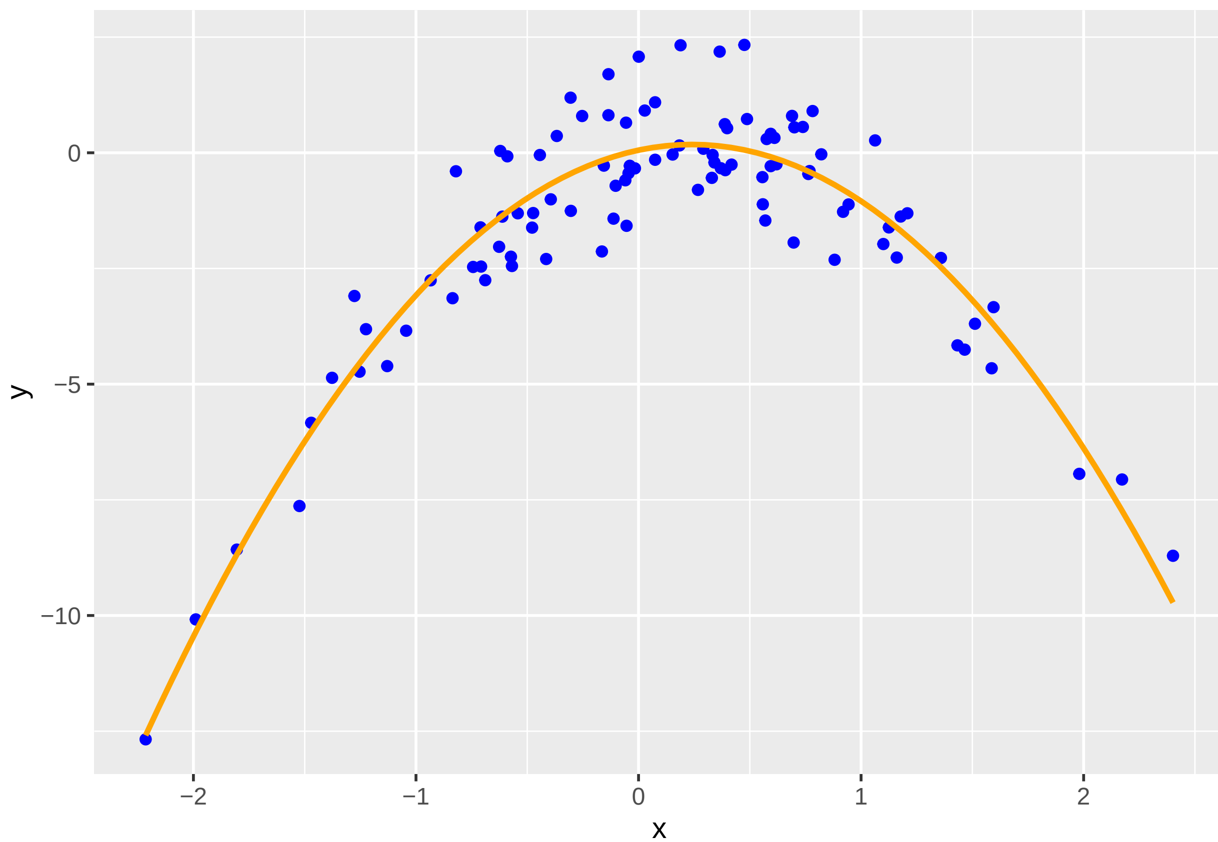
As we have one predictor coefficient and 100 observations (as rnorm(100) is used) , our **n = 1** and **p = 100**

Our equation will be ***Y = x - 2x^2 + e***

(b) Create a scatterplot of X against Y. Comment on what you find.

```
# Scatterplot of X against Y.



myfun <- function(x) {
  x - 2 * x^2

}
```

```r
# Using ggplot2 to plot a scatterplot

library(ggplot2)

ggplot(mapping  = aes(x = x,y = y)) +
  geom_point(colour = "blue") +
  geom_smooth(method = "lm", formula = "y ~ x + I(x^2)", se = FALSE,
              colour = "orange")
```




```r
#
```

As we can see, our equation creates a concave down parabola and and we can notice a little bit of noise at the vertex of our parabola. The points with the support of our curve also suggests the existence of a curved relationship which hints to a quadratic relationship.

(c) Set a random seed, and then compute then LOOCV errors that result from fitting the following four models using least squares:

As we can see from all the models, they start from polynomial order 1 to order 4, so we can use a loop to compute all the LOOCV errors.

Let's create a data frame out of our x and y values.

```r
# Creating a data frame using random seed

set.seed(25)
```

```r
data <- data.frame(x = x, y = y)
```

```r
# Computing LOOCV errors from polynomial order 1 to 4.

loocv.error <- rep(0,4)

for (i in  1:4) {

  glm.fit <- glm(y ~ poly(x, i), data = data)
  loocv.error[i] <- cv.glm(data, glm.fit)$delta[1]

}

poly_order <- c("Order 1", "Order 2", "Order 3", "Order 4")


setNames(loocv.error, poly_order)
```

```
##   Order 1   Order 2   Order 3   Order 4
## 7.2881616 0.9374236 0.9566218 0.9539049
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```r
# Changing seed and repeating the steps

set.seed(50)

loocv.error <- rep(0,4)

for (i in  1:4) {

  glm.fit <- glm(y ~ poly(x, i), data = data)
  loocv.error[i] <- cv.glm(data, glm.fit)$delta[1]

}

poly_order <- c("Order 1", "Order 2", "Order 3", "Order 4")


setNames(loocv.error, poly_order)
```

```
##   Order 1   Order 2   Order 3   Order 4
## 7.2881616 0.9374236 0.9566218 0.9539049
```

We get exact same error values with a random seed and this is because in LOOCV there is no element of randomness i.e. we will always have the same LOOCV error values every time.

(e) Which of the models in (c) had the smallest LOOCV error? Is that what you expected? Explain your answer.

We see that the the lowest value of LOOCV error is at Order 2 which is **0.937** which is not much surprising to see as the scatter plot which we created in (b) already hinted to a quadratic relationship.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results ?

```r
# Looking at coefficients significant for all the order models, can be seen in order 4 polynomial regre

glm.fit4 <- glm(y ~ poly(x, 4), data = data)

summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x, i), data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, i)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, i)3   0.26411    0.95905   0.275    0.784
## poly(x, i)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

6.6 Q9 - Predict the number of applications received using the other variables in the `College` data set.

(a) Split the data set into a training set and a test set.

```r
# Cleaning data and splitting the dataset into train and test data

# Loading the necessary packages

library(ISLR)
```

```
##
## Attaching package: 'ISLR'

## The following object is masked _by_ '.GlobalEnv':
##
##     Hitters
```

```
## The following objects are masked from 'package:ISLR2':
##
##     Auto, Credit
```

```r
library(glmnet)
```

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

```
## Loaded glmnet 4.1-4
```

```r
library(pls)

# Removing NA values

College <- na.omit(College)


set.seed(49)

samp <- sample(c(TRUE, FALSE), nrow(College), replace = TRUE, prob = c(0.7,0.3))

train <- College[samp, ]
test <- College[!samp, ]

y_test <- test$Apps
```

(b) Linear Regression Model

```r
# Fitting a linear model

lm.fit <- lm(Apps ~ ., data = train)

# Predicting the values

lm.pred <- predict(lm.fit, test)

lm.mse <- mean((lm.pred - y_test)^2)

lm.mse
```

```
## [1] 1071111
```

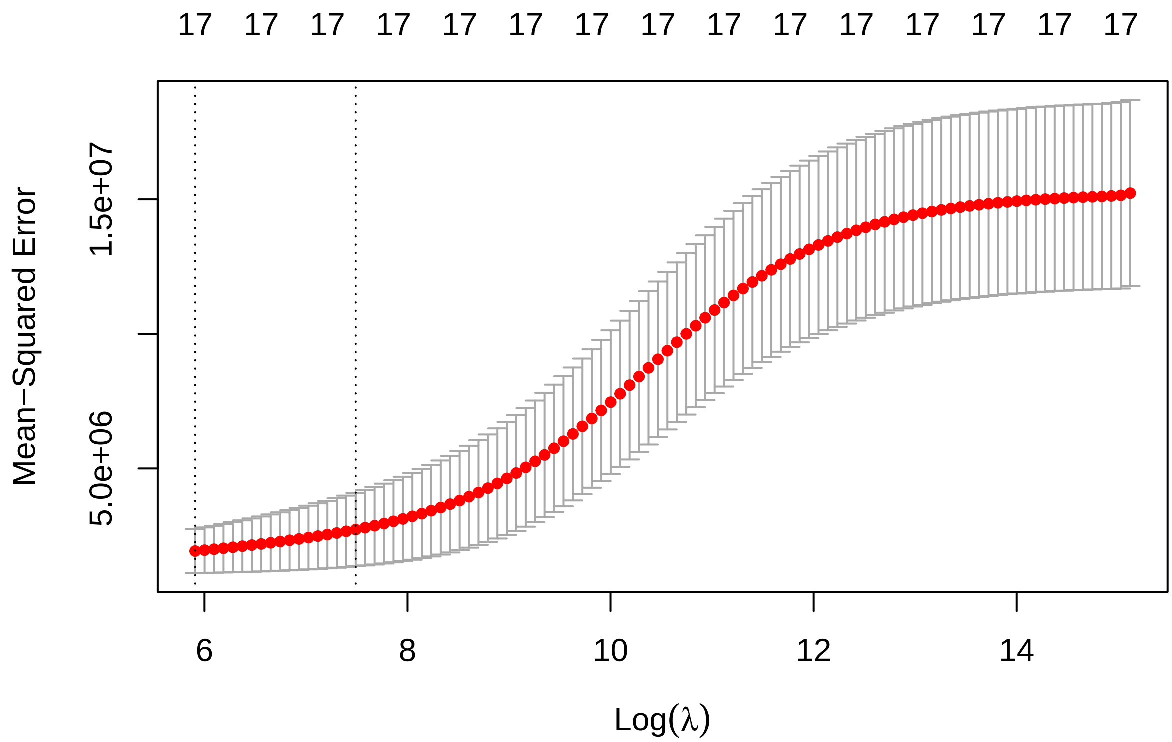
(c) Ridge Regression Model

```r
# Writing a function to calculate the R^2 value

R_square <- function(y, y_pred) {

  y_mean <- mean(y)

  rss <- sum((y - y_pred)^2)
```

```r
  tss <- sum((y - y_mean)^2)

  return (1 - (rss/tss))
}

# Calulcation R^2 value for Linear Model

lm.r2 <- R_square(y_test, lm.pred)

lm.r2
```

```
## [1] 0.9252001
```

```r
# Ridge Regression Model

library(glmnet)

train_mat <- model.matrix(train$Apps ~ ., data = train)

y_train <- train$Apps

test_mat <- model.matrix(test$Apps ~ ., data = test)

y_testR <- test$Apps

set.seed(20)

#ridge.fit <- glmnet(train_mat, y_train, alpha = 0)

par(mfrow = c(1,1))

ridge.cv <- cv.glmnet(train_mat, y_train, alpha = 0)

plot(ridge.cv)
```




```
# Finding the optimal lambda value and fitting the model again using it

opt_lambda <- ridge.cv$lambda.min

ridge.fit <- glmnet(train_mat, y_train, lambda = opt_lambda, alpha = 0)

ridge.fit
```

```
##
## Call:  glmnet(x = train_mat, y = y_train, alpha = 0, lambda = opt_lambda)
##
##   Df %Dev Lambda
## 1 17 90.8  368.1
```

```
# Predicting the values using ridge regression and finding the the test MSE

ridge.pred <- predict(ridge.fit, newx = test_mat, s = opt_lambda)

ridge_mse <- mean((ridge.pred - y_testR)^2)

ridge_mse
```

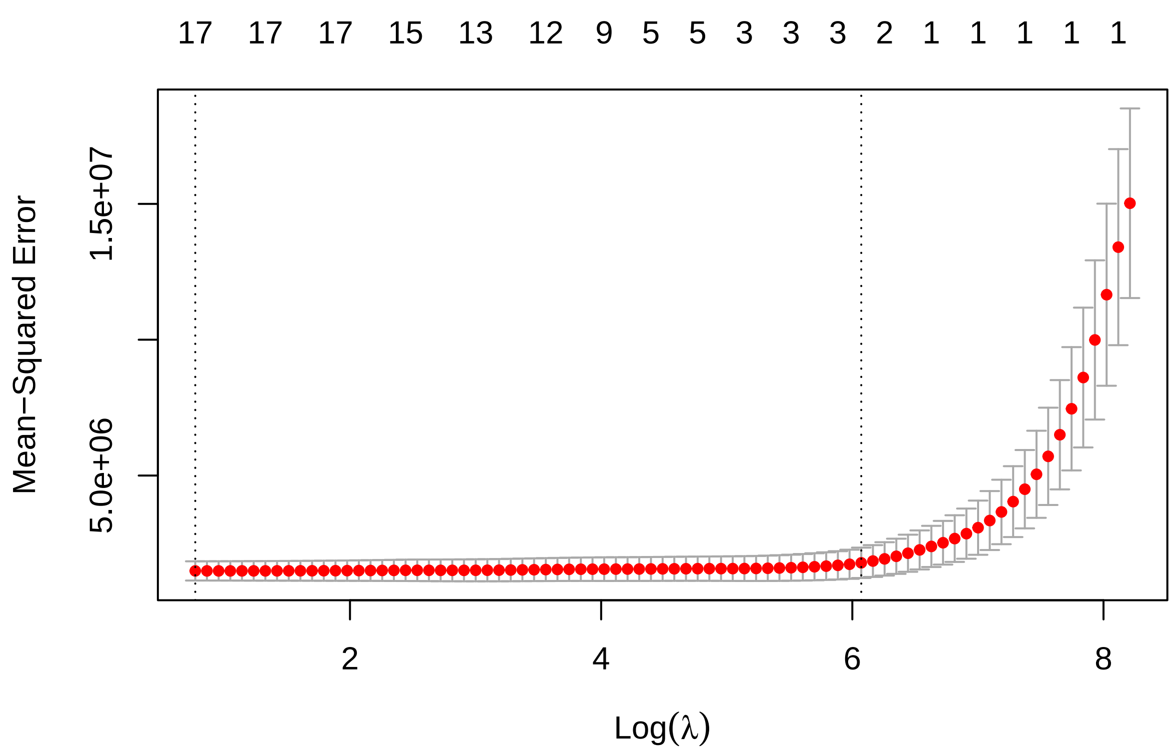
```
## [1] 1033935
```

```
# Calculating R^2 for Ridge regression

ridge.r2 <- R_square(y_testR, ridge.pred)

ridge.r2
```

```
## [1] 0.9277963
```

(d) Lasso Model

```
# Fitting Lasso by setting alpha = 1 in glmnet

set.seed(20)

lasso.cv <- cv.glmnet(train_mat, y_train, alpha = 1)

plot(lasso.cv)
```




```
# Finding the optimnal lamda value and fitting the lasso model again using that value.

lasso_lamda <- lasso.cv$lambda.min

lasso_lamda
```

```
## [1] 2.155763
```

```r
lasso.fit <- glmnet(train_mat, y_train, lambda = lasso_lamda, alpha = 1)

lasso.fit
```

```
##
## Call:  glmnet(x = train_mat, y = y_train, alpha = 1, lambda = lasso_lamda)
##
##   Df  %Dev Lambda
## 1 17 92.99  2.156
```

```r
# Predicting values using the optimal lambda and finding the test MSE

lasso.pred <- predict(lasso.fit, newx = test_mat, s = lasso_lamda)

lasso_mse <- mean((lasso.pred - y_testR)^2)

lasso_mse
```

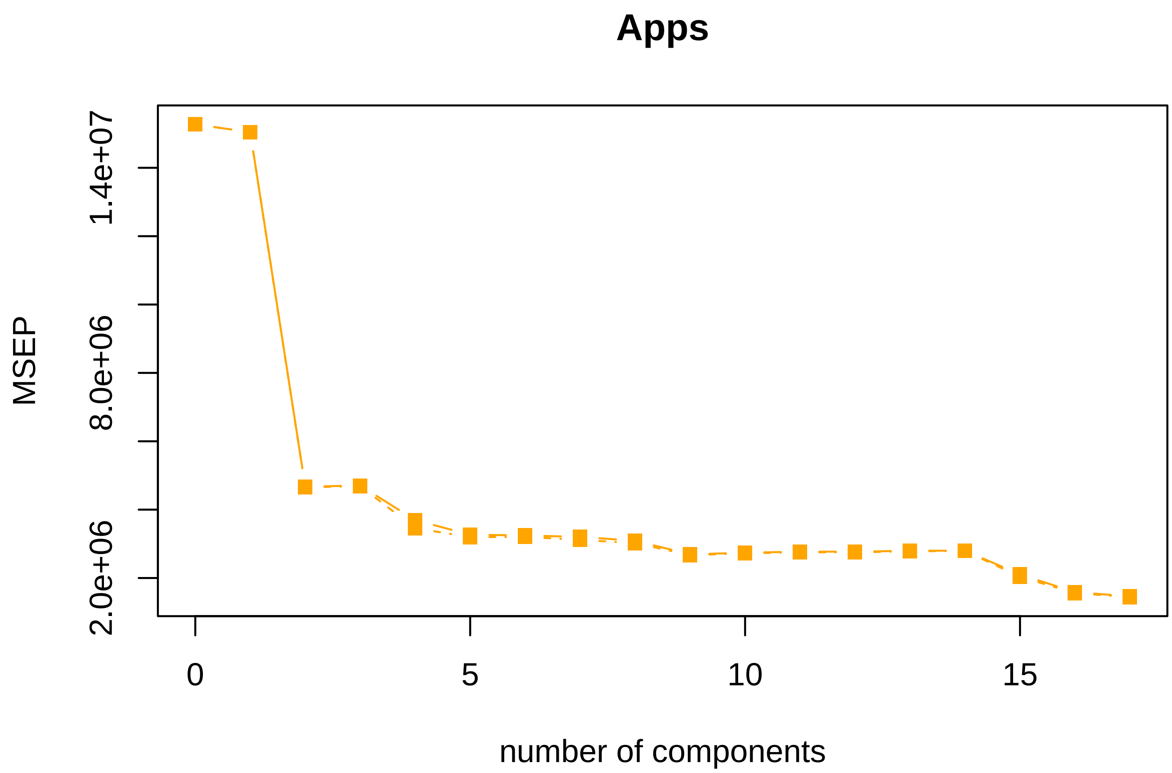
```
## [1] 1062778
```

```r
# calculation R^2 for Lasso Model

lasso.r2 <- R_square(y_testR, lasso.pred)

lasso.r2
```

```
## [1] 0.9257821
```

(e) PCR model

```r
# Fitting PCR mode using pcr

set.seed(20)

pcr.fit <- pcr(Apps ~ ., data = train, scale = TRUE,  validation = "CV")

# Checking for M value by using validation point

validationplot(pcr.fit, val.type = "MSEP", type = "b", col = "orange", pch = 15)
```



We can see the lowest point is at 17 i.e. M = 17

```
# Predicting the values with ncomp = 17
pcr.pred <- predict(pcr.fit, test, ncomp = 17)
pcr_mse <- mean((pcr.pred - test$Apps)^2)
pcr_mse
```

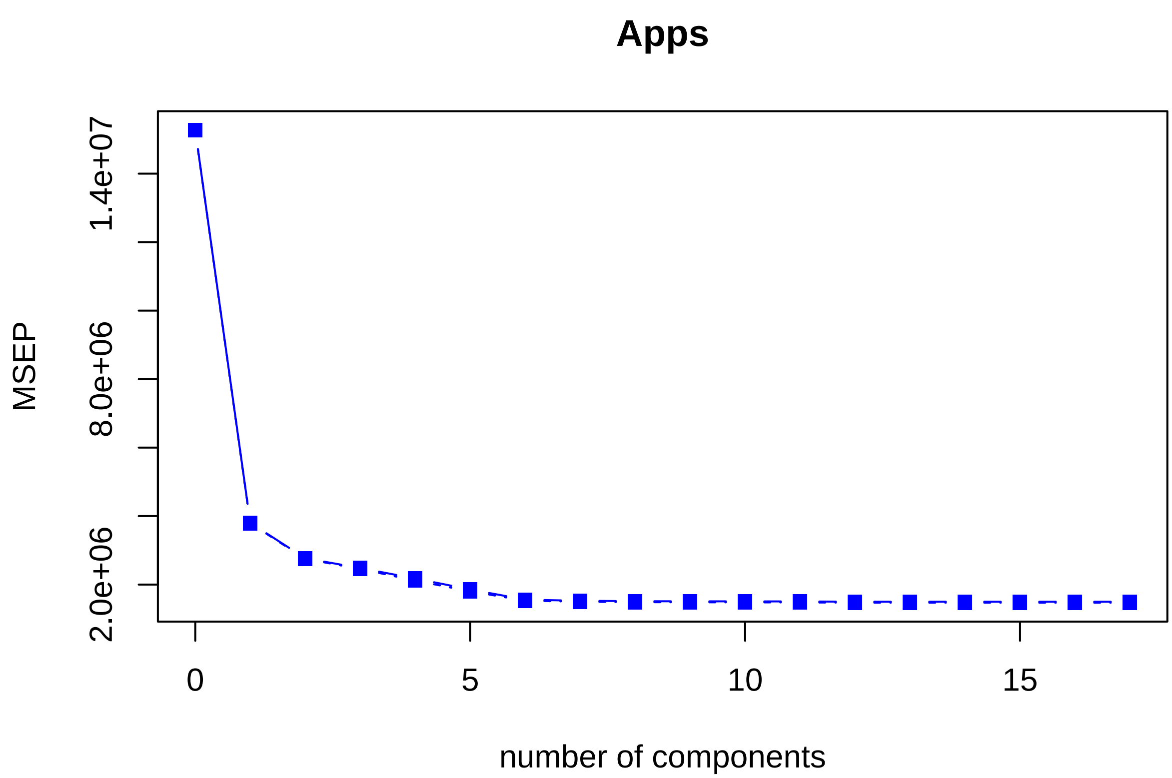
```
## [1] 1071111
```

```
# Calculating R^2 value for PCR
pcr.r2 <- R_square(y_test, pcr.pred)
pcr.r2
```

```
## [1] 0.9252001
```

(f) PLS Model

```r
# Fitting the PLS model

pls.fit <- plsr(Apps ~ ., data = train, scale = TRUE, validation = "CV")


validationplot(pls.fit, val.type = "MSEP", type = "b", col = "blue", pch = 15)
```



We can see the lowest point is at 13 i.e. M = 13

```r
# Predicting the values by using ncomp as 13

pls.pred <- predict(pls.fit, test, ncomp = 13)

pls_mse <- mean((pls.pred - test$Apps)^2)

pls_mse
```

```
## [1] 1072687
```

```r
# Calculating R^2 value for PLS

pls.r2 <- R_square(y_test, pls.pred)

pls.r2
```

```
## [1] 0.9250901
```

```r
# Merging all the MSEs in a data frame

cod <- data.frame(method = c("Linear", "Ridge", "Lasso", "PCR", "PLS"), test.MSE = c(lm.mse, ridge_mse,

cod
```

```
##   method test.MSE  RSquared
## 1 Linear  1071111 0.9252001
## 2  Ridge  1033935 0.9277963
## 3  Lasso  1062778 0.9257821
## 4    PCR  1071111 0.9252001
## 5    PLS  1072687 0.9250901
```

As we can see, the Ridge Regression model gives us the best R^2 with the value approximately **0.9277,** the other models give a similar R^2 value of around **0.925.**

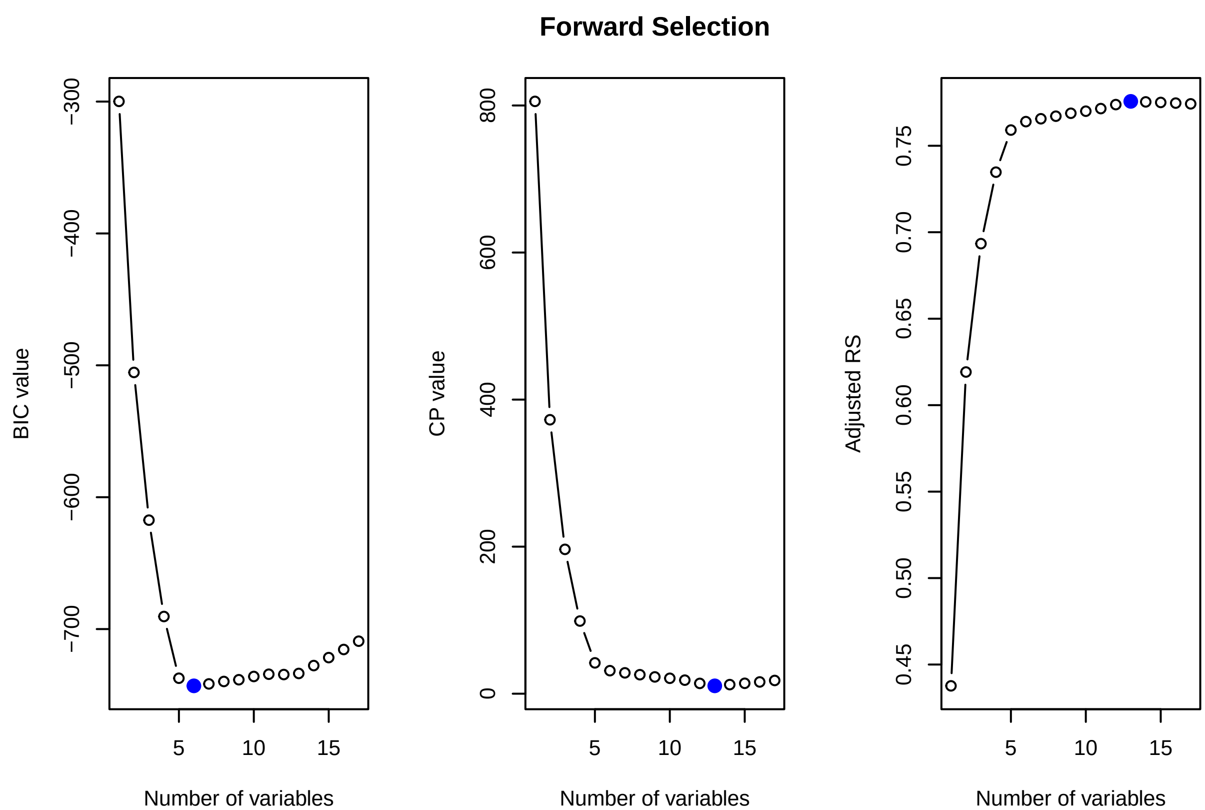
7.9 Q10 - `College` data set GAMs

(a) Split the data into a training set and a test set. Using out-of-state tuition as the response and the other variables as the predictors, perform forward stepwise selection on the training set in order to identify a satisfactory model that uses just a subset of the predictors.

```r
# Cleaning data and splitting the dataset into train and test data

# Loading the necessary packages

library(ISLR)
library(glmnet)
library(gam)
library(leaps)

# Removing NA values

College <- na.omit(College)


set.seed(120)

samp <- sample(c(TRUE, FALSE), nrow(College), replace = TRUE, prob = c(0.7,0.3))

train <- College[samp, ]
test <- College[!samp, ]
```

```r
# Performing forward stepwise selection with out of state tution as the response and other variables as

sub.for <- regsubsets(Outstate ~ ., data = train, nvmax = ncol(College)-1, method = "forward")

sum.for <- summary(sub.for)

# Finding minimum values for Cp and BIC and maximum value for adjusted R2
```

```r
cp.for <- which.min(sum.for$cp)
bic.for <- which.min(sum.for$bic)
ar2.for <- which.max(sum.for$adjr2)

# Plotting using in built plot function of regsubsets()

par(mfrow = c(1,3))

plot(sum.for$bic,xlab="Number of variables",ylab= "BIC value",type="b")

points(bic.for,sum.for$bic[bic.for],col="blue", cex = 2, pch = 20)

plot(sum.for$cp, xlab="Number of variables", ylab= "CP value",type="b", main = "Forward Selection")

points(cp.for, sum.for$cp[cp.for], col="blue", cex = 2, pch = 20)

plot(sum.for$adjr2, xlab="Number of variables", ylab= "Adjusted RS",type="b")

points(ar2.for, sum.for$adjr2[ar2.for], col="blue", cex = 2, pch = 20)
```



As per our graphs, the best BIC score is 6 but we get 13 for both adjusted R square and Cp scores. We'll go with 6 to get our coefficients.

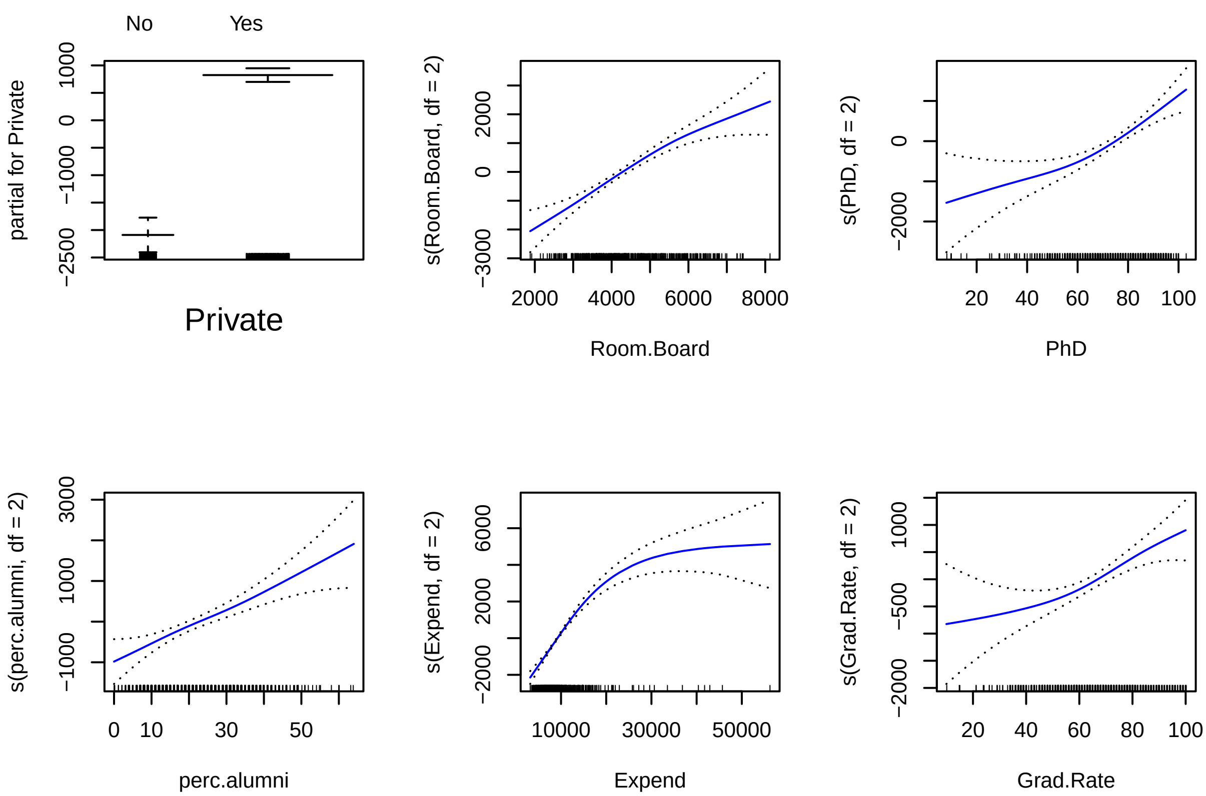
```r
# See coeeficient instances for the best 11-variable models identified by our forward selection
```

```r
for.co <- coef(sub.for, 6)

names(for.co)
```

```
## [1] "(Intercept)" "PrivateYes"  "Room.Board"  "PhD"         "perc.alumni"
## [6] "Expend"      "Grad.Rate"
```

(b) Fit a GAM on the training data, using out-of-state tuition as the response and the features selected in the previous step as the predictors. Plot the results, and explain your findings.

```r
# Fitting a GAM using the predictors selected from forward step

gam.fit <- gam(Outstate ~ Private + s(Room.Board, df = 2) +
                 s(PhD, df = 2) + s(perc.alumni, df = 2) + s(Expend, df = 2) +
                 s(Grad.Rate, df = 2), data = train)

par(mfrow = c(2,3))

plot(gam.fit, se = T, col = "blue")
```



(c) Evaluate the model obtained on the test set, and explain the results obtained

```r
# Predicting the values by passing the test data set

pred.gam <- predict(gam.fit, test)
```

```r
# Calculating test MSE, RMSE and R2 values

gam.mse <- mean((pred.gam - test$Outstate)^2)

cat("TEST MSE :", gam.mse, "\n")
```

```
## TEST MSE : 4626282
```

```r
gam.rmse <- sqrt(gam.mse)

cat("TEST RMSE :", gam.rmse, "\n")
```

```
## TEST RMSE : 2150.879
```

```r
gam.r2 <- R_square(test$Outstate, pred.gam)

cat("R squared :", gam.r2, "\n")
```

```
## R squared : 0.7239991
```

By performing GAM with the obtained 6 features from forward selection, we get a test RMSE of **2150.879** and R^2 value of **0.724** approximately.

Let's perform the same with a linear model to compare whether GAM helps us to improvise us or not.

```r
# Linear model using the 6 predictors

lm.pred1 <- predict(lm(Outstate ~ Private + Room.Board + PhD + perc.alumni +
                         Expend + Grad.Rate, data = train), test)

lm.mse <- mean((lm.pred1 - test$Outstate)^2)

cat("TEST MSE :", lm.mse, "\n")
```

```
## TEST MSE : 5055071
```

```r
lm.rmse <- sqrt(lm.mse)

cat("TEST RMSE :", lm.rmse, "\n")
```

```
## TEST RMSE : 2248.348
```

```r
lm1.r2 <- R_square(test$Outstate, lm.pred1)

cat("R Squared :", lm1.r2, "\n")
```

```
## R Squared : 0.6984178
```

As we compare the R squared acquired from our linear model i.e. **0.698,** we definitely improvise using GAM where we achieve higher R^2 value.