Delivery 1: Linear Model Selection and Regularisation (ISLR Chap. 6)

Mathematics for Big Data

Alfredo Hernández

10 April 2018

Contents

Preamble

First of all, we load packages and custom functions we will use later on:

```r
library(tidyverse)
# SRC: https://github.com/aldomann/maths-for-bigdata/blob/master/libs/multiplot.R
source("../libs/multiplot.R")
```

We are also going to set a knit hook to compile inline R code as characters when rendering the PDF with LaTeX, as well as some display options for figures:

```r
knitr::knit_hooks$set(inline = as.character)
knitr::opts_chunk$set(dpi = 300, fig.align='center')
```

Exercise 3

Suppose we estimate the regression coefficients in a linear regression model by minimising

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right) \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j| \leq s$$

for a particular value of s. For parts (a) through (e), indicate which of i. through v. is correct. Justify your answer.

(a) As we increase s from 0, the training RSS will:

  i. Increase initially, and then eventually start decreasing in an inverted U shape.
  ii. Decrease initially, and then eventually start increasing in a U shape.
  iii. Steadily increase.
  iv. Steadily decrease.
  v. Remain constant.

(b) Repeat (a) for test RSS.

(c) Repeat (a) for variance.

(d) Repeat (a) for (squared) bias.

(e) Repeat (a) for the irreducible error.

Answers

Exercise 3 (a)

(iv) Steadily decreases: as s increases 0, all β increase from 0 to their least square estimate values. Therefore, the training error steadily decreases to the Ordinary Least Square RSS for the training data.

Exercise 3 (b)

(ii) Decrease initially, and then eventually start increasing in a U shape: when $s = 0$, all β are 0, the model has a high test RSS. As s increases, the *beta* coefficients assume non-zero values and the model fits the test data better, resulting in a decrease of the test RSS. In the long run, the *beta* coefficients start to overfit the training data, resulting in an increase of the test RSS.

Exercise 3 (c)

(iii) Steadily increase: variance always increases with fewer constraints. At $s = 0$, the model predicts a constant and has almost no variance. As s increases, the values of the different β become highly dependent on training data, resulting in an increase of the variance.

Exercise 3 (d)

(iv) Steadily decrease: bias always decreases with more model flexibility. WAt $s = 0$, the model predicts a constant and the prediction is far from actual value; thus the bias is high. As s increases, more β become non-zero and thus the model continues to fit training data better, resulting in a decrease of the bias.

Exercise 3 (e)

(v) Remains constant: by definition, the irreducible error is independent of the selected model.

Exercise 5

It is well-known that Ridge regression tends to give similar coefficient values to correlated variables, whereas the Lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting.

Suppose that $n = 2$, $p = 2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$. Furthermore, suppose that $y_1 + y_2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, Ridge regression, or Lasso model is zero: $\hat{\beta}_0 = 0$.

(a) Write out the Ridge regression optimisation problem in this setting.

(b) Argue that in this setting, the Ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$.

(c) Write out the Lasso optimisation problem in this setting.

(d) Argue that in this setting, the Lasso coefficients $\hat{\beta}_1$ and $\hat{\beta}_2$ are not unique; in other words, there are many possible solutions to the optimisation problem in (c). Describe these solutions.

Answers

Exercise 5 (a)

A general form of Ridge regression optimisation looks like

$$\text{Minimise:} \quad \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{p}\hat{\beta}_j x_j)^2 + \lambda\sum_{j=1}^{p}\hat{\beta}_j^2.$$

In this case, $\hat{\beta}_0 = 0$ and $n = p = 2$. So, the optimisation looks like:

$$\boxed{\text{Minimise:} \quad (y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2 + \lambda(\hat{\beta}_1^2 + \hat{\beta}_2^2)}.$$

Exercise 5 (b)

First of all, we set $x_{11} = x_{12} = x_1$ and $x_{21} = x_{22} = x_2$. Then, we expand the previous expression; and we take the partial derivative to $\hat{\beta}_1$ and set equation to 0:

$(\hat{\beta}_1 x_1^2 - x_1 y_1 + \hat{\beta}_2 x_1^2) + (\hat{\beta}_1 x_2^2 - x_2 y_2 + \hat{\beta}_2 x_2^2) + \lambda\hat{\beta}_1 = 0 \Rightarrow \hat{\beta}_1(x_1^2 + x_2^2) + \hat{\beta}_2(x_1^2 + x_2^2) + \lambda\hat{\beta}_1 = x_1 y_1 + x_2 y_2.$

Now we add $2\hat{\beta}_1 x_1 x_2$ and $2\hat{\beta}_2 x_1 x_2$ to both sides of the equation:

$\hat{\beta}_1(x_1^2+x_2^2+2x_1x_2)+\hat{\beta}_2(x_1^2+x_2^2+2x_1x_2)+\lambda\hat{\beta}_1 = x_1y_1+x_2y_2+2\hat{\beta}_1x_1x_2+2\hat{\beta}_2x_1x_2\hat{\beta}_1(x_1+x_2)^2+\hat{\beta}_2(x_1+x_2)^2+\lambda\hat{\beta}_1 = x_1y_1+x_2y_2+2$

and because $x_1 + x_2 = 0$, we can eliminate the first two terms:

$$\lambda\hat{\beta}_1 = x_1 y_1 + x_2 y_2 + 2\hat{\beta}_1 x_1 x_2 + 2\hat{\beta}_2 x_1 x_2.$$

Likewise by taking the partial deritive to $\hat{\beta}_2$, we can get the equation:

$$\lambda\hat{\beta}_2 = x_1 y_1 + x_2 y_2 + 2\hat{\beta}_1 x_1 x_2 + 2\hat{\beta}_2 x_1 x_2.$$

Finally, we see that the left side of the equations for both $\lambda\hat{\beta}_1$ and $\lambda\hat{\beta}_2$ are the same so we have:

$$\lambda\hat{\beta}_1 = \lambda\hat{\beta}_2 \Rightarrow \boxed{\hat{\beta}_1 = \hat{\beta}_2}.$$

Exercise 5 (c)

A general form of Lasso optimisation looks like

$$\text{Minimise:} \quad \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_j)^2 + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|.$$

For Lasso, like in Ridge regression, we have

$$\boxed{\text{Minimise:} \quad (y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2 + \lambda(|\hat{\beta}_1| + |\hat{\beta}_2|)}.$$

Exercise 5 (d)

Here is a geometric interpretation of the solutions for the equation in (c) above. We use the alternate form of Lasso constraints $|\hat{\beta}_1| + |\hat{\beta}_2| < s$.

The Lasso constraint takes the form $|\hat{\beta}_1| + |\hat{\beta}_2| < s$, which when plotted takes the familiar shape of a diamond centred at origin $(0, 0)$.

Next consider the squared optimisation constraint $(y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2$. We use the facts $x_{11} = x_{12}$, $x_{21} = x_{22}$, $x_{11} + x_{21} = 0$, $x_{12} + x_{22} = 0$ and $y_1 + y_2 = 0$ to simplify it to

$$\text{Minimise:} \quad 2(y_1 - (\hat{\beta}_1 + \hat{\beta}_2)x_{11})^2.$$

This optimisation problem has a simple solution: $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$. This is a line parallel to the edge of Lasso-diamond $\hat{\beta}_1 + \hat{\beta}_2 = s$. Now solutions to the original Lasso optimisation problem are contours of the function $(y_1 - (\hat{\beta}_1 + \hat{\beta}_2)x_{11})^2$ that touch the Lasso-diamond $\hat{\beta}_1 + \hat{\beta}_2 = s$.

Finally, as $\hat{\beta}_1$ and $\hat{\beta}_2$ vary along the line $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$, these contours touch the Lasso-diamond edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ at different points. As a result, the entire edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ is a potential solution to the Lasso optimisation problem.

A similar argument can be made for the opposite Lasso-diamond edge: $\hat{\beta}_1 + \hat{\beta}_2 = -s$.

Thus, the Lasso problem does not have a unique solution. The general form of solution is given by two line segments:

$$\boxed{\hat{\beta}_1 + \hat{\beta}_2 = s, \quad \hat{\beta}_1 \geq 0,\, \hat{\beta}_2 \geq 0}, \text{and} \boxed{\hat{\beta}_1 + \hat{\beta}_2 = -s, \quad \hat{\beta}_1 \leq 0,\, \hat{\beta}_2 \leq 0}.$$

Exercise 8

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

(a) Use the `rnorm()` function to generate a predictor X of length $n = 100$, as well as a noise vector ϵ of length $n = 100$.

(b) Generate a response vector Y of length $n = 100$ according to the model

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon,$$

where β_0, β_1, β_2, and β_3 are constants of your choice.

(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 adjusted R^2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained.

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

(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 cross-validation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained.

(f) Now generate a response vector Y according to the model

$$Y = \beta_0 + \beta_7 X^7 + \epsilon,$$

and perform best subset selection and the Lasso. Discuss the results obtained.

Answers

Exercise 8 (a)

To be consistent with our results, we need to set a seed first:

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

Now we define the predictor X and the noise ϵ:

```
X <- rnorm(100)
epsilon <- rnorm(100)
```

Exercise 8 (b)

First of all, we define our β coefficients:

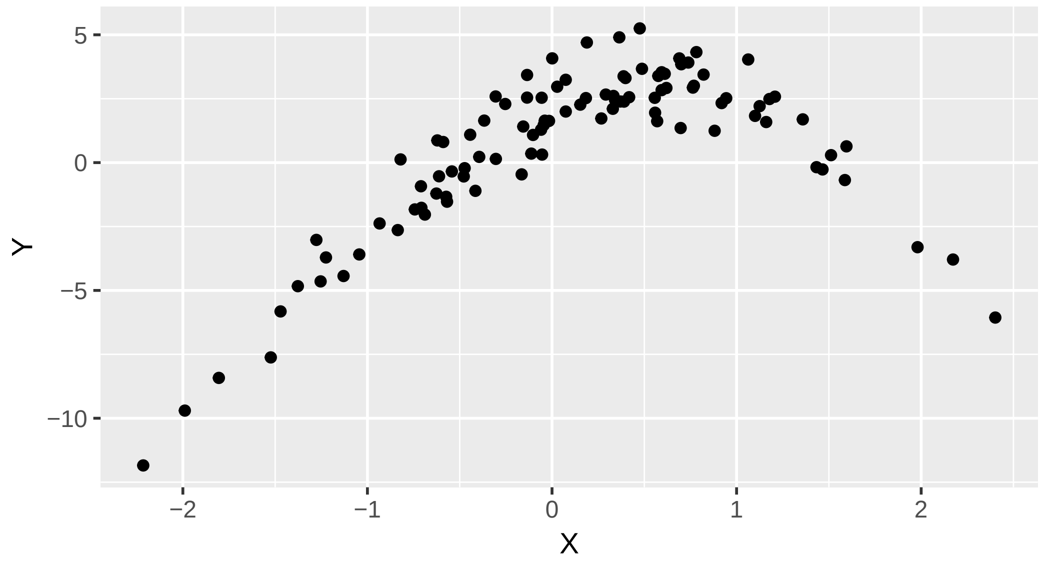
```
beta0 <- 2
beta1 <- 3
beta2 <- -2
beta3 <- -0.3
```

Now we define our Y response:

```
Y <- beta0 + beta1 * X + beta2 * X^2 + beta3 * X^3 + epsilon
```

We can see our function below:

```r
ggplot(data = tibble(X, Y)) +
    geom_point(aes(x = X, y = Y))
```



Exercise 8 (c)

We need the `leaps` package to use the `regsubsets()` function:

```r
library(leaps)
```

Now we use `regsubsets()` to select best model having a polynomial of X of degree 10:

```r
mod.full <- regsubsets(Y ~ poly(X, 10, raw = T),
                                           data = tibble(Y, X),
                                           nvmax = 10)
```

Note that we use the `poly()` function with the `raw = TRUE` option to use raw polynomials instead of orthogonal ones.

Now we can get the summary of the best selection using an exhaustive algorithm (the default method of `regsubsets()`):

```r
(mod.summary <- summary(mod.full))[["outmat"]]
```

```
##             poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
## 1  ( 1 )    " "                   "*"
## 2  ( 1 )    "*"                   "*"
## 3  ( 1 )    "*"                   "*"
## 4  ( 1 )    "*"                   "*"
## 5  ( 1 )    "*"                   "*"
## 6  ( 1 )    "*"                   "*"
## 7  ( 1 )    "*"                   "*"
## 8  ( 1 )    "*"                   "*"
## 9  ( 1 )    "*"                   "*"
## 10  ( 1 )   "*"                   "*"
##             poly(X, 10, raw = T)3 poly(X, 10, raw = T)4
## 1  ( 1 )    " "                   " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  "*"                    " "
## 4  ( 1 )  "*"                    " "
## 5  ( 1 )  "*"                    " "
## 6  ( 1 )  "*"                    " "
## 7  ( 1 )  "*"                    " "
## 8  ( 1 )  "*"                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)5 poly(X, 10, raw = T)6
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  "*"                    " "
## 5  ( 1 )  "*"                    "*"
## 6  ( 1 )  " "                    " "
## 7  ( 1 )  "*"                    "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)7 poly(X, 10, raw = T)8
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  " "                    " "
## 5  ( 1 )  " "                    " "
## 6  ( 1 )  "*"                    "*"
## 7  ( 1 )  " "                    "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  " "                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)9 poly(X, 10, raw = T)10
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  " "                    " "
## 5  ( 1 )  " "                    " "
## 6  ( 1 )  "*"                    " "
## 7  ( 1 )  " "                    "*"
## 8  ( 1 )  "*"                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
```

Now we find the model or subset size for the best C_p, BIC, and adjusted R^2 coefficients:

```
min.cp <- which.min(mod.summary$cp)
min.bic <- which.min(mod.summary$bic)
max.adjr2 <- which.max(mod.summary$adjr2)
```

Let us remember that the previous coefficients are defined as follows:

$$C_p = \frac{1}{n}(\text{RSS} + 2d\hat{\sigma}^2), \quad BIC = \frac{1}{n}(\text{RSS} + \log(n)d\hat{\sigma}^2), \quad \text{Adj}R^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)},$$

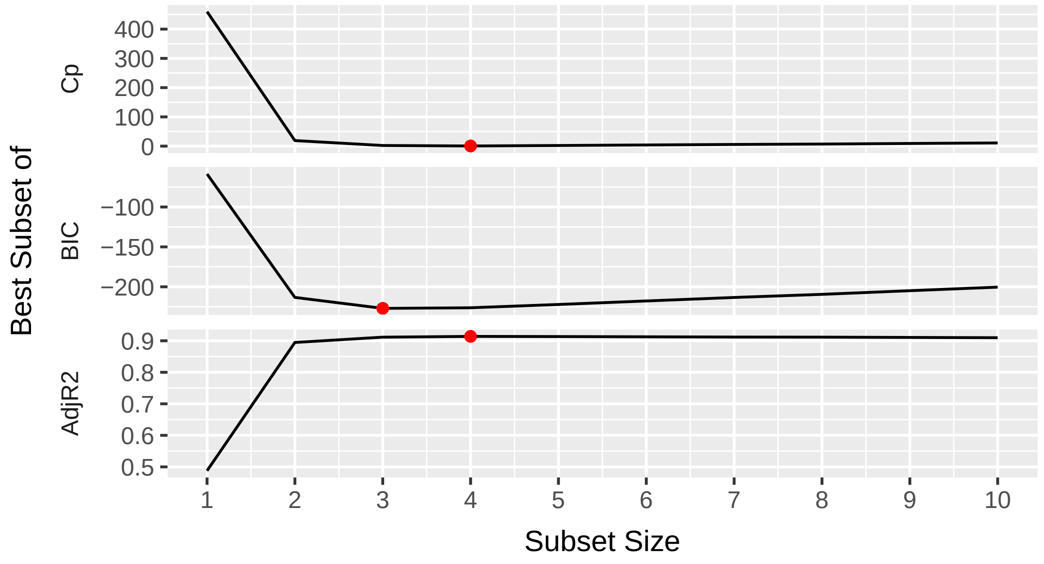
where $\text{TSS} = \sum(y_i - \bar{y})^2$ is the total sum of squares.

Now we define a `ggplot2` function to simplify the process of exploring the results:

```r
plot_statistics <- function(summary, stat, opt.stat, title.str = NULL) {
    variable <- summary[[tolower(stat)]]

    gg <- ggplot(tibble(x = seq_along(variable), y = variable)) +
        geom_line(aes(x = x, y = y)) +
        geom_point(aes(x = opt.stat, y = variable[opt.stat]), colour = "red") +
        scale_x_continuous(breaks = seq(1:10)) +
        labs(x = "Subset Size", y = paste("Best Subset of", stat))

    if (!is.null(title.str)) {
        gg <- gg + labs(title = title.str)
    }

    return(gg)
}
```

Now we compare the three statistics as a function of the subset size:

```r
ggmatrix(list(plot_statistics(mod.summary, "Cp", min.cp),
                            plot_statistics(mod.summary, "BIC", min.bic),
                            plot_statistics(mod.summary, "AdjR2", max.adjr2)),
                 nrow = 3, ncol = 1,
                 yAxisLabels = c("Cp", "BIC", "AdjR2"),
                 xlab = "Subset Size",
                 ylab = "Best Subset of",
                 switch = "both") +
    theme(strip.background = element_rect(fill = "white"),
                strip.placement = "outside")
```



As we can see from the results, with C_p, BIC, and adjusted R^2 criteria, 4, 3, 4-variable models are picked, respectively. The expression for this 4-variable and 3-variable model are the following, respectively:

```r
# 4-variable model
coefficients(mod.full, id = min.cp)
```

```
##           (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##            2.07200775            3.38745596           -2.15424359
## poly(X, 10, raw = T)3 poly(X, 10, raw = T)5
##           -0.74202574            0.08072292
```

```
# 3-variable model
coefficients(mod.full, id = min.bic)
```

```
##           (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##             2.0615072             2.9752803            -2.1237910
## poly(X, 10, raw = T)3
##            -0.2823614
```

As we can see, the BIC statistic selects the correct $\beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$ model, whilst the C_p and adjusted R^2 statistics additionally pick X^5.

Exercise 8 (d)

First of all, we need to fit forward and backward stepwise models to the data using the `regsubsets()` specifying the desired method:

```
mod.fwd <- regsubsets(Y ~ poly(X, 10, raw = T),
                                            data = tibble(Y, X),
                                            nvmax = 10,
                                            method = "forward")
mod.bwd <- regsubsets(Y ~ poly(X, 10, raw = T),
                                            data = tibble(Y, X),
                                            nvmax = 10,
                                            method = "backward")
```

Now we can get the summary of the best selection using both models:

```
(fwd.summary <- summary(mod.fwd))[["outmat"]]
```

```
##           poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
## 1  ( 1 )  " "                   "*"
## 2  ( 1 )  "*"                   "*"
## 3  ( 1 )  "*"                   "*"
## 4  ( 1 )  "*"                   "*"
## 5  ( 1 )  "*"                   "*"
## 6  ( 1 )  "*"                   "*"
## 7  ( 1 )  "*"                   "*"
## 8  ( 1 )  "*"                   "*"
## 9  ( 1 )  "*"                   "*"
## 10  ( 1 ) "*"                   "*"
##           poly(X, 10, raw = T)3 poly(X, 10, raw = T)4
## 1  ( 1 )  " "                   " "
## 2  ( 1 )  " "                   " "
## 3  ( 1 )  "*"                   " "
## 4  ( 1 )  "*"                   " "
## 5  ( 1 )  "*"                   " "
## 6  ( 1 )  "*"                   " "
## 7  ( 1 )  "*"                   " "
## 8  ( 1 )  "*"                   " "
## 9  ( 1 )  "*"                   " "
## 10  ( 1 ) "*"                   "*"
##           poly(X, 10, raw = T)5 poly(X, 10, raw = T)6
## 1  ( 1 )  " "                   " "
## 2  ( 1 )  " "                      " "
## 3  ( 1 )  " "                      " "
## 4  ( 1 )  "*"                      " "
## 5  ( 1 )  "*"                      "*"
## 6  ( 1 )  "*"                      "*"
## 7  ( 1 )  "*"                      "*"
## 8  ( 1 )  "*"                      "*"
## 9  ( 1 )  "*"                      "*"
## 10  ( 1 ) "*"                      "*"
##           poly(X, 10, raw = T)7 poly(X, 10, raw = T)8
## 1  ( 1 )  " "                      " "
## 2  ( 1 )  " "                      " "
## 3  ( 1 )  " "                      " "
## 4  ( 1 )  " "                      " "
## 5  ( 1 )  " "                      " "
## 6  ( 1 )  " "                      " "
## 7  ( 1 )  "*"                      " "
## 8  ( 1 )  "*"                      "*"
## 9  ( 1 )  "*"                      "*"
## 10  ( 1 ) "*"                      "*"
##           poly(X, 10, raw = T)9 poly(X, 10, raw = T)10
## 1  ( 1 )  " "                      " "
## 2  ( 1 )  " "                      " "
## 3  ( 1 )  " "                      " "
## 4  ( 1 )  " "                      " "
## 5  ( 1 )  " "                      " "
## 6  ( 1 )  "*"                      " "
## 7  ( 1 )  "*"                      " "
## 8  ( 1 )  "*"                      " "
## 9  ( 1 )  "*"                      "*"
## 10  ( 1 ) "*"                      "*"
```

```
(bwd.summary <- summary(mod.bwd))[["outmat"]]
```

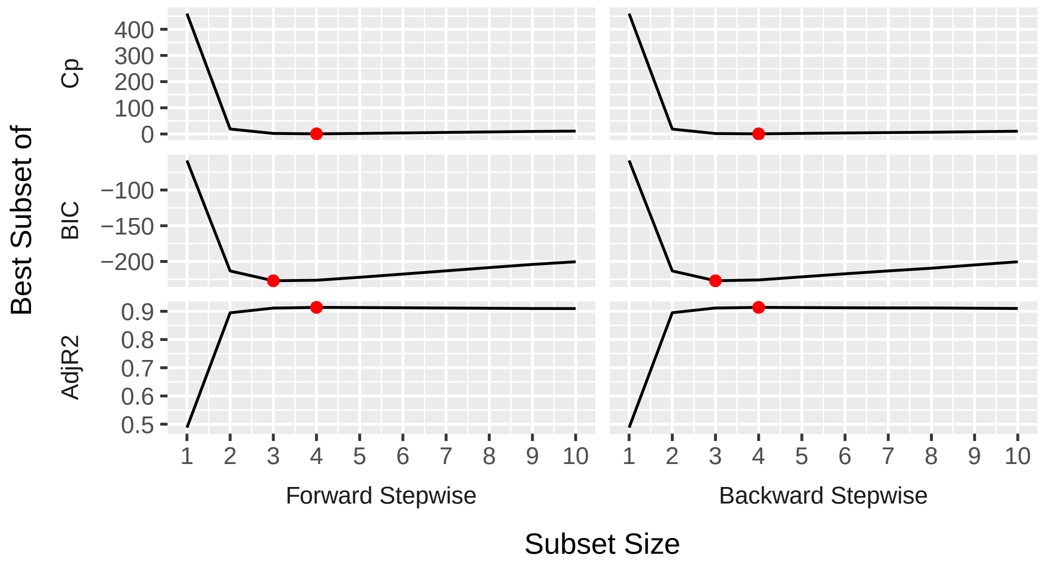
```
##           poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
## 1  ( 1 )  " "                      "*"
## 2  ( 1 )  "*"                      "*"
## 3  ( 1 )  "*"                      "*"
## 4  ( 1 )  "*"                      "*"
## 5  ( 1 )  "*"                      "*"
## 6  ( 1 )  "*"                      "*"
## 7  ( 1 )  "*"                      "*"
## 8  ( 1 )  "*"                      "*"
## 9  ( 1 )  "*"                      "*"
## 10  ( 1 ) "*"                      "*"
##           poly(X, 10, raw = T)3 poly(X, 10, raw = T)4
## 1  ( 1 )  " "                      " "
## 2  ( 1 )  " "                      " "
## 3  ( 1 )  "*"                      " "
## 4  ( 1 )  "*"                      " "
## 5  ( 1 )  "*"                      " "
## 6  ( 1 )  "*"                      " "
## 7  ( 1 )  "*"                      " "
## 8  ( 1 )  "*"                      "*"
## 9  ( 1 )  "*"                      "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)5 poly(X, 10, raw = T)6
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  " "                    " "
## 5  ( 1 )  " "                    " "
## 6  ( 1 )  " "                    " "
## 7  ( 1 )  " "                    "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)7 poly(X, 10, raw = T)8
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  " "                    " "
## 5  ( 1 )  " "                    "*"
## 6  ( 1 )  " "                    "*"
## 7  ( 1 )  " "                    "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  " "                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)9 poly(X, 10, raw = T)10
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  "*"                    " "
## 5  ( 1 )  "*"                    " "
## 6  ( 1 )  "*"                    "*"
## 7  ( 1 )  "*"                    "*"
## 8  ( 1 )  "*"                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
```

As we did before, we find the model or subset size for the best C_p, BIC, and adjusted R^2 coefficients:

```r
min.cp.fwd <- which.min(fwd.summary$cp)
min.bic.fwd <- which.min(fwd.summary$bic)
max.adjr2.fwd <- which.max(fwd.summary$adjr2)
```

```r
min.cp.bwd <- which.min(bwd.summary$cp)
min.bic.bwd <- which.min(bwd.summary$bic)
max.adjr2.bwd <- which.max(bwd.summary$adjr2)
```

```r
ggmatrix(list(# Forward Stepwise
                            plot_statistics(fwd.summary, "Cp", min.cp.fwd),
                            plot_statistics(fwd.summary, "BIC", min.bic.fwd),
                            plot_statistics(fwd.summary, "AdjR2", max.adjr2.fwd),
                            # Backward Stepwise
                            plot_statistics(bwd.summary, "Cp", min.cp.bwd),
                            plot_statistics(bwd.summary, "BIC", min.bic.bwd),
                            plot_statistics(bwd.summary, "AdjR2", max.adjr2.bwd)),
                 byrow = F,
                 nrow = 3, ncol = 2,
                 yAxisLabels = c("Cp", "BIC", "AdjR2"),
                 xAxisLabels = c("Forward Stepwise", "Backward Stepwise"),
                 xlab = "Subset Size",
                 ylab = "Best Subset of",
                 switch = "both") +
    theme(strip.background = element_rect(fill = "white"),
          strip.placement = "outside")
```



As we can see from the results, using the forward and backward stepwise methods we get the same exact results: the C_p and adjusted R^2 statistics pick X^5 additionally to X^0, X^1, X^2, and X^3 whilst for the BIC selects the correct model.

Exercise 8 (e)

We need the `glmnet` package to use Lasso on the data:

```r
library(glmnet)
```

```r
xmat <- model.matrix(Y ~ poly(X, 10, raw = T),
                                         data = tibble(Y,X))[, -1]
mod.lasso <- cv.glmnet(xmat, Y, alpha = 1)
```

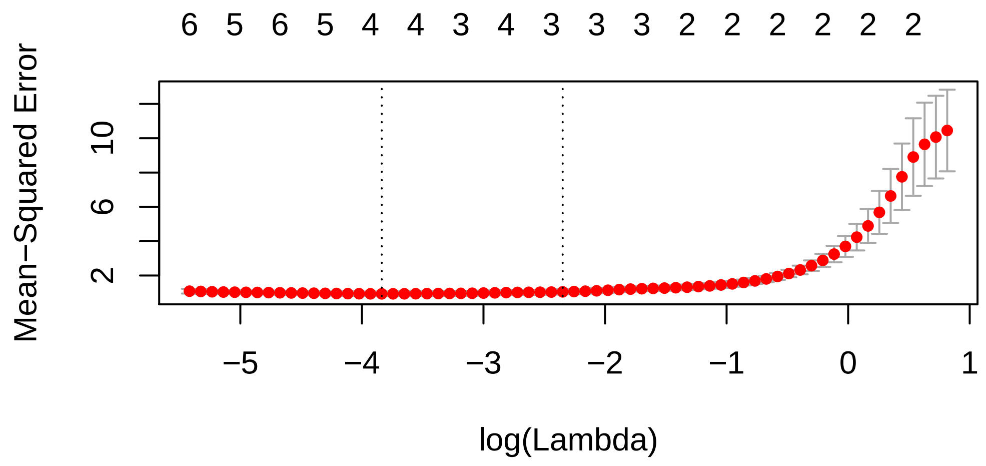
Notice that we use `[, -1]` to get rid of the intercept column.

We can easily find out the optimal λ:

```r
best.lambda <- mod.lasso$lambda.min
```

We can see the Lasso model in the following plot:

```r
plot(mod.lasso)
```



Now, the final step is to use the `predict()` function with the Lasso model to find the coefficients:

```r
predict(mod.lasso, s = best.lambda, type="coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept)             2.0840610046
## poly(X, 10, raw = T)1   2.8761768413
## poly(X, 10, raw = T)2  -2.1677871433
## poly(X, 10, raw = T)3  -0.2569001149
## poly(X, 10, raw = T)4   .
## poly(X, 10, raw = T)5   .
## poly(X, 10, raw = T)6   .
## poly(X, 10, raw = T)7   .
## poly(X, 10, raw = T)8   .
## poly(X, 10, raw = T)9   .
## poly(X, 10, raw = T)10  0.0001286763
```

Let us notice that Lasso predicts $\beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_{10} X^{10}$, but β_{10} is almost negligible.

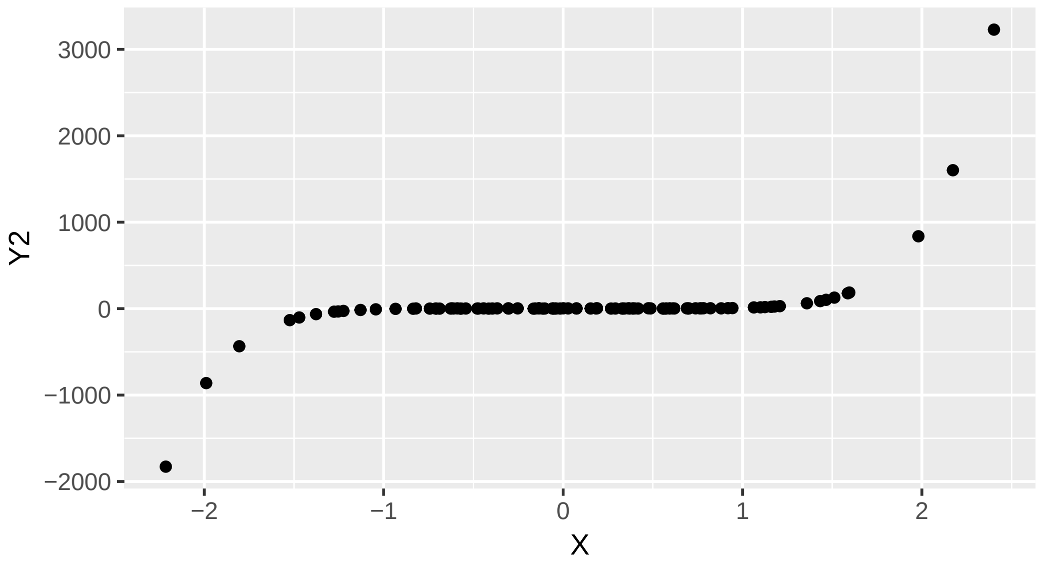
Exercise 8 (f)

First of all, we create a new Y response with different $\beta_7 = 7$:

```r
beta7 <- 7
Y2 <- beta0 + beta7 * X^7 + epsilon
```

We can see our function below:

```r
ggplot(data = tibble(X, Y2)) +
    geom_point(aes(x = X, y = Y2))
```



Now we use `regsubsets()` to select best model and perform a summary of the selected models:

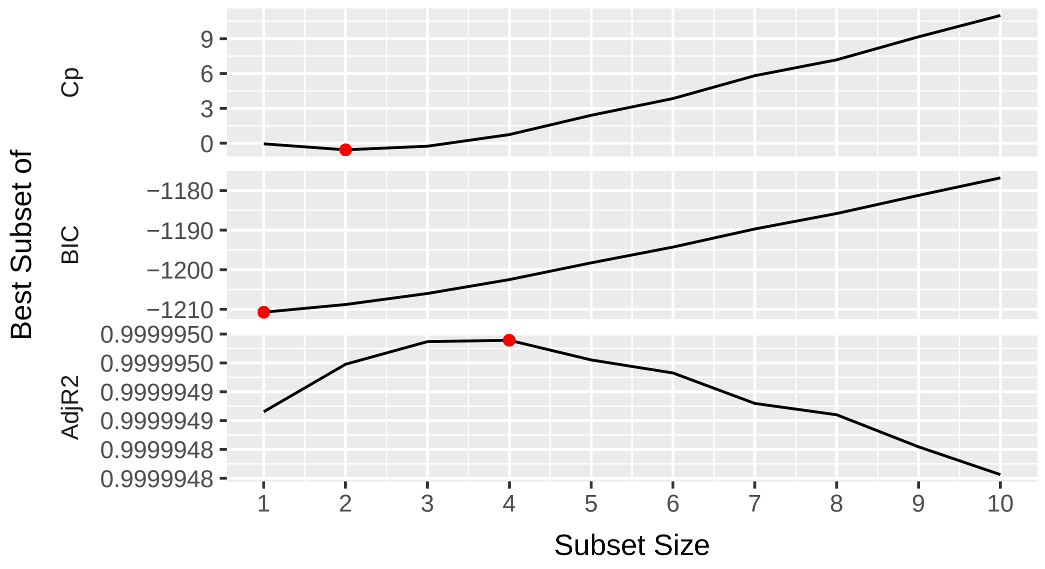
```r
mod2.full <- regsubsets(Y2 ~ poly(X, 10, raw = T),
                                            data = tibble(Y2, X),
                                            nvmax = 10)
(mod2.summary <- summary(mod2.full))[["outmat"]]
```

```
##           poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
## 1  ( 1 )  " "                   " "
## 2  ( 1 )  " "                   "*"
## 3  ( 1 )  " "                   "*"
## 4  ( 1 )  "*"                   "*"
## 5  ( 1 )  "*"                   "*"
## 6  ( 1 )  "*"                   " "
## 7  ( 1 )  "*"                   " "
## 8  ( 1 )  "*"                   "*"
## 9  ( 1 )  "*"                   "*"
## 10  ( 1 ) "*"                   "*"
##           poly(X, 10, raw = T)3 poly(X, 10, raw = T)4
## 1  ( 1 )  " "                   " "
## 2  ( 1 )  " "                   " "
## 3  ( 1 )  " "                   " "
## 4  ( 1 )  "*"                   " "
## 5  ( 1 )  "*"                   "*"
## 6  ( 1 )  "*"                   " "
## 7  ( 1 )  "*"                   " "
## 8  ( 1 )  "*"                   "*"
## 9  ( 1 )  "*"                   "*"
## 10  ( 1 ) "*"                   "*"
##           poly(X, 10, raw = T)5 poly(X, 10, raw = T)6
## 1  ( 1 )  " "                   " "
## 2  ( 1 )  " "                   " "
## 3  ( 1 )  "*"                   " "
## 4  ( 1 )  " "                   " "
## 5  ( 1 )  " "                   " "
## 6  ( 1 )  " "                   "*"
## 7  ( 1 )  "*"                   "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  " "                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)7 poly(X, 10, raw = T)8
## 1  ( 1 )  "*"                    " "
## 2  ( 1 )  "*"                    " "
## 3  ( 1 )  "*"                    " "
## 4  ( 1 )  "*"                    " "
## 5  ( 1 )  "*"                    " "
## 6  ( 1 )  "*"                    "*"
## 7  ( 1 )  "*"                    "*"
## 8  ( 1 )  "*"                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
##           poly(X, 10, raw = T)9 poly(X, 10, raw = T)10
## 1  ( 1 )  " "                    " "
## 2  ( 1 )  " "                    " "
## 3  ( 1 )  " "                    " "
## 4  ( 1 )  " "                    " "
## 5  ( 1 )  " "                    " "
## 6  ( 1 )  " "                    "*"
## 7  ( 1 )  " "                    "*"
## 8  ( 1 )  " "                    "*"
## 9  ( 1 )  "*"                    "*"
## 10  ( 1 ) "*"                    "*"
```

Let us find the model for the best C_p, BIC, and adjusted R^2 coefficients:

```
min2.cp <- which.min(mod2.summary$cp)
min2.bic <- which.min(mod2.summary$bic)
max2.adjr2 <- which.max(mod2.summary$adjr2)
```

```
ggmatrix(list(plot_statistics(mod2.summary, "Cp", min2.cp),
                            plot_statistics(mod2.summary, "BIC", min2.bic),
                            plot_statistics(mod2.summary, "AdjR2", max2.adjr2)),
                 nrow = 3, ncol = 1,
                 yAxisLabels = c("Cp", "BIC", "AdjR2"),
                 xlab = "Subset Size",
                 ylab = "Best Subset of",
                 switch = "both") +
    theme(strip.background = element_rect(fill = "white"),
                strip.placement = "outside")
```



```r
# 2-variable model
coefficients(mod2.full, id = min2.cp)
```

```
##           (Intercept) poly(X, 10, raw = T)2 poly(X, 10, raw = T)7
##             2.0704904            -0.1417084             7.0015552
```

```r
# 1-variable model
coefficients(mod2.full, id = min2.bic)
```

```
##           (Intercept) poly(X, 10, raw = T)7
##               1.95894               7.00077
```

```r
# 4-variable model
coefficients(mod2.full, id = max2.adjr2)
```

```
##           (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##             2.0762524             0.2914016            -0.1617671
## poly(X, 10, raw = T)3 poly(X, 10, raw = T)7
##            -0.2526527             7.0091338
```

What we see from the previous results is that *BIC* statistic picks the most accurate 1-variable model with matching coefficients, whilst other criteria pick additional variables.

Let us use now use Lasso to find the best model:

```r
xmat2 <- model.matrix(Y2 ~ poly(X, 10, raw = T),
                                         data = tibble(Y2, X))[, -1]
mod2.lasso <-  cv.glmnet(xmat2, Y2, alpha = 1)
```

Now we find the optimal lambda:

```r
best2.lambda <- mod2.lasso$lambda.min
```

```r
best.model2 <- glmnet(xmat2, Y2, alpha = 1)
predict(best.model2, s = best.lambda, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)            2.820215
## poly(X, 10, raw = T)1  .
## poly(X, 10, raw = T)2  .
## poly(X, 10, raw = T)3  .
## poly(X, 10, raw = T)4  .
## poly(X, 10, raw = T)5  .
## poly(X, 10, raw = T)6  .
## poly(X, 10, raw = T)7  6.796694
## poly(X, 10, raw = T)8  .
## poly(X, 10, raw = T)9  .
## poly(X, 10, raw = T)10 .
```

As we can see, Lasso also picks the best 1-variable model, but the intercept coefficient does not give a good estimation of the real value of β_0.

As a conclusion, it seems that BIC and Lasso effectively select the 1-variable model, but BIC gives a remarkably better estimation of β_0 and β_7.

Exercise 9

In this exercise, we will 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.

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

(c) Fit a Ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

(d) 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.

(e) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

Answers

Exercise 9 (a)

First of all, we load the `ISLR` package to work with the `College` data set:

```r
library(ISLR)
```

Before doing anything, let's make sure the data set does not have missing data:

```r
table(is.na(College))
```

```
##
## FALSE
## 13986
```

As we can see, the data is all right and there is no need to clean it.

Now we can work with the `College` data set and create a `college.train` (50% of the data) and `college.test` (50% of the data set):

```r
set.seed(10)
college.size <- nrow(College)
train.ind <- sample(1:college.size, college.size/2)

college.train <- College[train.ind, ]
college.test <- College[-train.ind, ]
```

Exercise 9 (b)

Now we will perform a ordinary least squares (OLS) linear model for the number of applications (`Apps`) and calculate the RSS for the test data:

```r
apps.lm <- lm(Apps ~ ., data = college.train)
apps.pred.lm <- predict(apps.lm, newdata = college.test)
apps.lm.test.rss <- mean((college.test$Apps - apps.pred.lm)^2)
```

The RSS for the test data using OLS is 1016995.53.

Exercise 9 (c)

Now we want to perform a Ridge regression on the data:

```r
xmat.train <- model.matrix(Apps ~ ., data=college.train)[, -1]
xmat.test <- model.matrix(Apps ~ ., data=college.test)[, -1]
```

Ridge regression involves tuning a hyperparameter, lambda. `cv.glmnet()` will generate default values, but it is common practice to define our own with the `lambda` argument:

```r
lambdas <- 10^seq(4, -2, by = -.1)
apps.ridge <- cv.glmnet(xmat.train, college.train$Apps, alpha = 0, lambda = lambdas)
best.lambda.ridge <- apps.ridge$lambda.min
```

Now we simply calculate the RSS for the test data:

```r
apps.pred.ridge <- predict(apps.ridge, s = best.lambda.ridge, newx = xmat.test)
apps.ridge.test.rss <- mean((college.test$Apps - apps.pred.ridge)^2)
```

The RSS for the test data using a Ridge regression is 1016908.69, which is slightly lower than the one obtained using OLS.

Exercise 9 (d)

Now we are going to perform a Lasso regression:

```r
apps.lasso <- cv.glmnet(xmat.train, college.train$Apps, alpha = 1, lambda = lambdas)
best.lambda.lasso <- apps.lasso$lambda.min
```

Now we simply calculate the RSS for the test data:

```r
apps.pred.lasso <- predict(apps.lasso, s = best.lambda.lasso, newx = xmat.test)
apps.lasso.test.rss <- mean((college.test$Apps - apps.pred.lasso)^2)
```

The RSS for the test data using a Lasso regression is 923547.78, which is lower than both OLS and Ridge regressions.

Exercise 9 (g)

Let us compare which coefficients are the "less important" according to each of the methods used. First of all, let us analyse the OLS:

```r
apps.lm$coefficients[abs(apps.lm$coefficients) < 0.05]
```

```
## P.Undergrad
##  0.02930371
```

For the shrinkage methods, we will use a custom function to get the coefficients:

```r
get_reduced_vars <- function(pred, value) {
    pred[,1][abs(pred[,1]) < value]
}
```

```r
ridge.coeffs <- predict(apps.ridge, s = best.lambda.ridge, type="coefficients")
lasso.coeffs <- predict(apps.lasso, s = best.lambda.lasso, type="coefficients")
```

```r
get_reduced_vars(ridge.coeffs, 0.05)
```

```
## P.Undergrad
##  0.02936083
```

```
get_reduced_vars(lasso.coeffs, 0.05)
```

```
## F.Undergrad P.Undergrad       Books    Personal
## -0.02957235  0.00000000  0.00000000  0.01583589
```

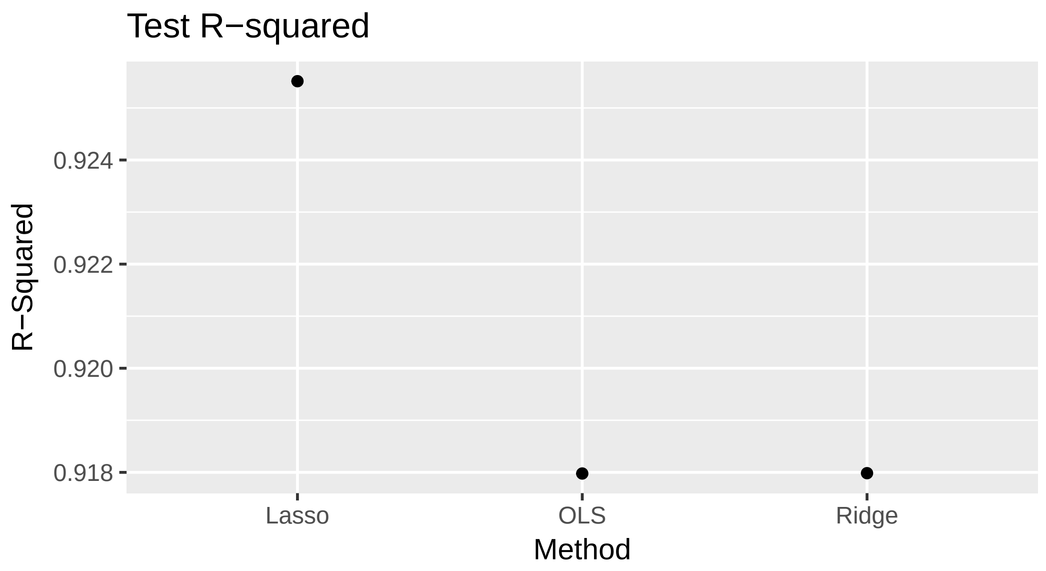
As it seems, both OLS and Ridge reduce the `P.Undergrad` variable to a low value, whilst Lasso totally shrinks it to zero, as well as the `Books` variable.

Here are the test $R^2 = 1 - \frac{RSS}{TSS}$ for all models:

```
test.avg <- mean(college.test$Apps)
apps.test.tss <- mean((college.test$Apps - test.avg)^2)
r2.lm.test <- 1 - apps.lm.test.rss / apps.test.tss
r2.ridge.test <- 1 - apps.ridge.test.rss / apps.test.tss
r2.lasso.test <- 1 - apps.lasso.test.rss / apps.test.tss
```

Let us see the R^2 for the three models:

```
apps.results <- tibble(method = as.factor(c("OLS", "Ridge", "Lasso")),
                                             r2 = c(r2.lm.test, r2.ridge.test, r2.lasso.test))
ggplot(apps.results) +
    geom_point(aes(x = method, y = r2)) +
    labs(title = "Test R-squared", x = "Method", y = "R-Squared")
```



The plot shows that test R^2 for all models are around 0.9, with Lasso having slightly higher test R^2 than OLS and Ridge. We can conclude that all models predict college applications with high accuracy.

Exercise 10

We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set.

(a) Generate a data set with $p = 20$ features, $n = 1000$ observations, and an associated quantitative response vector generated according to the model

$$Y = X \cdot \beta + \epsilon,$$

where β has some elements that are exactly equal to zero.

(b) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

(c) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

(d) Plot the test set MSE associated with the best model of each size.

(e) For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

(f) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

(g) Create a plot displaying $\sqrt{\sum_{j=1}^{p} \left(\beta_j - \hat{\beta}_j^r\right)^2}$ for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

Answers

Exercise 10 (a)

As usual, we start setting a seed to have consistent results:

```r
set.seed(4)
```

First of all, we define the number of features p and the number of observations n:

```r
p <- 20
n <- 1000
```

We define now the set of predictors X and the random error:

```r
X.mat <- matrix(rnorm(n * p), ncol = p)
epsilon <- rnorm(n)
```

We define a random β coefficients vector and set some of the β_i to be 0:

```r
betas <- sample(-5:5, p, replace = T)
betas[sample(1:p, 6, replace = F)] <- 0
betas
```

```
##  [1] -4  0 -1 -1  1 -3  1  3  5  5  0  0 -5  0  4  0  0  5 -1  2
```

This is, we have a model with 14 non-zero variables.

Now we can easily define the Y response vector:

```r
Y <- X.mat %*% betas + epsilon
```

Note that we use the inner product `%*%` for the matrix multiplication.

Exercise 10 (b)

Now we split our `X.mat` data set into a training set containing 10% of the observations, and a test set containing the remaining 90%:

```r
train.ind <- sample(1:n, n*0.1)
X.mat.train <- X.mat[train.ind, ]
X.mat.test <- X.mat[-train.ind, ]
```

Now we do the same for the Y response vector:

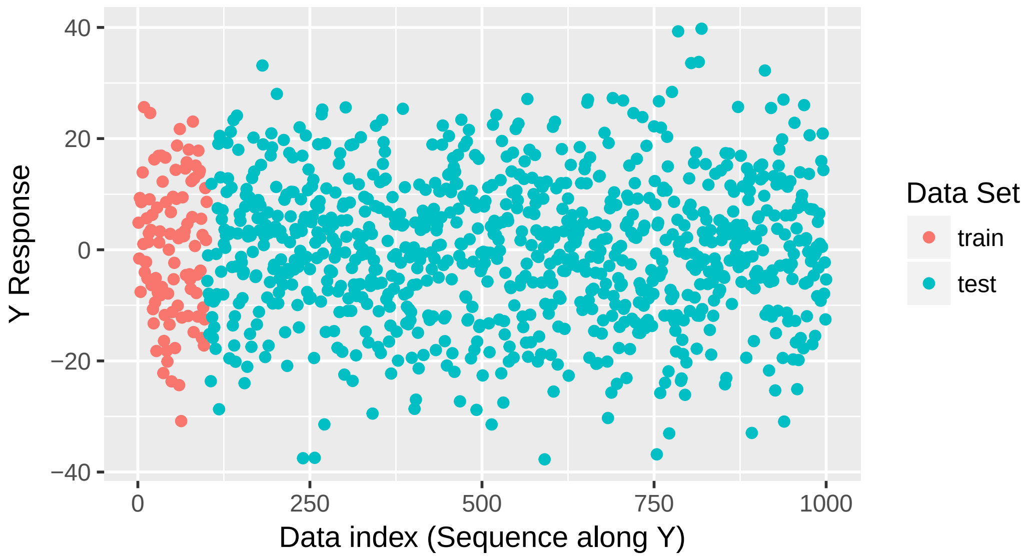
```r
Y.train <- Y[train.ind, ]
Y.test <- Y[-train.ind, ]
train.df <- as_tibble(cbind(y = Y.train, X.mat.train))
test.df <- as_tibble(cbind(y = Y.test, X.mat.test))
```

We will change the names of the columns to ease the identification of the parameters when analysing the different models:

```r
names(train.df) <- c("y", paste0("X", 1:20))
names(test.df) <- c("y", paste0("X", 1:20))
```

Let us have a quick glance at the response Y as a scatter plot (just ordered in sequential order):

```r
ggplot(data = rbind( cbind(train.df, type = rep("train", nrow(train.df))),
                                         cbind(test.df, type = rep("test", nrow(test.df)))) ) +
    geom_point(aes(x = seq_along(y), y = y, colour = type)) +
    labs(x = "Data index (Sequence along Y)", y = "Y Response", colour = "Data Set")
```



Exercise 10 (c)

We need the `leaps` package to use the `regsubsets()` function:

```r
library(leaps)
```

To calculate the MSE, we will need to use the `regsubsets` method for the `predict()` function:

```r
predict.regsubsets <- function(object, newdata, id, ...){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id=id)
  xvars <- names(coefi)
  mat[,xvars]%*%coefi
}
```

Now we use `regsubsets()` to select best model for each size:

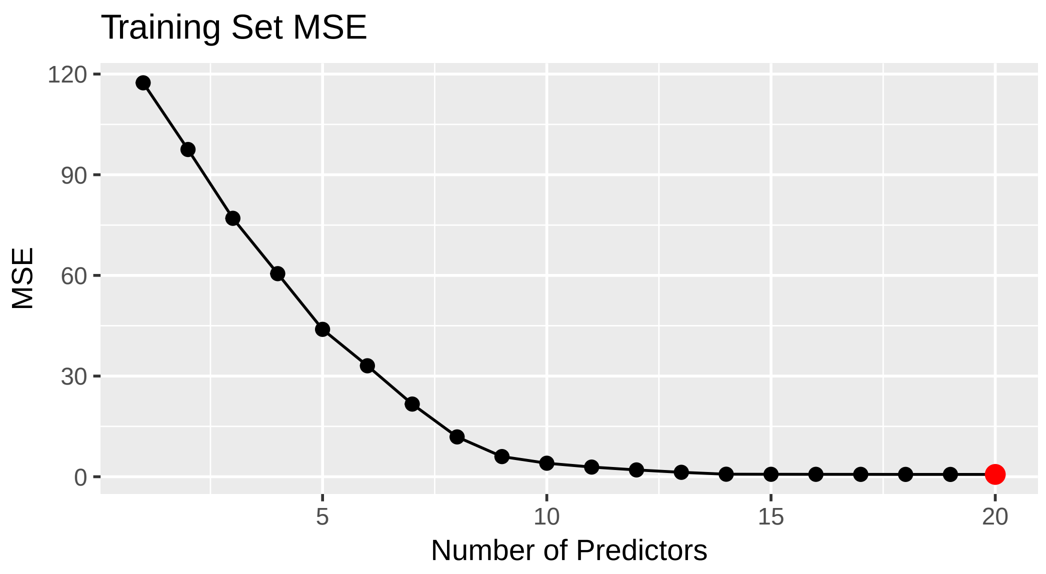
```r
regfit.full = regsubsets(y ~ ., data=train.df, nvmax=p)

err.full.train <- rep(NA, p)
for (i in 1:p) {
  pred.full <- predict(regfit.full, train.df, id=i)
  err.full.train[i] <- mean((train.df$y - pred.full)^2)
}
```

Let us see the training set MSE associated with the best model of each size. We will use a custom `ggplot2` function:

```r
plot_mse <- function(error) {
    error.df <- tibble(num = seq(1:length(error)), err = error)
    min.error.df <- error.df[which.min(error.df$err), ]
    ggplot(error.df)+
        aes(x = num, y = err) +
        geom_line() +
        geom_point(size = 2) +
        geom_point(data = min.error.df, colour = "red", size = 3) +
        labs(x = "Number of Predictors", y = "MSE")
}
```

Let us see the plot:

```r
plot_mse(err.full.train) + labs(title = "Training Set MSE")
```



The minimum for train error should be at maximum predictor count:

```r
which.min(err.full.train)
```

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

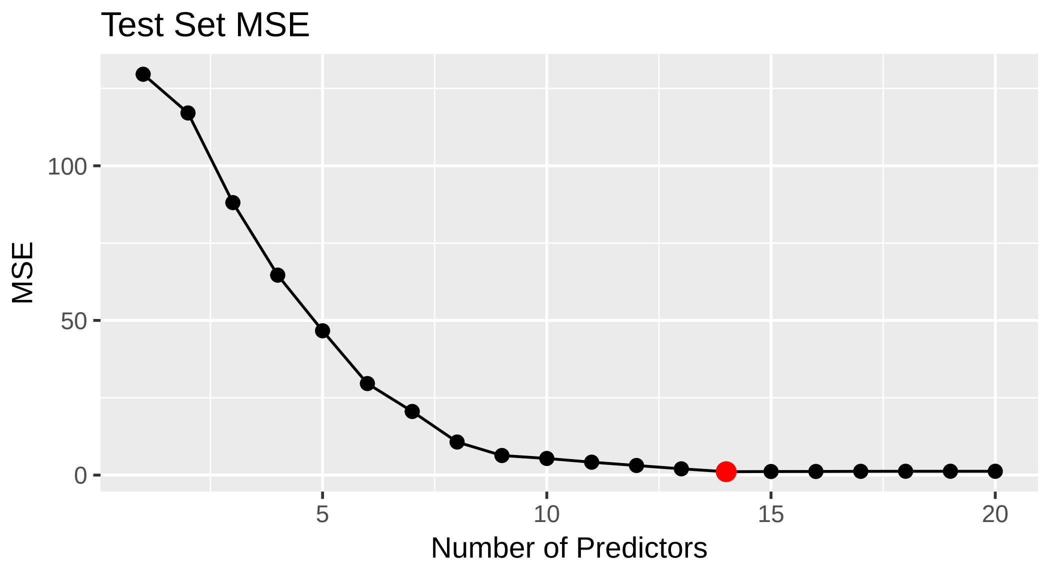
Exercise 10 (d)

Now we repeat the steps done previously, but now for the test set:

```r
err.full.test <- rep(NA, p)
for (i in 1:p) {
  pred.full <- predict(regfit.full, test.df, id=i)
  err.full.test[i] <- mean((test.df$y - pred.full)^2)
}
```

Let us now see the test set MSE associated with the best model of each size:

```r
plot_mse(err.full.test) + labs(title = "Test Set MSE")
```



Exercise 10 (e)

As expected, using the training data we get a model size that falls between using a model with just the intercept or a model containing all the features:

```
which.min(err.full.test)
```

```
## [1] 14
```

Exercise 10 (f)

Let us now see the coefficients of the response selected by the test set and compare them with the real model. For that, we will define a custom function to create a data frame (which will be used also later on):

```
get_coef_df <- function(betas, coefs, sort.opt = T) {
    merge(tibble(beta = names(betas), betas),
                tibble(beta = names(coefs), coefs),
                all.x = T, sort = sort.opt)
}
```

The coefficients are:

```
coef.best <- coef(regfit.full, id = which.min(err.full.test))
names(betas) <- paste0("X", 1:20)

(coefs.df <- get_coef_df(betas, coef.best, F))
```

```
##    beta betas      coefs
## 1    X1    -4 -3.9634243
## 2    X3    -1 -1.1878026
## 3    X4    -1 -0.8298787
## 4    X5     1  1.0845672
## 5    X6    -3 -2.9866262
## 6    X7     1  1.0045155
## 7    X8     3  3.0084885
## 8    X9     5  4.8776870
## 9   X10     5  4.9614235
## 10  X13    -5 -5.0151334
## 11  X15     4  4.0530124
## 12  X18     5  5.0545165
## 13  X19    -1 -0.9600655
## 14  X20     2  1.9208523
## 15  X17     0         NA
## 16   X2     0         NA
## 17  X11     0         NA
## 18  X12     0         NA
## 19  X14     0         NA
## 20  X16     0         NA
```

As we can see, the best subset model successfully selected all the correct predictors, i.e., the following β:

```
coefs.df$beta[which(coefs.df$betas == 0)]
```

```
## [1] "X17" "X2"  "X11" "X12" "X14" "X16"
```

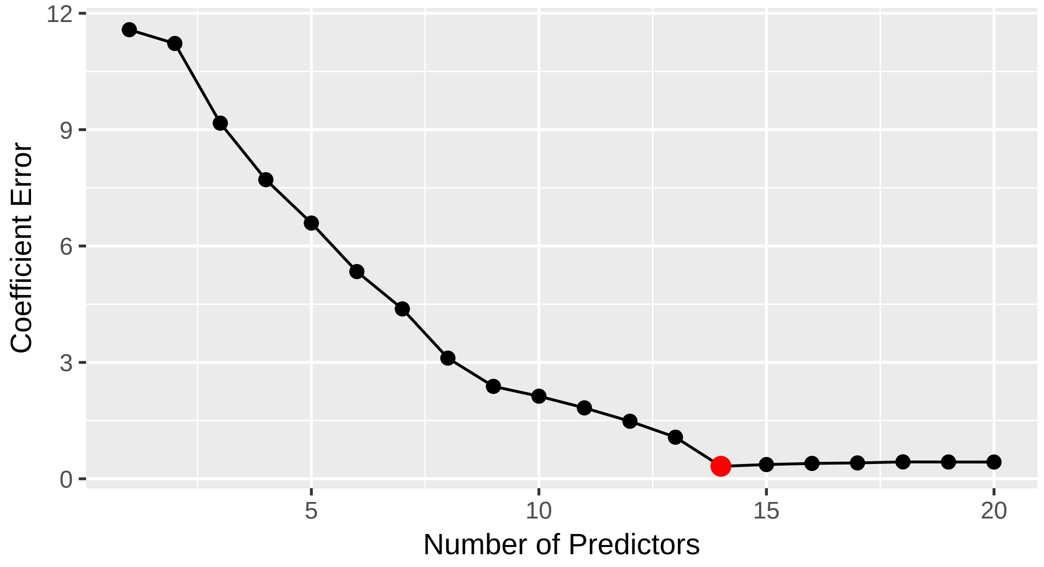
Exercise 10 (g)

Now we calculate the $\sqrt{\sum_{j=1}^{p} (\beta_j - \hat{\beta}_j^r)^2}$ error to analyse the difference between the real model and the best model of each size:

```r
err.coef <- rep(NA, p)
for (i in 1:p) {
  coef.i <- coef(regfit.full, id=i)
  df.err <- get_coef_df(betas, coef.i)
  df.err <- df.err %>%
    mutate(coefs = ifelse(is.na(coefs), 0, coefs) )
  err.coef[i] <- sqrt(sum((df.err$betas - df.err$coefs)^2))
}
```

Let us see a plot of the coefficient error:

```r
plot_mse(err.coef) + labs(y = "Coefficient Error")
```



As we can see, the coefficient error plot shows a very similar plot to the test error plot, although the minimum value is more pronounced.