# Resampling_and_Regularization_Tutorials

Repeat the R codes in 5.3 Lab: Cross-Validation and the Bootstrap and 6.5 Lab: Linear Models and Regularization Methods from 'An Introduction to Statistical Learning' (James et al., 2021), with detailed annotations.

Jason Huang

2025-12-05

## 目錄

# 5.3 Lab: Cross-Validation and the Bootstrap

## 5.3.1 The Validation Set Approach

```
library(ISLR2)
# 設定隨機種子確保結果可以精確重現
set.seed(1)
# 使用 `sample()` 將觀測集分成兩半，從原始的 392 個觀測值中隨機選擇 196 個觀測值。
# 訓練集
train <- sample(392, 196)
# 使用 lm() 函數中的 subset 選項，僅使用與訓練集對應的觀測值來擬合線性迴歸。
lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)
attach(Auto)
# 使用 predict() 函數來估計所有 392 個觀測值的反應值
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit, Auto))[-train]^2)
```

```
[1] 23.26601
```

線性迴歸擬合的估計檢定均方誤差為 23.27。我們可以使用 poly() 函數來估計二次迴歸和三次迴歸的檢定誤差。

```
# 估計二次迴歸
lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,
subset = train)
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
[1] 18.71646
```

```
# 估計三次迴歸
lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,
subset = train)
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
[1] 18.79401
```

這些錯誤率分別為 18.72% 和 18.79%。如果我們選擇不同的訓練集，那麼在驗證集上將會得到略有不同的錯誤率。

```
# 設定隨機種子確保結果可以精確重現
set.seed(2)
# 將觀測資料分為訓練集和驗證集
train <- sample(392, 196)
# 使用 lm() 函數中的 subset 選項，僅使用與訓練集對應的觀測值來擬合線性迴歸。
lm.fit <- lm(mpg ~ horsepower, subset = train)
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit, Auto))[-train]^2)
```

```
[1] 25.72651
```

```
# 估計二次迴歸
lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,
subset = train)
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
[1] 20.43036
```

```
# 估計三次迴歸
lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,
subset = train)
# 使用 mean() 函數來計算驗證集中 196 個觀測值的均方誤差 (MSE)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
[1] 20.38533
```

將觀測資料分成訓練集和驗證集後，我們發現，包含線性項、二次項和三次項的模型的驗證集誤差率分別為 25.73%、20.43% 和 20.39%。
使用二次函數預測油耗的模型優於僅包含線性函數的模型，而幾乎沒有證據表明使用三次函數的模型性能更佳。

### 5.3.2 Leave-One-Out Cross-Validation

可以使用 `glm()` 和 `cv.glm()` 函數自動計算任何廣義線性模型的 leave-one-out cross-validation (LOOCV) 估計值。

```
# 訓練 GLM 模型
glm.fit <- glm(mpg ~ horsepower, data = Auto)
# 看看係數
coef(glm.fit)
```

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

```
# 訓練 LM 模型
lm.fit <- lm(mpg ~ horsepower, data = Auto)
# 看看係數
coef(lm.fit)
```

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

兩者會產生相同的線性迴歸模型。但在本實驗中，我們將使用 `glm()` 進行線性迴歸，因為可以與 `cv.glm()` 函數一起使用!

```
library(boot)
# 訓練 GLM 模型
glm.fit <- glm(mpg ~ horsepower, data = Auto)
# `cv.glm()` 函數會產生一個包含多個元素的清單， delta 向量中的兩個數字包含交叉驗證結果。
cv.err <- cv.glm(Auto, glm.fit)
cv.err$delta
```

```
[1] 24.23151 24.23114
```

我們可以對越來越複雜的多項式擬合重複此過程。
為了實現自動化，我們使用 `for()` 函數啟動一個 for 循環，該循環迭代地擬合階數為 i = 1 到 i = 10 的多項式迴歸，計算對應的交叉驗證誤差，並將其儲存在向量 `cv.error` 的第 i 個元素中。

```
# 建立一個包含 0~10 的向量
cv.error <- rep(0, 10)
# 迭代地擬合階數為 i = 1 到 i = 10 的多項式迴歸
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
  }
# 看看結果
cv.error
```

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

我們可以看到線性擬合和二次擬合之間的估計測試均方誤差急劇下降，但使用更高階多項式並沒有明顯的改善。

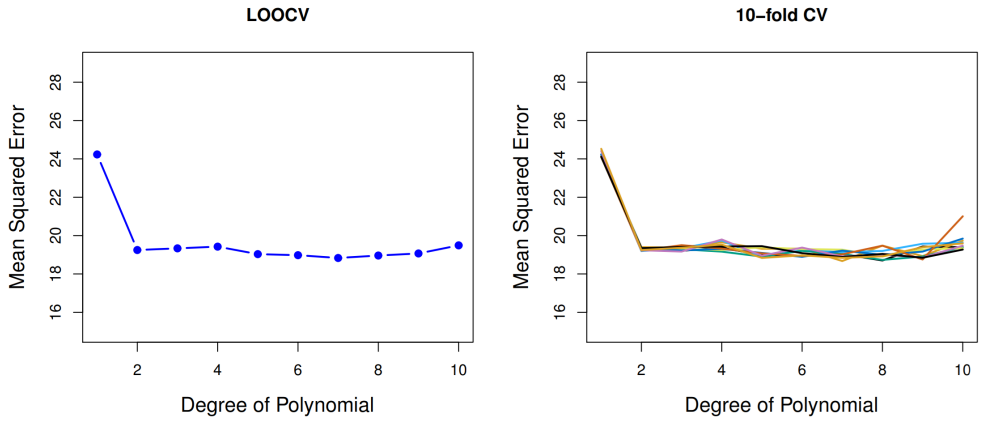



**FIGURE 5.4.** *Cross-validation was used on the* `Auto` *data set in order to estimate the test error that results from predicting* `mpg` *using polynomial functions of* `horsepower`. Left: *The LOOCV error curve.* Right: 10*-fold CV was run nine separate times, each with a different random split of the data into ten parts. The figure shows the nine slightly different CV error curves.*

### 5.3.3 k-Fold Cross-Validation

cv.glm() 函數也可用於實作 k 折交叉驗證。下面我們使用 k = 10，這是在 Auto 資料集上常用的 k 值。原則上對於最小二乘線性模型，LOOCV 的計算時間應該比 k 折交叉驗證更快，因為 LOOCV 可以使用公式。

```
set.seed(17)
# 初始化一個向量，用於存儲對應於一階到十階多項式擬合的交叉驗證誤差。
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
```

當我們執行 k 折交叉驗證時，與 delta 相關的兩個數值則略有不同。第一個數值是標準的 k 折交叉驗證

估計值，第二個數值是經過偏差校正後的版本。在本資料集上，這兩個估計值非常接近。

## 5.3.4 The Bootstrap

### Estimating the Accuracy of a Statistic of Interest

Bootstrap 的一大優點在於它幾乎適用於所有情況，無需複雜的數學計算。
1. 我們需要建立一個函數來計算感興趣的統計量。
2. 使用 boot 套件中的 boot() 函數，透過從資料集中重複抽取有放回的樣本來執行 Bootstrap。

```
# 此函數傳回或輸出基於對參數 index 所索引的觀測值應用公式的    估計值。
alpha.fn <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  (var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y))
  }
# 使用全部 100 個觀測值來估計
alpha.fn(Portfolio, 1:100)
```

```
[1] 0.5758321
```

```
# 使用 sample() 從 1 到 100 的範圍內隨機抽取 100 個觀測值 ( 有放回抽樣 )
set.seed(7)
alpha.fn(Portfolio, sample(100, 100, replace = T))
```

```
[1] 0.5385326
```

```
# 透過多次執行此命令來實現 Bootstrap 分析，記錄所有對應的    估計值，並計算最終的標準差。
# 產生 R = 1000 個    的 Bootstrap 估計值
boot(Portfolio , alpha.fn, R = 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Portfolio, statistic = alpha.fn, R = 1000)


Bootstrap Statistics :
     original       bias    std. error
```

```
t1* 0.5758321 0.0007959475  0.08969074
```

The final output shows that using the original data, $\hat{\alpha} = 0.5758$, and that the bootstrap estimate for SE($\hat{\alpha}$) is 0.0897.

### Estimating the Accuracy of a Linear Regression Model

Bootstrap 可用於評估統計學習方法的係數估計值和預測值的變異性。

```
# 首先建立一個簡單的函數 boot.fn()
# 接收汽車資料集以及一組觀測值的索引，並傳回線性迴歸模型的截距和斜率估計值。
boot.fn <- function(data, index)
  coef(lm(mpg ~ horsepower, data = data, subset = index))
# 看看全部 392 個觀測值
boot.fn(Auto, 1:392)
```

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

boot.fn() 函數也可以用於透過從觀測值中進行有放回的隨機抽樣來建立截距項和斜率項的 bootstrap 估計值。

```
set.seed(1)
# 第一次
boot.fn(Auto, sample(392, 392, replace = T))
```

```
(Intercept)  horsepower
 40.3404517  -0.1634868
```

```
# 第二次
boot.fn(Auto, sample(392, 392, replace = T))
```

```
(Intercept)  horsepower
 40.1186906  -0.1577063
```

```
# 結果不一樣喔!
# 使用 boot() 函數計算 1,000 次 bootstrap 估計的標準誤差
boot(Auto, boot.fn, 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
Call:
boot(data = Auto, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
      original        bias    std. error
t1* 39.9358610  0.0544513229 0.841289790
t2* -0.1578447 -0.0006170901 0.007343073
```

```
# 看看統計分析
summary(lm(mpg ~ horsepower, data = Auto))$coef
```

```
              Estimate  Std. Error   t value      Pr(>|t|)
(Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

boot 函數顯示 SE(β^_0) 的 bootstrap 估計值為 0.84，而 SE(β^_1) 的 bootstrap 估計值為 0.0073。可以使用標準公式計算線性模型中迴歸係數的標準誤差。
summary 計算所得的 β^0 和 β^1 的標準誤差估計值分別為：截距 0.717，斜率 0.0064。這些估計值與使用 bootstrap 方法得到的估計值略有不同。
這是否表示 bootstrap 方法有問題？不!! 線性擬合的殘差會被放大，σ^2 也會被放大。標準公式假設 xi 是固定的，所有變異性都來自誤差項 ϵi 的變異。
bootstrap 不依賴任何這些假設，因此它可能比 summary() 更準確地估計 β^0 和 β^1 的標準誤差。

```
# bootstrap 方法
boot.fn <- function(data, index)
  coef(
lm(mpg ~ horsepower + I(horsepower^2),
data = data, subset = index)
)
set.seed(1)
boot(Auto, boot.fn, 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
```

```
        original        bias     std. error
t1* 56.900099702  3.511640e-02 2.0300222526
t2* -0.466189630 -7.080834e-04 0.0324241984
t3*  0.001230536  2.840324e-06 0.0001172164
```

```
# 線性擬合方法
summary(
lm(mpg ~ horsepower + I(horsepower^2), data = Auto)
)$coef
```

```
                    Estimate   Std. Error   t value      Pr(>|t|)
(Intercept)     56.900099702 1.8004268063  31.60367 1.740911e-109
horsepower      -0.466189630 0.0311246171 -14.97816  2.289429e-40
I(horsepower^2)  0.001230536 0.0001220759  10.08009  2.196340e-21
```

# 6.5 Lab: Linear Models and Regularization Methods

## 6.5.1 Subset Selection Methods

### Best Subset Selection

將最佳子集選擇方法應用於擊球手資料集。我們希望根據棒球運動員上一年表現相關的各種統計數據來預測他們的薪水。

```
library(ISLR2)
# 看看有哪些東西
names(Hitters)
```

```
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
 [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
[13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
[19] "Salary"    "NewLeague"
```

```
# 看看維度
dim(Hitters)
```

```
[1] 322  20
```

```
# 使用 `is.na()` 函數來辨識缺失的觀測值
sum(is.na(Hitters$Salary))
```

```
[1] 59
```

```
# 我們看到有 59 名球員的薪資數據缺失
# na.omit() 函數會刪除所有包含缺失值的行
Hitters <- na.omit(Hitters)
# 再次看看維度
dim(Hitters)
```

```
[1] 263  20
```

```
# 沒缺失了!!
sum(is.na(Hitters))
```

```
[1] 0
```

```
library(leaps)
# 透過識別包含給定數量預測變數的最佳模型來執行最佳子集選擇
# 其中「最佳」由 RSS 量化
regfit.full <- regsubsets(Salary ~ ., Hitters)
# summary() 輸出每個模型大小的最佳變數集
summary(regfit.full)
```

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., Hitters)
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
Runs           FALSE      FALSE
RBI            FALSE      FALSE
Walks          FALSE      FALSE
Years          FALSE      FALSE
CAtBat         FALSE      FALSE
CHits          FALSE      FALSE
CHmRun         FALSE      FALSE
CRuns          FALSE      FALSE
CRBI           FALSE      FALSE
CWalks         FALSE      FALSE
LeagueN        FALSE      FALSE
DivisionW      FALSE      FALSE
PutOuts        FALSE      FALSE
Assists        FALSE      FALSE
```

```
Errors          FALSE      FALSE
NewLeagueN      FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
6  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   " "    " "   "*"
7  ( 1 ) " "   "*"  " "   " "  " " " "*"   " "   "*"    "*"   "*"    " "   " "
8  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   "*"    "*"   " "
         CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
```

星號表示給定變數包含在對應的模型中。預設情況下，regsubsets() 僅報告結果，最多報告到最佳八變量模型。但能使用 nvmax 選項，傳回所需的任意數量的變數。這裡擬合一個最多 19 個變數的模型。

```
# 擬合一個最多 19 個變數的模型
regfit.full <- regsubsets(Salary ~ ., data = Hitters,
nvmax = 19)
# 儲存每個模型大小的最佳變數集
reg.summary <- summary(regfit.full)
# 輸出看看
names(reg.summary)
```

```
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
reg.summary$rsq
```

```
 [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
 [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
[15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

同時繪製所有模型的 RSS、調整後的 $R^2$、Cp 和 BIC 值，將有助於我們決定選擇哪個模型!!

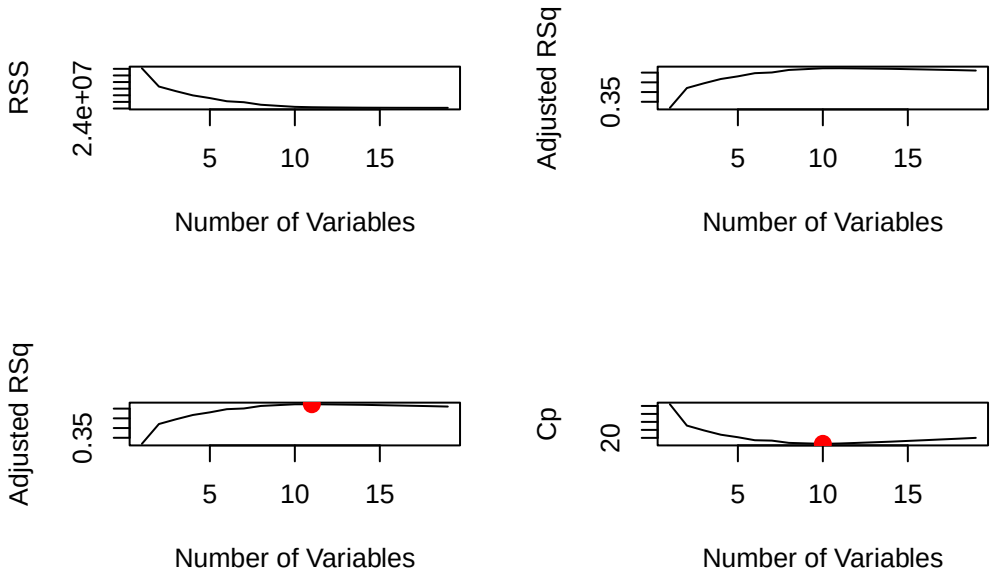
```
# 建立 2*2 的圖
par(mfrow = c(2, 2))
# RSS 圖
plot(reg.summary$rss, xlab = "Number of Variables",
ylab = "RSS", type = "l")
# 調整後的 R² 圖
plot(reg.summary$adjr2, xlab = "Number of Variables",
ylab = "Adjusted RSq", type = "l")
which.max(reg.summary$adjr2)
```

```
[1] 11
```

```
plot(reg.summary$adjr2, xlab = "Number of Variables",
ylab = "Adjusted RSq", type = "l")
points(11, reg.summary$adjr2[11], col = "red", cex = 2,
pch = 20)
# Cp 圖
plot(reg.summary$cp, xlab = "Number of Variables",
ylab = "Cp", type = "l")
which.min(reg.summary$cp)
```

```
[1] 10
```

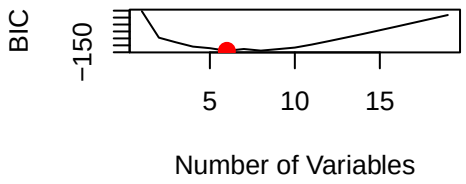
```
points(10, reg.summary$cp[10], col = "red", cex = 2,
pch = 20)
```




```
which.min(reg.summary$bic)
```

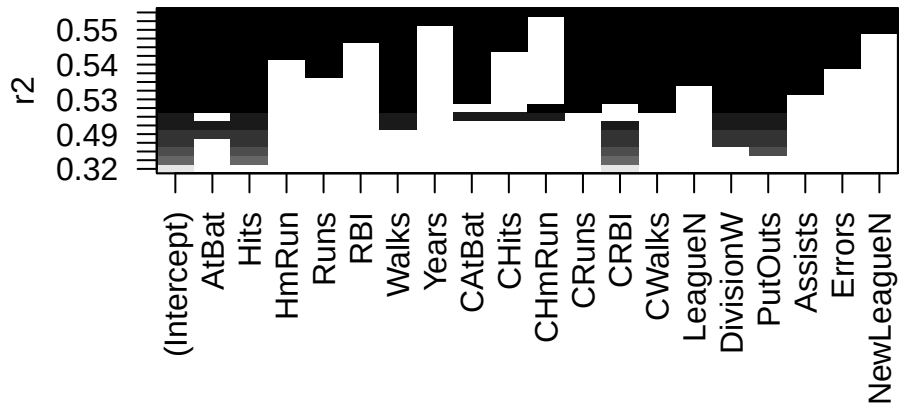
```
[1] 6
```

```
# BIC 圖
plot(reg.summary$bic, xlab = "Number of Variables",
ylab = "BIC", type = "l")
points(6, reg.summary$bic[6], col = "red", cex = 2,
pch = 20)
```

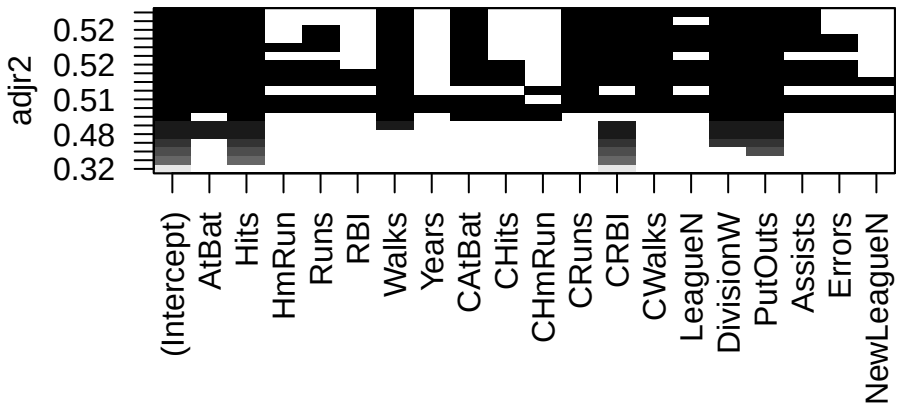



regsubsets() 函數內建了一個 plot() 指令，可用於顯示給定預測變數數量的最佳模型的選定變量，並根據 BIC、Cp、調整後的 $R^2$ 或 AIC 進行排序。
每個圖的第一行都包含一個黑色方塊，代表根據與該統計量關聯的最優模型選擇的每個變數。

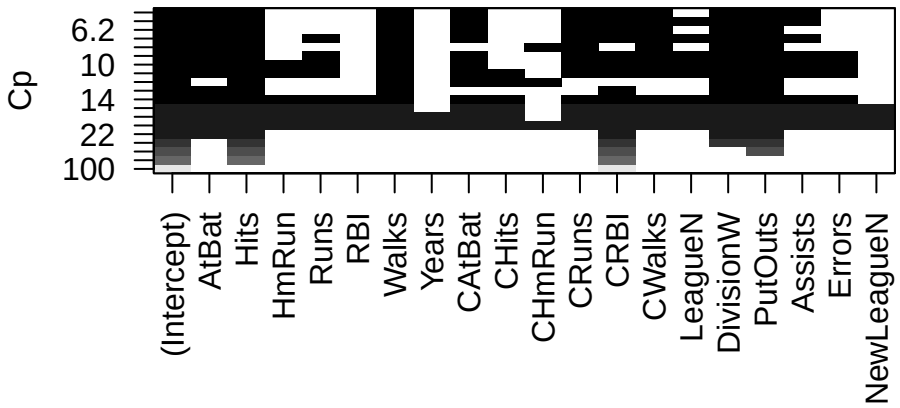
```
plot(regfit.full, scale = "r2")
```



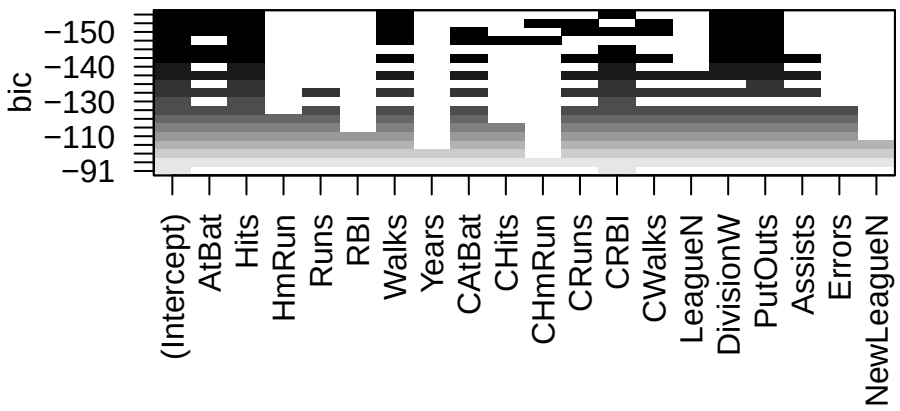
```
plot(regfit.full, scale = "adjr2")
```




```
plot(regfit.full, scale = "Cp")
```




```
plot(regfit.full, scale = "bic")
```




```
coef(regfit.full, 6)
```

```
 (Intercept)        AtBat         Hits        Walks         CRBI    DivisionW
  91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338
     PutOuts
   0.2643076
```

### Forward and Backward Stepwise Selection

我們也可以使用 regsubsets() 函數執行前向逐步選擇或後向逐步選擇，使用參數 method = "forward" 或 method = "backward" 。

```
# 前向逐步選擇
regfit.fwd <- regsubsets(Salary ~ ., data = Hitters,
nvmax = 19, method = "forward")
summary(regfit.fwd)
```

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
```

```
Runs           FALSE      FALSE
RBI            FALSE      FALSE
Walks          FALSE      FALSE
Years          FALSE      FALSE
CAtBat         FALSE      FALSE
CHits          FALSE      FALSE
CHmRun         FALSE      FALSE
CRuns          FALSE      FALSE
CRBI           FALSE      FALSE
CWalks         FALSE      FALSE
LeagueN        FALSE      FALSE
DivisionW      FALSE      FALSE
PutOuts        FALSE      FALSE
Assists        FALSE      FALSE
Errors         FALSE      FALSE
NewLeagueN     FALSE      FALSE
1 subsets of each size up to 19
Selection Algorithm: forward
          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 )  " "   " "  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
2  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
3  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
4  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
5  ( 1 )  "*"   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
6  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    " "   "*"
7  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    " "   "*"
8  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    "*"   "*"
9  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
10  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
11  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
12  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
13  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
14  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
15  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    "*"   " "    "*"   "*"
16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   " "   "*"    "*"   " "    "*"   "*"
17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   " "   "*"    "*"   " "    "*"   "*"
18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
```

```
2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
7  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```
# 後向逐步選擇
regfit.bwd <- regsubsets(Salary ~ ., data = Hitters,
nvmax = 19, method = "backward")
summary(regfit.bwd)
```

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "backward")
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
Runs           FALSE      FALSE
RBI            FALSE      FALSE
Walks          FALSE      FALSE
Years          FALSE      FALSE
CAtBat         FALSE      FALSE
CHits          FALSE      FALSE
CHmRun         FALSE      FALSE
CRuns          FALSE      FALSE
CRBI           FALSE      FALSE
```

```
CWalks          FALSE      FALSE
LeagueN         FALSE      FALSE
DivisionW       FALSE      FALSE
PutOuts         FALSE      FALSE
Assists         FALSE      FALSE
Errors          FALSE      FALSE
NewLeagueN      FALSE      FALSE
1 subsets of each size up to 19
Selection Algorithm: backward
          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 )  " "   " "  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
2  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
3  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
4  ( 1 )  "*"   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
5  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    "*"   " "
6  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    "*"   " "
7  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    "*"   " "
8  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    "*"   "*"
9  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
10  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
11  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
12  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
13  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
14  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
15  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    "*"   " "    "*"   "*"
16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   " "   "*"    "*"   " "    "*"   "*"
17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   " "   "*"    "*"   " "    "*"   "*"
18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
4  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
5  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
7  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
```

```
11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```
# 比較三者的選擇結果
coef(regfit.full, 7)
```

```
 (Intercept)         Hits        Walks       CAtBat        CHits       CHmRun
  79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073    1.4420538
   DivisionW      PutOuts
-129.9866432    0.2366813
```

```
coef(regfit.fwd, 7)
```

```
 (Intercept)        AtBat         Hits        Walks         CRBI       CWalks
 109.7873062   -1.9588851    7.4498772    4.9131401    0.8537622   -0.3053070
   DivisionW      PutOuts
-127.1223928    0.2533404
```

```
coef(regfit.bwd, 7)
```

```
 (Intercept)        AtBat         Hits        Walks        CRuns       CWalks
 105.6487488   -1.9762838    6.7574914    6.0558691    1.1293095   -0.7163346
   DivisionW      PutOuts
-116.1692169    0.3028847
```

使用前向逐步選擇法時，最佳單變數模型僅包含 CRBI，而最佳雙變數模型在此基礎上也包含 Hits。對於此資料集，最佳單變數模型到最佳六變數模型在最佳子集選擇法和前向選擇法中均相同。
然而透過前向逐步選擇法、後向逐步選擇法和最佳子集選擇法辨識出的最佳七變數模型卻各不相同。

### Choosing Among Models Using the Validation-Set Approach and Cross-Validation

現在我們將考慮如何使用驗證集和交叉驗證方法來實現這一點，為了使這些方法能夠準確估計測試誤差，我們必須僅使用訓練觀測值來執行模型擬合的所有步驟，包括變數選擇。因此，確定給定大小的最佳模型必須僅使用訓練觀測值。

```
set.seed(1)
# 拆分為訓練集和測試集
train <- sample(c(TRUE, FALSE), nrow(Hitters),
replace = TRUE)
test <- (!train)
# 將 regsubsets() 應用於訓練集去進行最佳子集選擇
regfit.best <- regsubsets(Salary ~ .,
data = Hitters[train, ], nvmax = 19)
# 根據測試資料建立模型矩陣
test.mat <- model.matrix(Salary ~ ., data = Hitters[test, ])
# 初始化一向量，存入 val.errors
val.errors <- rep(NA, 19)
# 從 regfit.best 中提取該大小的最佳模型的係數，形成預測值並計算測試均方誤差 (MSE)。
for (i in 1:19) {
  coefi <- coef(regfit.best, id = i)
  pred <- test.mat[, names(coefi)] %*% coefi
  val.errors[i] <- mean((Hitters$Salary[test] - pred)^2)
}
# 看看 errors 的變化
val.errors
```

```
 [1] 164377.3 144405.5 152175.7 145198.4 137902.1 139175.7 126849.0 136191.4
 [9] 132889.6 135434.9 136963.3 140694.9 140690.9 141951.2 141508.2 142164.4
[17] 141767.4 142339.6 142238.2
```

```
# 我們發現包含七個變數的模型是最佳模型
which.min(val.errors)
```

```
[1] 7
```

```
# 看看是哪七個
coef(regfit.best, 7)
```

```
 (Intercept)        AtBat         Hits        Walks        CRuns       CWalks
  67.1085369   -2.1462987    7.0149547    8.0716640    1.2425113   -0.8337844
   DivisionW      PutOuts
-118.4364998    0.2526925
```

regsubsets() 函數沒有 predict() 方法，我們可以把上面的步驟記錄下來，自己寫一個 predict 方法。最後我們對完整資料集執行最佳子集選擇，並選擇最佳的七變數模型，使用完整資料集對於獲得更準確的係數估計至關重要。

```
# 自己寫一個 predict 方法
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
  }
# 對完整資料集執行最佳子集選擇
regfit.best <- regsubsets(Salary ~ ., data = Hitters,
nvmax = 19)
# 選擇最佳的七變數模型
coef(regfit.best, 7)
```

```
 (Intercept)         Hits        Walks       CAtBat        CHits       CHmRun
  79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073    1.4420538
   DivisionW      PutOuts
-129.9866432    0.2366813
```

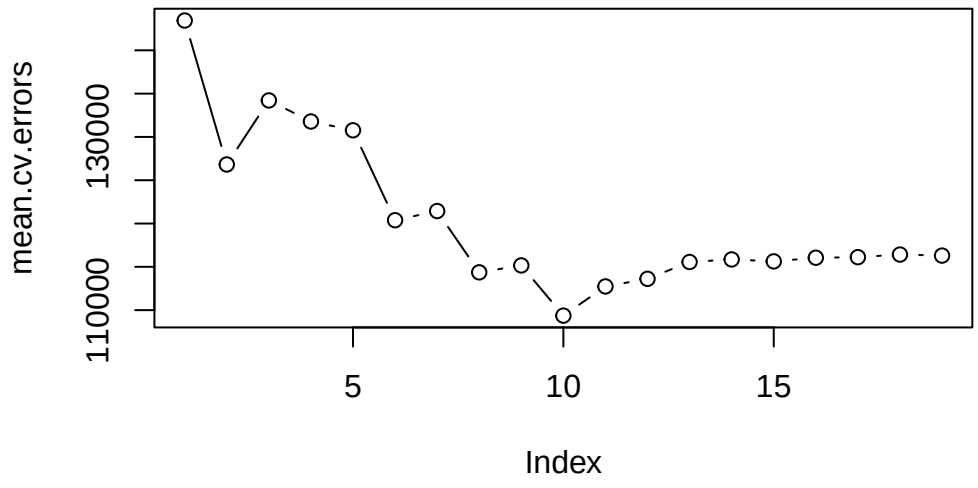
我們發現在完整資料集上表現最佳的七變數模型與在訓練集上表現最佳的七變數模型的變數集不同。現在我們嘗試使用交叉驗證在不同規模的模型中進行選擇，這種方法比較複雜，因為我們必須在每個訓練集中執行最佳子集選擇。

```
# 設定 K-fold 交叉驗證的折數 (K=10)
k <- 10
# 取得資料集總列數 (樣本數)
n <- nrow(Hitters)
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# 將每個觀測值分配到 k = 10 個折疊中的一個
folds <- sample(rep(1:k, length = n))
# 建立一個 10x19 的矩陣來儲存誤差 (10 個折疊, 最多 19 個變數)
cv.errors <- matrix(NA, k, 19,
dimnames = list(NULL, paste(1:19)))
# 依序輪流將第 j 折當作「驗證集」，其餘當「訓練集」
for (j in 1:k) {
    # 在訓練集 (排除第 j 折的資料) 上執行「最佳子集選擇法」
    best.fit <- regsubsets(Salary ~ .,
                           data = Hitters[folds != j, ],
                           nvmax = 19)
```

```
    # 測試不同變數數量 (i = 1 到 19 個變數) 的模型表現
    for (i in 1:19) {
        # 對驗證集 (第 j 折) 進行預測
        pred <- predict(best.fit, Hitters[folds == j, ], id = i)
        # 計算均方誤差 (MSE) 並存入矩陣對應位置
        cv.errors[j, i] <- mean((Hitters$Salary[folds == j] - pred)^2)
    }
}
# 計算每一行 (不同變數數量) 在 10 次驗證中的平均誤差 (CV Error)
mean.cv.errors <- apply(cv.errors, 2, mean)
# 顯示平均誤差數值
mean.cv.errors
```

```
       1        2        3        4        5        6        7        8
143439.8 126817.0 134214.2 131782.9 130765.6 120382.9 121443.1 114363.7
       9       10       11       12       13       14       15       16
115163.1 109366.0 112738.5 113616.5 115557.6 115853.3 115630.6 116050.0
      17       18       19
116117.0 116419.3 116299.1
```

```
# 設定繪圖版面為 1x1
par(mfrow = c(1, 1))
# 繪製「變數數量 vs 平均誤差」圖，用來判斷選幾個變數誤差最小。
plot(mean.cv.errors, type = "b")
```



```
# 使用「完整資料集」重新建立最佳子集模型
reg.best <- regsubsets(Salary ~ ., data = Hitters,
nvmax = 19)
# 查看最佳模型中（查看 10 個變數）的係數估計值
coef(reg.best, 10)
```

```
 (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns
162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798    1.4082490
       CRBI       CWalks    DivisionW      PutOuts      Assists
   0.7743122   -0.8308264 -112.3800575    0.2973726    0.2831680
```

## 6.5.2 Ridge Regression and the Lasso

我們將使用 glmnet 套件來進行嶺回歸和 Lasso 回歸。此套件的主要函數是 glmnet()，它可以用來擬合嶺迴歸模型、Lasso 模型等。此函數的語法，我們必須傳入一個 x 矩陣和一個 y 向量，並且不使用 y □ x 語法。
我們將對 Hitters 數據集進行嶺回歸和 lasso 回歸，以預測球員的薪資，在繼續之前，請確保已從資料中移除缺失值。

```
# 傳入一個 x 矩陣
x <- model.matrix(Salary ~ ., Hitters)[, -1]
# 傳入一個 y 向量
y <- Hitters$Salary
```

`model.matrix()` 函數對於建立 x 尤其有用；它不僅產生一個對應於 19 個預測變數的矩陣，而且還會自動將所有 qualitative variables 轉換為虛擬變數 dummy variables。
後者特性非常重要，因為 `glmnet()` 函數只能接受數值型 numerical、定量輸入 quantitative inputs。

### Ridge Regression

glmnet() 函數有一個 alpha 參數，用於確定擬合的模型類型。如果 alpha=0，則擬合嶺迴歸模型；如果 alpha=1，則擬合 Lasso 模型。我們首先擬合嶺迴歸模型。
預設情況下，glmnet() 函數會對自動選擇的 λ 值範圍執行嶺迴歸。然而，這裡我們選擇在 λ = 10¹□ 到 λ = 10□² 的網格值範圍內實現該函數，這基本上涵蓋了從僅包含截距的零模型到最小二乘擬合的所有情況。
每個 λ 值都關聯一個嶺回歸係數向量，該向量儲存在一個矩陣中，可透過 coef() 函數存取。在本例中，這是一個 20×100 的矩陣，包含 20 行（每行對應一個預測變量，外加一個截距）和 100 列（每列對應一個 λ 值）。

```
# 載入 glmnet 套件（專門用於 Ridge 和 Lasso）
library(glmnet)
# 設定 lambda（懲罰強度）的範圍
# 產生 100 個值，從 10^10（很大）到 10^-2（很小）
# Lambda 越大，對係數的懲罰越重（係數被壓縮得越小）
grid <- 10^seq(10, -2, length = 100)
# 建立 Ridge 回歸模型
# alpha = 0 代表 Ridge（L2 正則化）；（若 alpha = 1 則為 Lasso）
# 一次算出這 100 種 lambda 對應的所有係數結果
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
# 看看係數矩陣的維度
# 結果應為 [變數個數 p+1] x [lambda 個數 100]
dim(coef(ridge.mod))
```

```
[1]  20 100
```

```
# 顯示第 50 個 lambda 的數值
ridge.mod$lambda[50]
```

```
[1] 11497.57
```

```
# 顯示第 50 個 lambda 對應的「所有變數係數」
coef(ridge.mod)[, 50]
```

```
  (Intercept)         AtBat          Hits         HmRun          Runs
407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
          RBI         Walks         Years        CAtBat         CHits
  0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
       CHmRun         CRuns          CRBI        CWalks       LeagueN
  0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
    DivisionW       PutOuts       Assists        Errors    NewLeagueN
 -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```
# 計算係數的 L2 Norm（歐幾里得長度）
# 用來量化係數整體的大小（不包含截距項 [-1]）
sqrt(sum(coef(ridge.mod)[-1, 50]^2))
```

```
[1] 6.360612
```

```
# 第 60 個 lambda（比第 50 個小）
ridge.mod$lambda[60]
```

```
[1] 705.4802
```

```
# 對應的係數（理論上會比第 50 個時大一點）
coef(ridge.mod)[, 60]
```

```
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI
54.32519950   0.11211115   0.65622409   1.17980910   0.93769713   0.84718546
       Walks        Years       CAtBat        CHits       CHmRun        CRuns
 1.31987948   2.59640425   0.01083413   0.04674557   0.33777318   0.09355528
       CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
 0.09780402   0.07189612  13.68370191 -54.65877750   0.11852289   0.01606037
      Errors   NewLeagueN
-0.70358655   8.61181213
```

```
# 計算 L2 Norm（應比第 50 個時大）
sqrt(sum(coef(ridge.mod)[-1, 60]^2))
```

```
[1] 57.11001
```

```
# 預測/提取特定 lambda 值的係數
# s = 50 是指「當 lambda = 50 時」(並非第 50 個索引)
# 取得前 20 個變數的係數估計值
predict(ridge.mod, s = 50, type = "coefficients")[1:20, ]
```

```
  (Intercept)         AtBat          Hits         HmRun          Runs
 4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
          RBI         Walks         Years        CAtBat         CHits
 8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
       CHmRun         CRuns          CRBI        CWalks       LeagueN
 6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
    DivisionW       PutOuts       Assists        Errors    NewLeagueN
-1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

現在我們將樣本分為訓練集和測試集，以便估計嶺迴歸和 Lasso 迴歸的測試誤差，隨機劃分資料集有兩種常用方法。
第一種方法是產生一個由 TRUE 和 FALSE 元素組成的隨機向量，並選擇對應於 TRUE 的觀測值作為訓練資料。
第二種方法是隨機選擇一個介於 1 和 n 之間的數字子集；這些數字可以用作訓練觀測值的索引。

```
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# 隨機抽取一半的索引作為訓練集
train <- sample(1:nrow(x), nrow(x) / 2)
# 剩下的另一半作為測試集
```

```
test <- (-train)
# 先把測試集的正確答案 (y) 存起來備用
y.test <- y[test]
# --- 建立 Ridge 模型 (在訓練集上) ---
# alpha = 0 代表 Ridge
# thresh 是收斂門檻，設小一點讓結果精確些
ridge.mod <- glmnet(x[train, ], y[train], alpha = 0,
lambda = grid, thresh = 1e-12)
# --- 情境 A: 假設 Lambda = 4 ---
# 用訓練好的模型預測測試集，指定 s (lambda) = 4
ridge.pred <- predict(ridge.mod, s = 4, newx = x[test, ])
# 計算 MSE (均方誤差)：越小越好
mean((ridge.pred - y.test)^2)
```

```
[1] 142199.2
```

```
# --- 比較基準 (Baseline): 只用平均值猜測 ---
# 如果完全不看 x，只用「訓練集的平均值」來猜，誤差是多少？
# 這通常是用來檢查模型是否有效的最低標準
mean((mean(y[train]) - y.test)^2)
```

```
[1] 224669.9
```

```
# --- 情境 B: 假設 Lambda 超級大 (s = 1e10) ---
# Lambda 很大 -> 懲罰超重 -> 所有係數趨近於 0 -> 模型退化成只剩截距
# 預測結果會跟上面的「只用平均值猜」幾乎一樣
ridge.pred <- predict(ridge.mod, s = 1e10, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 224669.8
```

```
# --- 情境 C: 假設 Lambda = 0 ---
# exact = T 表示如果 grid 裡剛好沒有 0，請幫我重新精確計算一次
# 驗證 Ridge Regression 如果不給懲罰 (Lambda=0)，結果應該跟傳統 lm 一樣。
ridge.pred <- predict(ridge.mod, s = 0, newx = x[test, ],
exact = T, x = x[train, ], y = y[train])
mean((ridge.pred - y.test)^2)
```

```
[1] 168588.6
```

```
# --- 驗證：比較傳統 lm() ---
# 跑一般的線性迴歸
```

```
lm(y ~ x, subset = train)
```

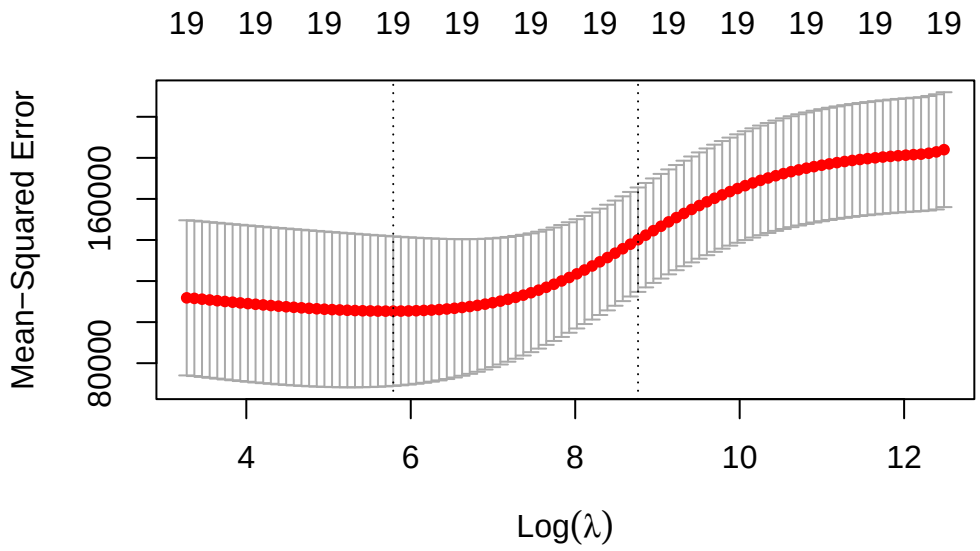
```
Call:
lm(formula = y ~ x, subset = train)

Coefficients:
(Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI
   274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
     xWalks       xYears      xCAtBat       xCHits      xCHmRun       xCRuns
     3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
      xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists
     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
    xErrors  xNewLeagueN
     -4.7128     -71.0951
```

```
# 顯示 Ridge (s=0) 的係數，會發現跟上面的 lm() 係數幾乎一樣。
predict(ridge.mod, s = 0, exact = T, type = "coefficients",
x = x[train, ], y = y[train])[1:20, ]
```

```
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI
274.0200994   -0.3521900   -1.6371383    5.8146692    1.5423361    1.1241837
       Walks        Years       CAtBat        CHits       CHmRun        CRuns
   3.7288406  -16.3795195   -0.6411235    3.1629444    3.4005281   -0.9739405
        CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
  -0.6003976    0.3378422 119.1434637 -144.0853061    0.1976300    0.6804200
      Errors   NewLeagueN
  -4.7127879  -71.0898914
```

```
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# --- 執行交叉驗證 (CV) ---
# cv.glmnet 會自動做 10-fold CV，試出哪個 lambda 誤差最小。
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
# 畫圖：X 軸是 Log Lambda，Y 軸是 MSE，最低點就是最佳解。
plot(cv.out)
```




```
# 找出最佳 Lambda
bestlam <- cv.out$lambda.min
# 顯示最佳的 lambda 值
bestlam
```

```
[1] 326.0828
```

```
# --- 用最佳 Lambda 再次預測測試集 ---
# 這是這個模型最終的效能分數
ridge.pred <- predict(ridge.mod, s = bestlam,
newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 139856.6
```

```
# --- 用完整資料 (x, y) 重新跑一次模型 ---
out <- glmnet(x, y, alpha = 0)
# 顯示最終模型的係數 (使用最佳 lambda)
predict(out, type = "coefficients", s = bestlam)[1:20, ]
```

```
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI
15.44383120   0.07715547   0.85911582   0.60103106   1.06369007   0.87936105
       Walks        Years       CAtBat        CHits       CHmRun        CRuns
 1.62444617   1.35254778   0.01134999   0.05746654   0.40680157   0.11456224
       CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
```

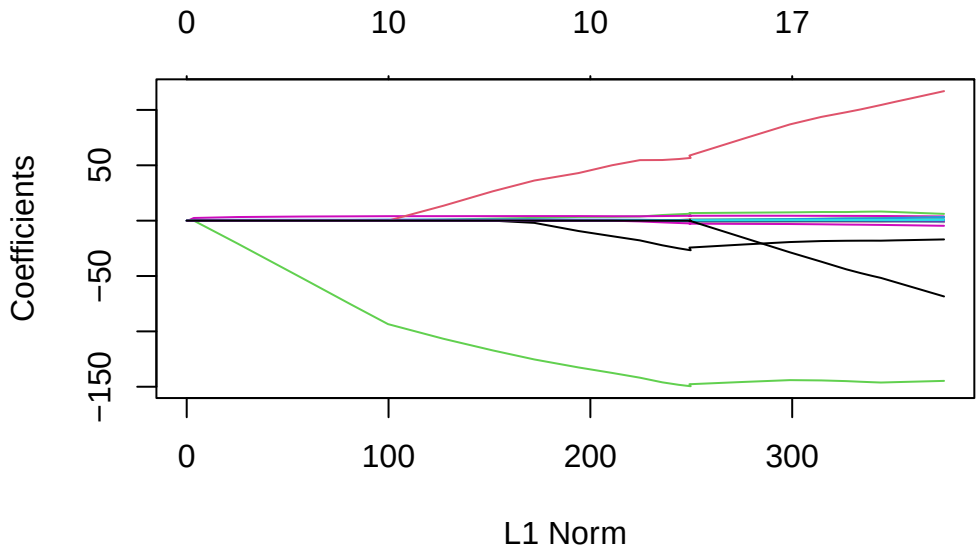
```
  0.12116504    0.05299202  22.09143197 -79.04032656    0.16619903    0.02941950
      Errors    NewLeagueN
-1.36092945    9.12487765
```

正如預期的那樣，所有係數都不為零——嶺回歸不會進行變數選擇！其中:
1. s = 0：就是一般線性迴歸 (Least Squares)。
2. s = 很大：係數全部被壓扁成 0，只剩下截距 (Intercept)。
3. cv.glmnet：是用來自動幫你在 0 到很大之間，找到一個平衡點 (Bias-Variance Trade-off)，讓預測誤差最小。

### The Lasso

我們看到在 Hitters 資料集上，合理選擇 λ 值的嶺迴歸模型可以優於最小平方法以及零模型。現在讓我們來探討 lasso 迴歸模型是否比嶺迴歸模型更準確或更容易解釋，為了擬合 lasso 模型，我們再次使用 glmnet() 函數；但是這次我們使用參數 alpha=1。

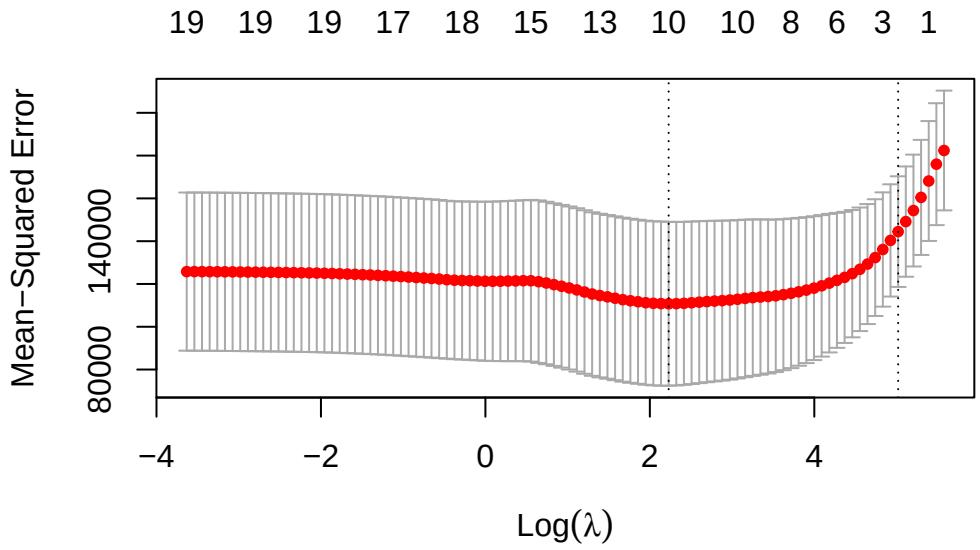
```
# 建立 Lasso 模型，alpha = 1 代表 Lasso 回歸。
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1,
lambda = grid)
# 畫出係數變化圖，每條線代表一個變數 (Feature) 的係數走勢。
plot(lasso.mod)
```




這張圖通常長得像一堆線從右邊發散出來，然後往左邊收斂。X 軸 (L1 Norm) 上:
1. 右邊代表 Lambda 很小 (懲罰輕)，接近一般的線性回歸，所有變數都有係數。2. 左邊代表 Lambda

很大 (懲罰重)，係數被壓縮。
Y 軸 (Coefficients) 是變數的係數值，你會看到線條一條接一條地歸零 (變成平平的貼在 0 軸上)，這就是 Lasso 最強大的功能 —— 變數挑選 (Variable Selection)。
從係數圖中可以看出，根據調參的選擇，某些係數將恰好等於零，現在我們進行交叉驗證並計算對應的測試誤差。

```
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# 執行交叉驗證 (Cross-Validation)，alpha = 1 代表 Lasso。
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
# 畫出 MSE 誤差圖 (U 型曲線)，用來視覺化尋找最低點。
plot(cv.out)
```




```
# 取出讓誤差最小/最佳的那個 Lambda 值
bestlam <- cv.out$lambda.min
# 使用之前訓練好的 lasso.mod 來預測測試集
lasso.pred <- predict(lasso.mod, s = bestlam,
newx = x[test, ])
# 計算均方誤差，評估模型準不準。
mean((lasso.pred - y.test)^2)
```

```
[1] 143673.6
```

```
# 建立最終模型 (使用完整資料 x, y)
# 既然確定了 bestlam，就用「所有資料」重新訓練一次，以獲得最穩健的係數。
```

```
out <- glmnet(x, y, alpha = 1, lambda = grid)
# 查看前 20 個變數的係數
lasso.coef <- predict(out, type = "coefficients",
s = bestlam)[1:20, ]
# 顯示係數結果
# 你會看到很多變數的係數變成了 "0.00"
lasso.coef
```

```
  (Intercept)         AtBat          Hits         HmRun          Runs
   1.27479059   -0.05497143    2.18034583    0.00000000    0.00000000
          RBI         Walks         Years        CAtBat         CHits
   0.00000000    2.29192406   -0.33806109    0.00000000    0.00000000
       CHmRun         CRuns          CRBI        CWalks       LeagueN
   0.02825013    0.21628385    0.41712537    0.00000000   20.28615023
     DivisionW       PutOuts       Assists        Errors    NewLeagueN
-116.16755870    0.23752385    0.00000000   -0.85629148    0.00000000
```

lasso 迴歸相比嶺迴歸的一個顯著優勢在於，其得到的係數估計值是稀疏的。這裡我們看到 19 個係數估計值中有 8 個剛好為零，因此透過交叉驗證選擇的 λ 值的 lasso 模型僅包含 11 個變數。

### 6.5.3 PCR and PLS Regression

#### Principal Components Regression

可以使用 pls 函式庫中的 pcr() 函數執行主成分迴歸 (PCR)，我們將 PCR 應用於 Hitters 資料集，以預測 Salary。同樣我們須確保已從資料中移除缺失值。

```
# 載入 pls 套件 (包含 PCR 和 PLS 功能)
library(pls)
# --- 第一階段：先用完整資料跑一次，觀察大概需要幾個成分 ---
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(2)
# 建立 PCR 模型
# scale = TRUE：強迫將資料標準化 (因為 PCR 對尺度很敏感)
# validation = "CV"：自動執行交叉驗證來評估誤差
pcr.fit <- pcr(Salary ~ ., data = Hitters, scale = TRUE,
validation = "CV")
# 查看模型摘要 (會顯示每個成分解釋了多少變異量)
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
# 畫圖：X 軸是「主成分個數」，Y 軸是「預測誤差 (MSEP)」
# 用來看選幾個成分時誤差最小
validationplot(pcr.fit, val.type = "MSEP")
```


```
# --- 第二階段：正式評估 (Train/Test Split) ---
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# 只用「訓練集 (train)」來訓練模型
pcr.fit <- pcr(Salary ~ ., data = Hitters, subset = train,
scale = TRUE, validation = "CV")
# 再畫一次圖，決定最佳成分個數。
validationplot(pcr.fit, val.type = "MSEP")
```


```
# 用「測試集 (test)」來算分數
# ncomp = 5：指定使用 5 個主成分來預測 (PCR 的精隨，選太少太簡單，太多會變成一般回歸。)
pcr.pred <- predict(pcr.fit, x[test, ], ncomp = 5)
# 計算測試集的 MSE (均方誤差)
mean((pcr.pred - y.test)^2)
```

```
[1] 142811.8
```

```
# --- 第三階段：建立最終模型 ---
# 既然確定 5 個成分最好，就用「所有資料 (x, y)」重新訓練一個最終版本。
pcr.fit <- pcr(y ~ x, scale = TRUE, ncomp = 5)
# 查看最終模型細節
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
```

### Partial Least Squares

PLS (Partial Least Squares，偏最小平方迴歸) 是 PCR 的進化版，PLS 通常可以用更少的主成分 (ncomp) 達到跟 PCR 一樣好、甚至更好的預測效果。
PCR (主成分迴歸)：是「非監督式」的。它只顧著把 X (特徵) 濃縮，完全不管 Y (薪水)。它假設變異量大的特徵就重要，但有時候雜訊的變異量也很大，這會導致 PCR 挑錯重點。
PLS (偏最小平方迴歸)：是「監督式」的。它在濃縮 X 的時候，會偷看 Y。它尋找的新變數（方向），不僅變異量大，還要跟 Y (薪水) 有高相關性。
我們使用 pls 函式庫中的 plsr() 函數來實作偏最小平方法 (PLS)，其語法與 pcr() 函數完全相同。

```
# 設定隨機種子，確保每次隨機抽樣結果一致。
set.seed(1)
# 訓練 PLS 模型，PLS 找主成分會同時考慮 X 和 Y 的關係。
pls.fit <- plsr(Salary ~ ., data = Hitters, subset = train, scale
= TRUE, validation = "CV")
# 查看摘要：注意看 % variance explained（解釋變異量）
# 你通常會發現 PLS 用比較少的成分就能解釋很多的 Y
summary(pls.fit)
```

```
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
```

```
# 畫圖選擇最佳成分個數，看看誤差 (MSEP) 在第幾個成分時最低。
validationplot(pls.fit, val.type = "MSEP")
```


當僅使用 M = 1 個偏最小平方法方向時，交叉驗證誤差最小，現在我們評估對應的測試集均方誤差 (MSE)。

```
# 對測試集進行預測，選擇 ncomp = 1 (只用 1 個主成分)
# 因為 PLS 效率很高，常常 1~2 個成分就夠準了 (PCR 剛才用了 5 個)
pls.pred <- predict(pls.fit, x[test, ], ncomp = 1)
# 計算預測誤差 (MSE)
mean((pls.pred - y.test)^2)
```

```
[1] 151995.3
```

測試均方誤差 (MSE) 與嶺回歸、lasso 回歸和 PCR 得到的測試 MSE 相當，但略高。
最後我們使用完整資料集進行偏最小二乘法 (PLS) 分析，其中 M = 1，即交叉驗證識別出的成分數。

```
# 建立最終模型，確定 ncomp = 1 效果不錯後，用「完整資料」重新訓練一次。
pls.fit <- plsr(Salary ~ ., data = Hitters, scale = TRUE,
```

```
ncomp = 1)
# 查看最終模型細節
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
```

單一成分 PLS 擬合解釋的薪資變異數百分比（43.05%）幾乎與最終五成分模型 PCR 擬合解釋的變異數百分比（44.90%）相同。這是因為 PCR 僅試圖最大化預測變數的變異數解釋量，而 PLS 則尋找能夠同時解釋預測變數和反應變數變異數的方向!!