HW3

108071601 計財所碩二 賴冠維

2020/11/1

(2)

(a)

先對X_i進行標準化,產生Z_i,
並且對Y_i進行去中心化，產生$\widetilde{Y_i}$

```
## Warning: package 'glmnet' was built under R version 4.0.3

## Loading required package: Matrix

## Loaded glmnet 4.0-2
```

首先固定λ為 0.01，再利用 glmnet 函式估計 Ridge Regression 的 OLS 解，

分為(x_i, y_i)、$(z_i, \widetilde{Y_i})$兩組進行，
求得參數 **par1** 及 **par2**，分別代表$\hat{\beta}$以及$\tilde{\beta}$

```
## [1] -0.5588065  4.2011865 -1.2774805 -0.3433947  1.1131355 -3.7603531

## [1] -0.00000000000000005961494  3.74019025532804860745273
## [3] -1.25768969424733034756514 -0.32447521474505081062745
## [5]  1.18504269468205314375098 -3.98623379718136572336107
```

因為z_i不為方陣，因此在求反矩陣時使用$Moore - Penrose 偽逆矩陣$。
藉此我們便可從$\tilde{\beta}$推得$\hat{\beta}$,矩陣表示式如下：

$$X\hat{\beta} = Z\tilde{\beta}$$

$$\hat{\beta} = X^{-1}Z\tilde{\beta}$$

- 表一為從$\tilde{\beta}$推導$\hat{\beta}$的數值
- 表二為上題從 glmnet()函式所得到的參數。

```
##            [,1]
## [1,]  4.1991588
## [2,] -1.2218201
## [3,] -0.3430106
## [4,]  1.0933922
## [5,] -3.7700609

##      [,1]
## [1,] "Beta_hat 1: 4.20118651254475"
## [2,] "Beta_hat 2: -1.27748052895946"
## [3,] "Beta_hat 3: -0.34339467379893"
```

```
## [4,] "Beta_hat 4: 1.11313549138748"
## [5,] "Beta_hat 5: -3.76035306912517"
```

此外也可從$\hat{\beta}$推得$\tilde{\beta}$的數值，
承列如下：

```
##             [,1]
## [1,]  3.7401903
## [2,] -1.2576897
## [3,] -0.3244752
## [4,]  1.1850427
## [5,] -3.9862338
```

```
##       [,1]
## [1,] "Beta_Standardize 1: 3.74019025532805"
## [2,] "Beta_Standardize 2: -1.25768969424733"
## [3,] "Beta_Standardize 3: -0.324475214745051"
## [4,] "Beta_Standardize 4: 1.18504269468205"
## [5,] "Beta_Standardize 5: -3.98623379718137"
```

(b)

產生λ_i: 2^{-10}、2^{-9}... 、2^4 、2^5共 16 組

```
##                 [,1]
##  [1,] 32.0000000000
##  [2,] 16.0000000000
##  [3,]  8.0000000000
##  [4,]  4.0000000000
##  [5,]  2.0000000000
##  [6,]  1.0000000000
##  [7,]  0.5000000000
##  [8,]  0.2500000000
##  [9,]  0.1250000000
## [10,]  0.0625000000
## [11,]  0.0312500000
## [12,]  0.0156250000
## [13,]  0.0078125000
## [14,]  0.0039062500
## [15,]  0.0019531250
## [16,]  0.0009765625
```

由於此題定義的 Loss Function 如下：

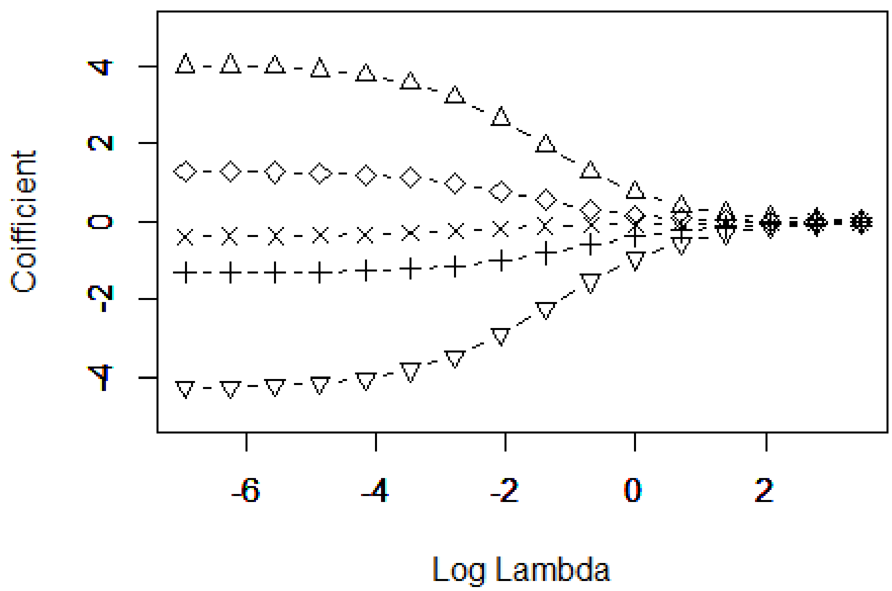
$$min \frac{1}{2N} \Sigma_{i=1}^{N} (y_i - \Sigma_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \Sigma_{j=1}^{j} \beta_j^2$$

故一階導數為 0 之最小值，其矩陣表示式為：

$$\hat{\beta}_{\lambda}^{ridge} = (\frac{1}{N} X^T X + 2\lambda I_p)^{-1} \frac{1}{N} X^T Y$$

且上式 X 需要標準化，Y 需去中心化，可得結果如下

1. 依不同的λ_i所求之$\hat{\beta}(\lambda)$畫出 Solution Path Line Plot



2. 列出從$(2^{-10}\ldots2^{5})$之下的$\hat{\beta}(\lambda)$

```
##           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  4.020300  4.0041486  3.9722278  3.9098700  3.7907745  3.5728162
## [2,] -1.312362 -1.3093520 -1.3033565 -1.2914645 -1.2680992 -1.2231561
## [3,] -0.376736 -0.3735725 -0.3673717 -0.3554547 -0.3334134 -0.2954987
## [4,]  1.290542  1.2844804  1.2725069  1.2491435  1.2046274  1.1235493
## [5,] -4.261262 -4.2454611 -4.2142277 -4.1531895 -4.0365239 -3.8226664
##            [,7]       [,8]       [,9]       [,10]       [,11]       [,12]
## [1,]  3.2033346  2.6513172  1.9648283  1.28366151  0.74709780  0.39976831
## [2,] -1.1407041 -1.0033481 -0.8092079 -0.58874990 -0.38835442 -0.23601979
## [3,] -0.2382756 -0.1688230 -0.1077309 -0.07128855 -0.05369614 -0.04079682
## [4,]  0.9874544  0.7880784  0.5489850  0.32561934  0.16494763  0.07368899
## [5,] -3.4589043 -2.9116415 -2.2218062 -1.52005123 -0.94257961 -0.54214434
##            [,13]       [,14]        [,15]        [,16]
## [1,]  0.20408642  0.10222344  0.050961069  0.025409120
## [2,] -0.13426479 -0.07262863 -0.037967456 -0.019442321
## [3,] -0.02785124 -0.01690186 -0.009424342 -0.004993725
## [4,]  0.03071533  0.01273554  0.005489593  0.002488339
## [5,] -0.29568199 -0.15556787 -0.080010647 -0.040609009
```

$\hat{\beta}(2)$ 在 Origin Scale 下的估計值

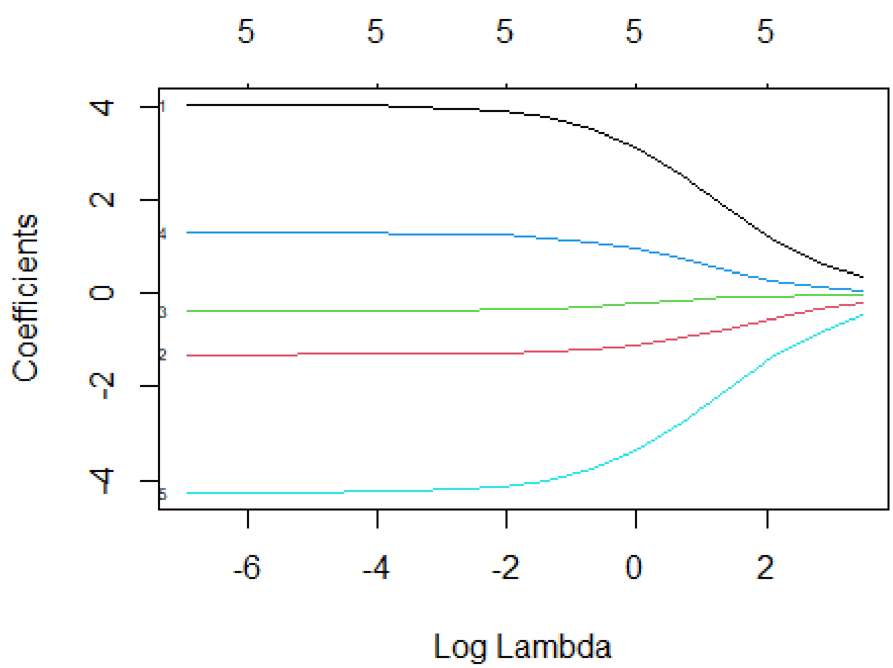
```
##            [,1]
## [1,]  4.3202397
## [2,] -1.4257163
## [3,] -0.3573408
## [4,]  1.2204615
## [5,] -3.8767087
```

(c)

使用 glmnet()函式所求給定在λ_i之下的$\hat{\beta}(\lambda)$的估計值

```
##           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
## V1  4.0354887  4.0342465  4.0321537  4.0277084  4.0187853  3.9998389  3.9629047
## V2 -1.3154439 -1.3154067 -1.3149106 -1.3138770 -1.3125249 -1.3081668 -1.3009412
## V3 -0.3797479 -0.3794963 -0.3791142 -0.3782932 -0.3766445 -0.3728343 -0.3655736
## V4  1.2960005  1.2954515  1.2945884  1.2927764  1.2889759  1.2822591  1.2685813
## V5 -4.2757784 -4.2744406 -4.2722708 -4.2678197 -4.2583968 -4.2407819 -4.2050531
##          [,8]       [,9]     [,10]      [,11]      [,12]       [,13]       [,14]
## V1  3.891004  3.7550071  3.509723  3.1032234  2.5158766  1.81736870  1.15729912
## V2 -1.287130 -1.2605472 -1.209400 -1.1171954 -0.9673876 -0.76427394 -0.54427435
## V3 -0.351754 -0.3267444 -0.284791 -0.2240416 -0.1544224 -0.09791823 -0.06660194
## V4  1.242014  1.1915913  1.100699  0.9514429  0.7405346  0.49950974  0.28633785
## V5 -4.135156 -4.0021456 -3.761532 -3.3606441 -2.7770164 -2.07197881 -1.38684773
##           [,15]       [,16]
## V1  0.65994053  0.34870677
## V2 -0.35245312 -0.21100990
## V3 -0.05096718 -0.03808681
## V4  0.14080939  0.06167905
## V5 -0.84498895 -0.47989915
```

將 glmnet()所求之$\hat{\beta}(\lambda)$畫出 Solution Path Line Plot



- 從(b),(c)小題兩種估計的 Coefficients 可發現:

1. 兩者的所估計的值於2^{-10}的時候近乎相同，但隨i變動，兩數之間開始出現差異，可見到2^5時，兩數已明顯不相同。
2. 由 Solution Line Plot 更可明顯看出 自行推導的矩陣解並未收斂。
   由上述兩點推測，差異的來源可來自於:
   glmnet()函式所用的 Loss Function 與課程上所推導的有差別。

使用 vignette()查閱 glmnet 套件的說明後發現，

glmnet 的默認的分配為 Gaussian，以下為 glmnet 所用 Ridge Regression 的 Loss Function：

$$min\frac{1}{2N}\Sigma_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)^2+\lambda[||\beta||_2^2/2]$$

與此題所使用的 Loss Function 不相同，

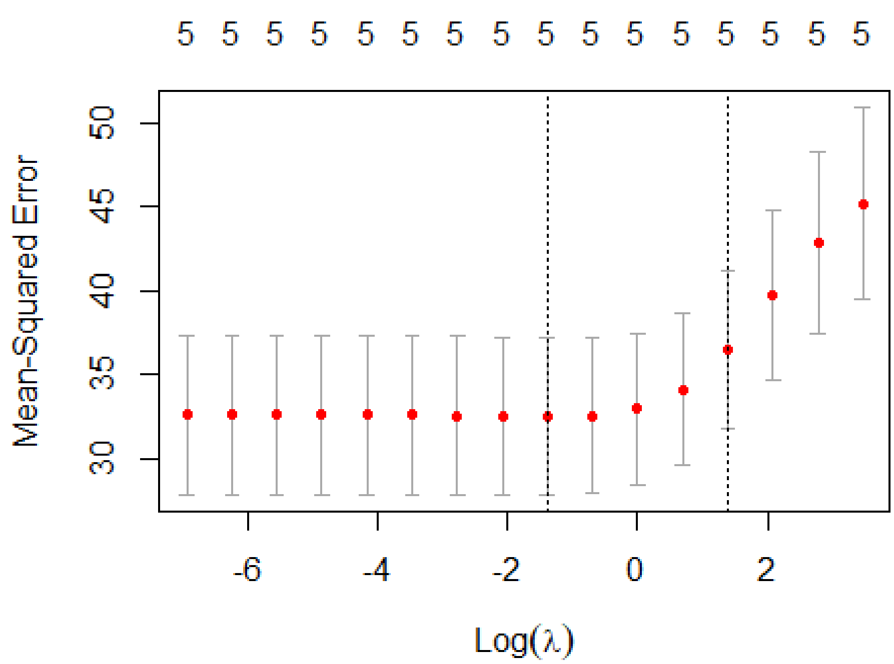
$$min\frac{1}{2N}\Sigma_{i=1}^{N}(y_i-\Sigma_{j=1}^{p}x_{ij}\beta_j)^2+\lambda\Sigma_{j=1}^{j}\beta_j^2$$

因此兩者差異可能來自：
後面λ項差了一個除以 2 的部分，而導致我們自己求的矩陣解的λ_i效果為套件的 2 倍， 因此隨著λ_i的值增加兩者的差異擴大，也導致自行求的矩陣解收斂速度快於 glmnet()

(d)

使用 Cross Validation 的方式計算模型在不同λ_i之下的表現，表現如下：
可以發現不管是 min 或 lse 之下，都是 5 個參數，並未達到變數篩選的功能， 其原因可能來自我們使用模擬的隨機項，變數間並無意義上的差別。

```
##
## Call:  cv.glmnet(x = z, y = y_1, lambda = num, nfolds = 10, family = "gaussian",
alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min   0.25   32.50 4.682       5
## 1se   4.00   36.49 4.746       5
```



我們將資料以$(75\%, 25\%)$分為 Train、Test Set
接著以λ_{min}及λ_{lse} 配飾兩個$Ridge Regression$
以 Train Set 配飾模型，比較其預測 Test Set 的 SSE，借此衡量兩模型的表現。

兩個模型估計出不同的$\widehat{\beta_i}$

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##             s0
## V1  3.5223511
## V2 -2.3957434
## V3 -0.2242332
## V4  2.1627620
## V5 -4.4781497
```

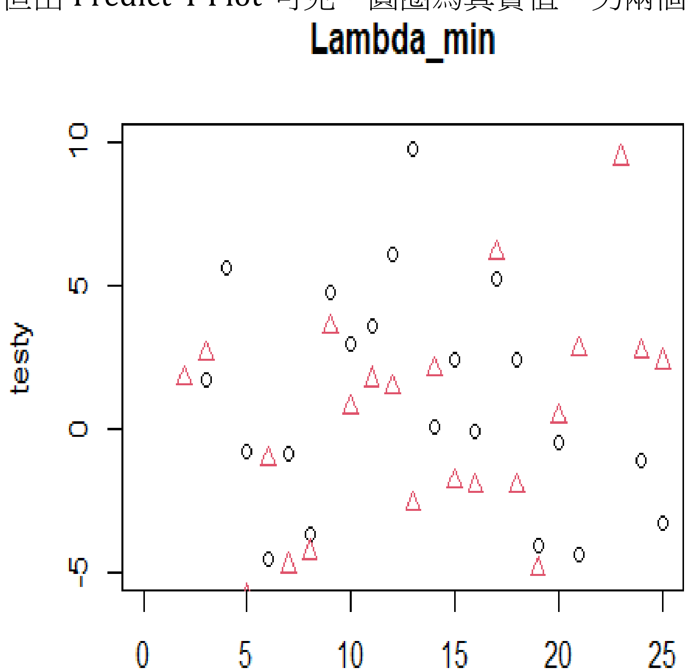
```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##             s0
## V1  1.9071403
## V2 -1.5158969
## V3 -0.2215735
## V4  0.9047308
## V5 -2.3901481
```

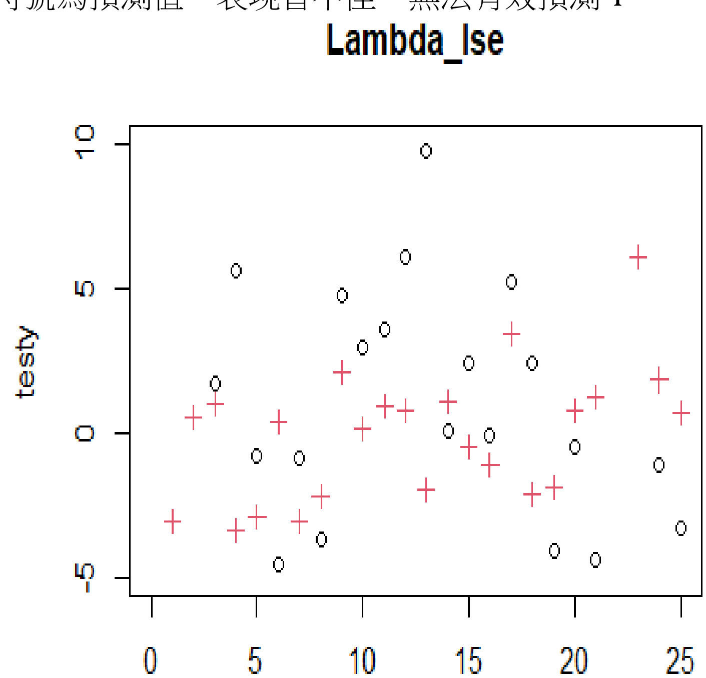
由此可見使用λ_{lse}的模型的 SSE 較小，表現較佳

```
## [1] "SSE of lambda.min: 1034.73466015926"
```

```
## [1] "SSE of lambda.lse: 851.897302170231"
```

但由 Predict-Y Plot 可見，圓圈為真實值，另兩個符號為預測值，表現皆不佳，無法有效預測 Y。





附錄(程式碼)：

```
library(magrittr)
options(scipen = 999)
## (1)
set.seed(36)
n = 100; sigma = 5; beta0 = c(2,-2,0.5,1,-3)
cormat = diag(1,nrow=5,ncol=5) ; cormat[cormat==0] = 0.5
cholmat = chol(cormat) #Choleskey 分解
x= matrix(rnorm(5*n,0,1), ncol = 5) %*% cholmat
err = rnorm(n,0,sigma)
y = x %*% beta0 + err

## (2)
#### (a)
library(glmnet)
x_center = sapply(1:5,function(a)
{
  mean(x[,a])
}
)
x_sd = sapply(1:5, function(a){
  sd(x[,a])
}
)
z = sapply(1:5, function(a){
  (x[,a] - x_center[a])/x_sd[a]
})
y_1 = y - mean(y)
fit_ridge = glmnet(x = x,y = y,alpha = 0)
(par1 = fit_ridge %>% coef(s=0.01) %>% as.numeric())
fit1_ridge = glmnet(x = z,y = y_1,alpha = 0)
```

```
(par2 = fit1_ridge %>% coef(s=0.01) %>% as.numeric())
library(MASS)
ginv(x) %*% z %*% par2[2:6]
paste0("Beta_hat ",seq(1:5),": ",par1[2:6]) %>% as.matrix()
ginv(z) %*% x %*% par1[2:6]
paste0("Beta_Standardize ",seq(1:5),": ",par2[2:6]) %>% as.matrix()


#### (b)
(num = 2^c(-10:5)%>% sort(decreasing = T)) %>% as.matrix()
par_own = sapply(1:16, function(a){
  q = solve((1/length(y_1))*t(z)%*%z + 2*num[a]*diag(1,nrow=5,ncol=5))
  p = (1/length(y_1))*t(z) %*% y_1
  ans = q %*% p
})
n = log(num)
plot(par_own[1,],x=n,type = "b",ylim = c(-5,5),pch=2,xlab = "Log Lambda",
     ylab= "Coifficient")
for (i in 2:5) {
  par(new=T)
  plot(par_own[i,],x=n,type = "b",ylim = c(-5,5),pch=i+1,ylab= "",xlab = "")
}
par_own = sapply(1:16, function(a){
  par_own[,a] = par_own[,17-a]
})
par_own %>% as.matrix()
a = solve(t(x)%*%x + 2*diag(1,nrow=5,ncol=5))
a %*% t(x) %*% y %>% as.matrix()
num = 2^c(-10:5)%>% sort(decreasing = T)
fit2_ridge = glmnet(x = z,
                  y = y_1,
                  alpha = 0,
```

```
                lambda = num)
par_glment =  fit2_ridge$beta %>% as.matrix()
par_glment = sapply(1:16, function(a){
  par_glment[,a] = par_glment[,17-a]
})
par_glment
plot(fit2_ridge, xvar = "lambda",label = T)


#### (d)
set.seed(10)
CVRidge = cv.glmnet(x = z,y = y_1,family = "gaussian",lambda = num,nfold = 10,alpha = 0)
CVRidge
plot(CVRidge)
set.seed(123)
index = sample(1:100,size = 75,replace = F)
trainx = z[index,]
trainy = y_1[index,]
testx = z[-index,]
testy = y_1[-index,]
ridge1 = glmnet(x = trainx,y = trainy,family = "gaussian",alpha = 0,lambda = CVRidge$lambda.min)
ridge2 = glmnet(x = trainx,y = trainy,family = "gaussian",alpha = 0,lambda = CVRidge$lambda.1se )
print(ridge1$beta)
print(ridge2$beta)
pred_ridge1 = predict(ridge1,testx)
pred_ridge2 = predict(ridge2,testx)
paste0("SSE of lambda.min: ",sum((pred_ridge1-testy)^2))
paste0("SSE of lambda.lse: ",sum((pred_ridge2-testy)^2))
plot(testy,xlim = c(0,25),ylim = c(-5,10),main = "Lambda_min")
points(pred_ridge1,col=2,pch=2)
plot(testy,xlim = c(0,25),ylim = c(-5,10),main = "Lambda_lse")
points(pred_ridge2,col=2,pch=3)
```