STAT 542: Homework 2

Shashi Roshan (sroshan2)

Contents

Question 1 [30 Points] Linear Model Selection

We will use the Boston Housing data (`BostonHousing2` from the `mlbench` package) for this question. If you do not use R, you can download a `.csv` file from the course website. We will remove variables `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. First, you need to standardize all variables marginally (only the covariates, not the outcome) to mean 0 and sample variation 1. Answer the following questions by performing linear regression and model selection.

a. [5 Points] Perform a linear regression and obtain the ordinary least square estimators and their variances.

```r
library(mlbench)
library(leaps)

data(BostonHousing2)

df = as.data.frame(BostonHousing2)

# remove columns : medv, town and tract (as mentioned in the Ques)
df$medv = NULL
df$town = NULL
df$tract = NULL

predictors = setdiff(colnames(df), c('cmedv'))

numerical_predictors = predictors

df$chas = as.numeric(df$chas)

# scale the numerical predictors (mean= 0, sd = 1)
for (numerical_predictor in numerical_predictors) {
  df[, numerical_predictor] = scale(df[, numerical_predictor])
}
```

```r
linear_model = lm(cmedv ~ ., data = df)
summary(linear_model)
```

```
##
## Call:
## lm(formula = cmedv ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5831  -2.7643  -0.5994   1.7482  26.0822
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.52885    0.20895 107.820  < 2e-16 ***
## lon         -0.29674    0.25427  -1.167 0.243770
## lat          0.27772    0.22665   1.225 0.221055
## crim        -0.89927    0.28053  -3.206 0.001436 **
## zn           1.08600    0.32035   3.390 0.000755 ***
## indus        0.10459    0.42364   0.247 0.805106
## chas         0.65480    0.21972   2.980 0.003024 **
## nox         -1.83372    0.46412  -3.951 8.93e-05 ***
## rm           2.63742    0.29269   9.011  < 2e-16 ***
## age          0.06946    0.37582   0.185 0.853440
## dis         -2.94784    0.43974  -6.704 5.61e-11 ***
## rad          2.67064    0.57975   4.607 5.23e-06 ***
## tax         -2.17182    0.62807  -3.458 0.000592 ***
## ptratio     -1.89892    0.29504  -6.436 2.92e-10 ***
## b            0.83774    0.24310   3.446 0.000618 ***
## lstat       -3.83790    0.36001 -10.660  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.7 on 490 degrees of freedom
## Multiple R-squared:  0.7458, Adjusted R-squared:  0.738
## F-statistic: 95.82 on 15 and 490 DF,  p-value: < 2.2e-16
```

```
# variance of beta_hat
coef(summary(linear_model))[,2]^2
```

```
## (Intercept)         lon         lat        crim          zn       indus        chas
##  0.04365969  0.06465208  0.05137158  0.07869648  0.10262204  0.17947492  0.04827557
##         nox          rm         age         dis         rad         tax     ptratio
##  0.21540709  0.08566990  0.14123974  0.19337482  0.33611403  0.39447244  0.08704823
##           b       lstat
##  0.05909841  0.12961030
```

```
# or
# diag(vcov(linear_model))
```

b. [5 Points] Starting from the full model, use stepwise regression with backward and BIC criterion to select the best model. Which variables are removed from the full model?

```
# stepwise using BIC, direction backward
linear_model_2 = step(linear_model, direction="backward", k=log(nrow(df)), trace=0)
summary(linear_model_2)
```

```
##
## Call:
## lm(formula = cmedv ~ crim + zn + chas + nox + rm + dis + rad +
##     tax + ptratio + b + lstat, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.566  -2.686  -0.552   1.790  26.167
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  22.5289     0.2087 107.964  < 2e-16 ***
## crim         -0.9174     0.2794  -3.283 0.001099 **
## zn            1.0985     0.3126   3.514 0.000481 ***
## chas          0.6927     0.2150   3.221 0.001360 **
## nox          -2.0066     0.4060  -4.943 1.06e-06 ***
## rm            2.6550     0.2829   9.384  < 2e-16 ***
## dis          -3.2012     0.3876  -8.259 1.35e-15 ***
## rad           2.5822     0.5471   4.720 3.08e-06 ***
## tax          -2.0354     0.5633  -3.613 0.000333 ***
## ptratio      -1.9853     0.2769  -7.169 2.77e-12 ***
## b             0.8401     0.2419   3.473 0.000561 ***
## lstat        -3.7736     0.3356 -11.243  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.694 on 494 degrees of freedom
## Multiple R-squared:  0.7444, Adjusted R-squared:  0.7387
## F-statistic: 130.8 on 11 and 494 DF,  p-value: < 2.2e-16
```

```
# list of predictors removed from the full model after stepwise
setdiff(names(linear_model$coefficients), names(linear_model_2$coefficients))
```

```
## [1] "lon"   "lat"   "indus" "age"
```

c. [5 Points] Starting from this full model, use the best subset selection and list the best model of each model size.

```
RSSleaps = regsubsets(cmedv ~ ., data = df, nvmax = length(predictors))
summary(RSSleaps)$outmat
```

```
##           lon lat crim zn  indus chas nox rm  age dis rad tax ptratio b   lstat
## 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 ) "*" "*" "*"  "*" "*"   "*"  "*" "*" "*" "*" "*" "*" "*"     "*" "*"
```

d. [5 Points] Use the BIC criterion to select the best model from part c). Which variables are removed from the full model?

```r
# the model with the lowest BIC is preferred
bic_scores = summary(RSSleaps)$bic
bic_scores
```

```
##  [1] -390.2869 -501.7644 -553.5203 -567.0714 -592.0998 -599.6891 -605.1761 -607.5510
##  [9] -606.6796 -611.1827 -615.4752 -610.4086 -605.6812 -599.5209 -593.3297
```

```r
# model with 11 variables has lowest BIC score
order(bic_scores)[1]
```

```
## [1] 11
```

```r
# BIC score of best model
bic_scores[order(bic_scores)[1]]
```

```
## [1] -615.4752
```

Variables removed from the full model : lon, lat, indus, age

e. [10 Points] Our solution is obtained based on the scaled X. Can you recover the original OLS estimates based on the original data? For this question, you can use information from the original design matrix. However, you cannot refit the linear regression. Provide a rigorous mathematical derivation and also validate that by comparing it to the `lm()` function on the original data.

```r
df_unscaled = as.data.frame(BostonHousing2)

df_unscaled$medv = NULL
df_unscaled$town = NULL
df_unscaled$tract = NULL

df_unscaled$chas = as.numeric(df_unscaled$chas)

# intercept calculation
intercept = linear_model$coefficients['(Intercept)']

for (numerical_predictor in numerical_predictors){
  intercept = intercept - (linear_model$coefficients[numerical_predictor]* mean(df_unscaled[, numerical_
}

unscaled_estimates = c()

for (numerical_predictor in numerical_predictors){
  unscaled_estimate = linear_model$coefficients[numerical_predictor] / sd(df_unscaled[, numerical_predi
  unscaled_estimates = c(unscaled_estimates, unscaled_estimate)
}

# unscaled estimates retrieved from scaled estimates
c(intercept, unscaled_estimates)
```

```
##   (Intercept)           lon           lat          crim            zn         indus
## -4.375739e+02 -3.935202e+00  4.495441e+00 -1.045469e-01  4.656476e-02  1.524535e-02
##          chas           nox            rm           age           dis           rad
##  2.578020e+00 -1.582458e+01  3.753712e+00  2.467659e-03 -1.399927e+00  3.067145e-01
##           tax       ptratio             b         lstat
## -1.288633e-02 -8.771212e-01  9.176196e-03 -5.374411e-01
```

```
linear_model_unscaled_predictors = lm(cmedv ~ ., data = df_unscaled)
summary(linear_model_unscaled_predictors)
```

```
##
## Call:
## lm(formula = cmedv ~ ., data = df_unscaled)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5831  -2.7643  -0.5994   1.7482  26.0822
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.376e+02  3.031e+02  -1.444 0.149494
## lon         -3.935e+00  3.372e+00  -1.167 0.243770
## lat          4.495e+00  3.669e+00   1.225 0.221055
## crim        -1.045e-01  3.261e-02  -3.206 0.001436 **
## zn           4.657e-02  1.374e-02   3.390 0.000755 ***
## indus        1.524e-02  6.175e-02   0.247 0.805106
## chas         2.578e+00  8.650e-01   2.980 0.003024 **
## nox         -1.582e+01  4.005e+00  -3.951 8.93e-05 ***
## rm           3.754e+00  4.166e-01   9.011  < 2e-16 ***
## age          2.468e-03  1.335e-02   0.185 0.853440
## dis         -1.400e+00  2.088e-01  -6.704 5.61e-11 ***
## rad          3.067e-01  6.658e-02   4.607 5.23e-06 ***
## tax         -1.289e-02  3.727e-03  -3.458 0.000592 ***
## ptratio     -8.771e-01  1.363e-01  -6.436 2.92e-10 ***
## b            9.176e-03  2.663e-03   3.446 0.000618 ***
## lstat       -5.374e-01  5.042e-02 -10.660  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.7 on 490 degrees of freedom
## Multiple R-squared:  0.7458, Adjusted R-squared:  0.738
## F-statistic: 95.82 on 15 and 490 DF,  p-value: < 2.2e-16
```

Que. 1.e.

$$\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{p} \left(\frac{x_i - \bar{x}_i}{\sigma_i} \right) \hat{\beta}_i$$

Rearranging above equation :

$$\hat{y} = \underbrace{\left(\hat{\beta}_0 - \sum_{i=1}^{p} \hat{\beta}_i \frac{\bar{x}_i}{\sigma_i} \right)}_{\text{unscaled intercept}} + \sum_{i=1}^{p} \underbrace{\left(\frac{\hat{\beta}_i}{\sigma_i} \right)}_{\text{unscaled predictors}} x_i$$

Question 2 [70 Points] Ridge Regression and Scaling Issues

For this question, you can **ONLY** use the base package. We will use the dataset from Question 1 a). However, you should further standardize the outcome variable `cmedv` to mean 0 and sample variance 1. Hence, no intercept term is needed when you fit the model.

a. [30 points] First, fit a ridge regression with your own code, with the objective function $||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda||\beta||^2$.

- You should consider a grid of 100 penalty λ values. Your choice of lambda grid can be flexible. However, they should be appropriate for the scale of the objective function.
- Calculate the degrees of freedom and the leave-one-out cross-validation (computationally efficient version) based on your choice of the penalty grid.
- Report details of your model fitting result. In particular, you should produce a plot similar to page 25 in the lecture note `Penalized`

```r
library(mlbench)
library(leaps)

data(BostonHousing2)

df = as.data.frame(BostonHousing2)

# remove columns : medv, town and tract (as mentioned in the Ques)
df$medv = NULL
df$town = NULL
df$tract = NULL
```

```r
predictors = setdiff(colnames(df), c('cmedv'))

numerical_predictors = predictors

df$chas = as.numeric(df$chas)

# scale the numerical predictors (mean= 0, sd = 1)
for (numerical_predictor in numerical_predictors) {
  df[, numerical_predictor] = scale(df[, numerical_predictor])
}

# scale the outcome variable
df[, 'cmedv'] = scale(df[, 'cmedv'])

lambda_values = seq(1, 100, by = 1)
lambda_values
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
##  [22]  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42
##  [43]  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
##  [64]  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84
##  [85]  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```r
X = data.matrix(df[, predictors])
Y = data.matrix(df[, 'cmedv'])
Xt_X = t(X) %*% X
Xt_Y = t(X) %*% Y
I = diag(dim(Xt_X)[1])

svd_matrix = svd(X)
D = svd_matrix$d

matrix_of_ridge_estimates = matrix(0, nrow = length(lambda_values), ncol = 15)

lambda_idx = 1
dof_array = c()
cv_error_array = c()

for (lambda_value in 1:100){
  beta_hat_ridge = solve(Xt_X + lambda_value*I) %*% Xt_Y

  dof = 0
  for (idx in 1:length(D)){
    dof = dof + (D[idx]^2 / (D[idx]^2 + lambda_value))
  }
  dof_array = c(dof_array, dof)

  hat_matrix = X %*% solve(Xt_X + lambda_value*I) %*% t(X)

  cv_error = 0
  for(idx in 1:length(Y)){
    cv_error = cv_error + ((Y[idx,] - X[idx,] %*% beta_hat_ridge) / (1 - hat_matrix[idx, idx]) )^2
  }
  cv_error_array = c(cv_error_array, cv_error)

  matrix_of_ridge_estimates[lambda_idx, ] = beta_hat_ridge
  lambda_idx = lambda_idx + 1
}

# degrees of freedom
dof_array
```

```
##   [1] 14.90341 14.80968 14.71863 14.63009 14.54390 14.45993 14.37805 14.29814 14.22011
##  [10] 14.14384 14.06926 13.99629 13.92484 13.85485 13.78625 13.71898 13.65299 13.58823
##  [19] 13.52464 13.46217 13.40079 13.34046 13.28114 13.22278 13.16536 13.10885 13.05322
##  [28] 12.99843 12.94445 12.89128 12.83887 12.78721 12.73627 12.68604 12.63648 12.58759
##  [37] 12.53935 12.49173 12.44473 12.39832 12.35249 12.30722 12.26250 12.21832 12.17466
##  [46] 12.13152 12.08887 12.04671 12.00503 11.96381 11.92305 11.88273 11.84285 11.80339
##  [55] 11.76435 11.72572 11.68748 11.64964 11.61218 11.57510 11.53838 11.50203 11.46603
##  [64] 11.43038 11.39506 11.36009 11.32544 11.29111 11.25710 11.22340 11.19000 11.15691
##  [73] 11.12411 11.09160 11.05937 11.02742 10.99575 10.96435 10.93321 10.90234 10.87172
##  [82] 10.84136 10.81125 10.78138 10.75175 10.72236 10.69321 10.66429 10.63559 10.60712
##  [91] 10.57887 10.55083 10.52301 10.49540 10.46800 10.44080 10.41381 10.38702 10.36042
## [100] 10.33401
```

```
# cross validation
cv_error_array
```

```
##   [1] 139.4936 139.4533 139.4255 139.4088 139.4018 139.4032 139.4122 139.4279 139.4495
##  [10] 139.4764 139.5080 139.5438 139.5835 139.6266 139.6728 139.7218 139.7734 139.8273
##  [19] 139.8833 139.9412 140.0009 140.0622 140.1250 140.1892 140.2546 140.3212 140.3889
##  [28] 140.4576 140.5273 140.5978 140.6691 140.7411 140.8139 140.8874 140.9614 141.0361
##  [37] 141.1113 141.1870 141.2632 141.3399 141.4171 141.4946 141.5726 141.6509 141.7296
##  [46] 141.8087 141.8880 141.9678 142.0478 142.1281 142.2087 142.2895 142.3707 142.4520
##  [55] 142.5337 142.6155 142.6977 142.7800 142.8625 142.9453 143.0283 143.1114 143.1948
##  [64] 143.2784 143.3621 143.4461 143.5302 143.6145 143.6989 143.7836 143.8684 143.9533
##  [73] 144.0384 144.1237 144.2091 144.2947 144.3805 144.4663 144.5524 144.6385 144.7249
##  [82] 144.8113 144.8979 144.9846 145.0715 145.1585 145.2456 145.3328 145.4202 145.5077
##  [91] 145.5953 145.6830 145.7709 145.8589 145.9470 146.0352 146.1235 146.2119 146.3005
## [100] 146.3892
```

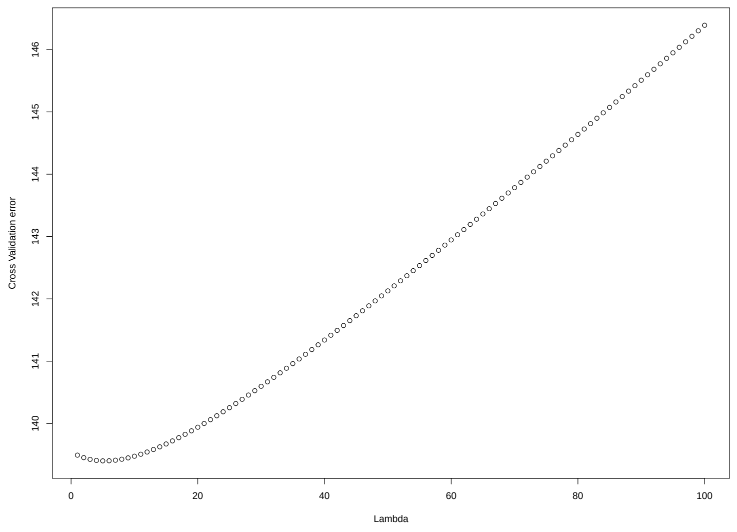
```
# minimum cv error
cv_error_array[order(cv_error_array)][1]
```

```
## [1] 139.4018
```

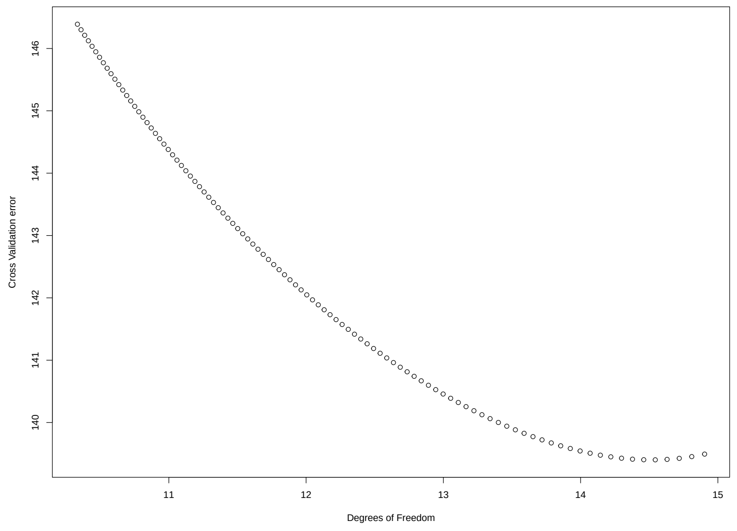
```
# optimal lambda corresponding to minimum cv error
lambda_values[order(cv_error_array)][1]
```

```
## [1] 5
```

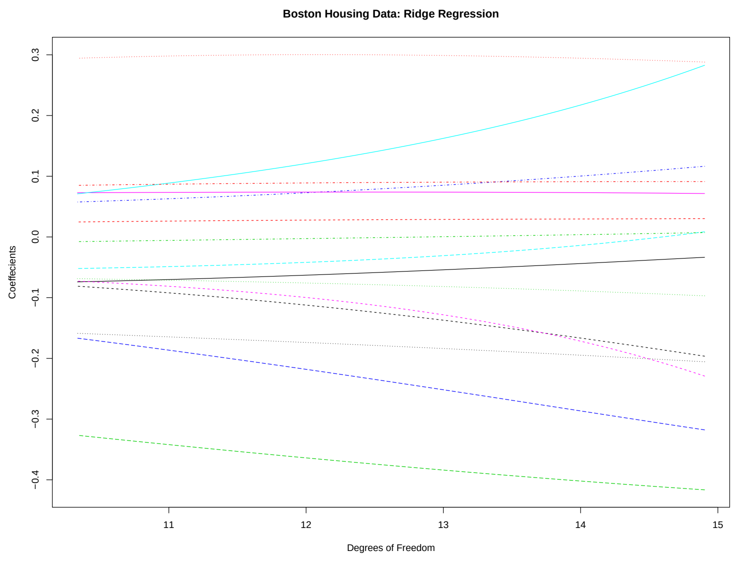
```r
plot(x = lambda_values, y = cv_error_array, xlab = 'Lambda', ylab='Cross Validation error')
```



```r
plot(x = dof_array, y = cv_error_array, xlab = 'Degrees of Freedom', ylab='Cross Validation error')
```



```r
matplot(dof_array, matrix_of_ridge_estimates, type = "l",
        xlab = "Degrees of Freedom", ylab = "Coeffecients", xlim = c(min(dof_array), max(dof_array)))
text(rep(1, 15), matrix_of_ridge_estimates[1, ], colnames(X))
title("Boston Housing Data: Ridge Regression")
```



b. [25 points] Following the setting of part a), with $\lambda = 10$, recover the original solution based on the unstandardized data (with intercept). Again, you cannot refit the model on the original data. Provide a rigorous mathematical derivation. In this case, is your solution the same as a ridge model (with your previous code) fitted on the original data? Make sure that you check the model fitting results from either model. What is the difference? Please list all possible reasons that may cause this difference.

```r
library(mlbench)
library(leaps)

data(BostonHousing2)

# on scaled data
df = as.data.frame(BostonHousing2)

# remove columns : medv, town and tract (as mentioned in the Ques)
df$medv = NULL
df$town = NULL
df$tract = NULL

predictors = setdiff(colnames(df), c('cmedv'))

numerical_predictors = predictors

df$chas = as.numeric(df$chas)

# scale the numerical predictors (mean= 0, sd = 1)
for (numerical_predictor in numerical_predictors) {
  df[, numerical_predictor] = scale(df[, numerical_predictor])
}

# scale the outcome variable
df[, 'cmedv'] = scale(df[, 'cmedv'])

X = data.matrix(df[, predictors])
```

```r
Y = data.matrix(df[, 'cmedv'])
Xt_X = t(X) %*% X
Xt_Y = t(X) %*% Y
I = diag(dim(Xt_X)[1])

lambda_value = 10

beta_hat_ridge = solve(Xt_X + lambda_value*I) %*% Xt_Y

beta_hat_ridge_scaled = beta_hat_ridge

# ridge coefficient estimates on scaled data
beta_hat_ridge_scaled
```

```
##                 [,1]
## lon     -0.042199724
## lat      0.029678266
## crim    -0.090134412
## zn       0.102707722
## indus   -0.010803782
## chas     0.072853630
## nox     -0.171107562
## rm       0.293547542
## age      0.004270751
## dis     -0.291578395
## rad      0.226580857
## tax     -0.179527471
## ptratio -0.196434192
## b        0.091029089
## lstat   -0.404452653
```

```r
# estimate unscaled parameters from scaled parameters
df_unscaled = as.data.frame(BostonHousing2)

df_unscaled$medv = NULL
df_unscaled$town = NULL
df_unscaled$tract = NULL

df_unscaled$chas = as.numeric(df_unscaled$chas)

unscaled_estimates = c()

for (numerical_predictor in numerical_predictors){
  unscaled_estimate = beta_hat_ridge_scaled[numerical_predictor, ] * sd(df_unscaled[, 'cmedv']) / sd(df_
  unscaled_estimates = c(unscaled_estimates, unscaled_estimate)
}

# intercept term calculation
product_array = c()
for (numerical_predictor in numerical_predictors){
  product = unscaled_estimates[numerical_predictor] * mean(df_unscaled[, numerical_predictor])
  product_array = c(product_array, product)
}
```

```
intercept = mean(df_unscaled[, 'cmedv']) - sum(product_array)

# ridge coefficient estimates on unscaled data (retrived from coefficient estimates on scaled data)
c(intercept, unscaled_estimates)
```

```
##                        lon           lat          crim            zn         indus
## -5.230568e+02 -5.138698e+00  4.411193e+00 -9.621876e-02  4.043659e-02 -1.446022e-02
##          chas           nox            rm           age           dis           rad
##  2.633742e+00 -1.355861e+01  3.836236e+00  1.393122e-03 -1.271459e+00  2.389392e-01
##           tax       ptratio             b         lstat
## -9.780948e-03 -8.331357e-01  9.155445e-03 -5.200565e-01
```

```
# fit a ridge regression on unscaled data
df = as.data.frame(BostonHousing2)

# remove columns : medv, town and tract (as mentioned in the Ques)
df$medv = NULL
df$town = NULL
df$tract = NULL

predictors = setdiff(colnames(df), c('cmedv'))

numerical_predictors = predictors

X = data.matrix(df[, predictors])
X = cbind(Intercept = 1, X)
Y = data.matrix(df[, 'cmedv'])
Xt_X = t(X) %*% X
Xt_Y = t(X) %*% Y
I = diag(dim(Xt_X)[1])

lambda_value = 10

beta_hat_ridge = solve(Xt_X + lambda_value*I) %*% Xt_Y

# ridge coefficient estimates on unscaled data
beta_hat_ridge
```

```
##                  [,1]
## Intercept -0.01847072
## lon       -0.29044471
## lat        0.11657382
## crim      -0.09957870
## zn         0.05069658
## indus     -0.04059860
## chas       1.94239403
## nox       -2.36421555
## rm         3.66811686
## age       -0.01054512
## dis       -1.27221510
## rad        0.27788415
## tax       -0.01438424
## ptratio   -0.76950401
## b          0.00992302
## lstat     -0.56600607
```

Que. 2. b. Let $z_i = \dfrac{x_i - \bar{x}_i}{\sigma_i}$

Linear regression on scaled data :

$$\frac{\hat{y} - \bar{y}}{\sigma_y} = \hat{\beta}_0 + \sum_{i=1}^{p} z_i \hat{\beta}_i$$

$$= \hat{\beta}_0 + \sum_{i=1}^{p} \left(\frac{x_i - \bar{x}_i}{\sigma_i} \right) \hat{\beta}_i$$

$$\hat{y} = \bar{y} + \hat{\beta}_0 \sigma_y + \sum^{p} \left(\frac{x_i - \bar{x}_i}{\sigma_i} \right) \hat{\beta}_i \sigma_y$$

$$\hat{y} = \underbrace{\left(\bar{y} + \hat{\beta}_0 \sigma_y - \sum^{p} \frac{\hat{\beta}_i \bar{x}_i \sigma_y}{\sigma_i} \right)}_{\substack{\beta_0 \\ \text{(unscaled intercept)}}} + \sum \underbrace{\left(\frac{\hat{\beta}_i \sigma_y}{\sigma_i} \right)}_{\substack{\beta_i \\ \text{(unscaled predictor)}}} x_i$$

It is not possible to retrieve unscaled estimates from scaled estimates. Reason :
While performing Ridge regression on scaled data, since we centred the outcome variable, the intercept term disappeared, and it is not penalized. However, while performing Ridge regression on unscaled data, there is intercept term present, which is penalized as well. Penalization of the intercept would make the procedure depend on the origin chosen for Y ; that is, adding a constant c to each of the targets yi would not simply result in a shift of the predictions by the same amount c. The coefficients gets penalized without an intercept term. Hence, the panalty on the variables in both the cases is not applied in the same way, and it is not possible to retrieve the unscaled parameters estimates from the scaled parameter estimates (when the intercept term is missing).

c. [15 points] A researcher is interested in only penalizing a subset of the variables, and leave the rest of them unpenalized. In particular, the categorical variables `zn`, `chas`, `rad` should not be penalized. You should use the data in part 2), which does not concern the intercept. Following the derivation during our lecture:

- Write down the objective function of this new model
- Derive the theoretical solution of this model and implement it with $\lambda = 10$
- Check your model fitting results and report the residual sum of squares

```
library(mlbench)
library(leaps)

data(BostonHousing2)

# on scaled data
df = as.data.frame(BostonHousing2)

# remove columns : medv, town and tract (as mentioned in the Ques)
df$medv = NULL
df$town = NULL
df$tract = NULL

predictors = setdiff(colnames(df), c('cmedv'))

numerical_predictors = predictors

df$chas = as.numeric(df$chas)

# scale the numerical predictors (mean= 0, sd = 1)
for (numerical_predictor in numerical_predictors) {
  df[, numerical_predictor] = scale(df[, numerical_predictor])
}

# scale the outcome variable
df[, 'cmedv'] = scale(df[, 'cmedv'])

X = data.matrix(df[, predictors])
Y = data.matrix(df[, 'cmedv'])
Xt_X = t(X) %*% X
Xt_Y = t(X) %*% Y
I = diag(dim(Xt_X)[1])
I[4, 4] = 0
I[6, 6] = 0
I[11, 11] = 0

lambda_value = 10

beta_hat_ridge = solve(Xt_X + lambda_value*I) %*% Xt_Y

beta_hat_ridge_scaled = beta_hat_ridge

# ridge coefficient estimates
beta_hat_ridge_scaled
```

```
##                  [,1]
## lon     -0.040277616
## lat      0.031478940
## crim    -0.094748374
## zn       0.109184439
## indus   -0.004793193
## chas     0.073569228
## nox     -0.174905051
## rm       0.290764008
## age      0.005436640
## dis     -0.293954170
## rad      0.257998369
## tax     -0.205259442
## ptratio -0.198751882
## b        0.091876337
## lstat   -0.405234537
```

```
y_hat = X %*% beta_hat_ridge_scaled
y = df[, 'cmedv']
RSS = sum((y - y_hat)^2)

# residual sum of squares
RSS
```

```
## [1] 128.6341
```

Ques. 2. c.

Objective function, $L = ||y - X\beta||_2^2 + \lambda ||\beta||_2^2$

To find β which minimizes above objective function, we take derivative wrt β, and equate it to 0 :

$$\frac{dL}{d\beta} = -2X^T(y - X\beta) + 2\lambda\beta$$

$$X^T y = (X^T X + \lambda I)\beta$$

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

For variables which should not be penalized, their values in I should be set to 0.

Zn :	$I[4,4] = 0$
chas :	$I[6,6] = 0$
rad :	$I[11,11] = 0$