P9120hw1_code

Ze Li

```
library(ISLR)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
library(leaps)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(pls)
```

```
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:caret':
```

```
##
##     R2
##
## The following object is masked from 'package:stats':
##
##     loadings
```

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

Question 2

i

```r
set.seed(1)

p <- 10  # number of independent normal variables
n <- 100  # number of samples per variable
mu <- rnorm(p, mean = 0, sd = 1) # true mean

# Training sample (X_j's)
training_sample <- matrix(rnorm(n * p, mean = rep(mu, each = n), sd = 1), nrow = n, ncol = p)

# Sample means for each variable X_j
sample_means <- colMeans(training_sample)

# Then mean Xmax is an estimate of mu_max based on the training sample
mu_max_est <- max(sample_means)
mu_max_est
```

```
## [1] 1.614055
```

ii

```r
# Generate B = 1000 bootstrap samples from the training sample.
B <- 1000

# Initialize a vector to store bootstrap estimates of mu_max
bootstrap_mu_max <- numeric(B)

# For each bootstrap sample b, compute the estimate of mu_max as previously
for (b in 1:B) {
  bootstrap_sample <- training_sample[sample(1:n, n, replace = TRUE), ]
  bootstrap_means <- colMeans(bootstrap_sample)
  bootstrap_mu_max[b] <- max(bootstrap_means)
}
summary(bootstrap_mu_max)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.262   1.545   1.614   1.613   1.681   1.921
```

iii

```
# Construct 95% confidence intervals for mu_max using the three bootstrap inference methods presented i
# Percentile method
percentile_ci <- quantile(bootstrap_mu_max, probs = c(0.025, 0.975))
percentile_ci
```

```
##     2.5%    97.5%
## 1.421194 1.797888
```

```
# Basic method
basic_ci <- 2 * mu_max_est - quantile(bootstrap_mu_max, probs = c(0.975, 0.025))
basic_ci
```

```
##    97.5%     2.5%
## 1.430222 1.806916
```

```
# BCa method
# Function to compute jackknife estimates
jackknife <- function(training_sample, mu_max_est) {
  n <- nrow(training_sample)
  jack_means <- numeric(n)
  for (i in 1:n) {
    jackknife_sample <- training_sample[-i, ]
    jack_means[i] <- max(colMeans(jackknife_sample))
  }
  return(jack_means)
}

# Bias-correction factor z_0
z0 <- qnorm(mean(bootstrap_mu_max < mu_max_est))

# Acceleration factor a
jack_means <- jackknife(training_sample, mu_max_est)
mean_jack <- mean(jack_means)
acceleration <- sum((mean_jack - jack_means)^3) / (6 * (sum((mean_jack - jack_means)^2))^(3/2))

# BCa CI
alpha <- c(0.025, 0.975)
z_alpha <- qnorm(alpha)
adjusted_alpha <- pnorm(z0 + (z0 + z_alpha) / (1 - acceleration * (z0 + z_alpha)))
bca_ci <- quantile(bootstrap_mu_max, probs = adjusted_alpha)
bca_ci
```

```
## 2.310458% 97.30143%
##  1.419709  1.795951
```

a-f

```r
set.seed(123)  # For reproducibility

# Function to generate a training set based on the scenario
generate_training_set <- function(p, mu) {
  n <- 100  # Number of observations
  training_sample <- matrix(rnorm(n * p, mean = rep(mu, each = n), sd = 1), nrow = n, ncol = p)
  return(training_sample)
}

# Function to perform bootstrap confidence intervals
bootstrap_ci <- function(training_sample, B = 1000) {
  n <- nrow(training_sample)
  p <- ncol(training_sample)

  # Compute the sample means for the training set
  sample_means <- colMeans(training_sample)
  mu_max_est <- max(sample_means)

  # Bootstrap sampling
  bootstrap_mu_max <- numeric(B)
  for (b in 1:B) {
    bootstrap_sample <- training_sample[sample(1:n, n, replace = TRUE), ]
    bootstrap_means <- colMeans(bootstrap_sample)
    bootstrap_mu_max[b] <- max(bootstrap_means)
  }

  # Confidence Intervals
  # Percentile Method
  percentile_ci <- quantile(bootstrap_mu_max, probs = c(0.025, 0.975))

  # Basic Method
  basic_ci <- 2 * mu_max_est - quantile(bootstrap_mu_max, probs = c(0.975, 0.025))

  # BCa Method
  jackknife <- function(training_sample) {
    jack_means <- numeric(n)
    for (i in 1:n) {
      jackknife_sample <- training_sample[-i, ]
      jack_means[i] <- max(colMeans(jackknife_sample))
    }
    return(jack_means)
  }

  z0 <- qnorm(mean(bootstrap_mu_max < mu_max_est))
  jack_means <- jackknife(training_sample)
  mean_jack <- mean(jack_means)
  acceleration <- sum((mean_jack - jack_means)^3) / (6 * (sum((mean_jack - jack_means)^2))^(3/2))
  alpha <- c(0.025, 0.975)
  z_alpha <- qnorm(alpha)
  adjusted_alpha <- pnorm(z0 + (z0 + z_alpha) / (1 - acceleration * (z0 + z_alpha)))
  bca_ci <- quantile(bootstrap_mu_max, probs = adjusted_alpha)

  return(list(percentile = percentile_ci, basic = basic_ci, bca = bca_ci, mu_max_est = mu_max_est))
```

```r
}

# Simulation Parameters
scenarios <- list(
  list(p = 2, mu = rep(1, 2)),          # (a)
  list(p = 2, mu = 1:2),                # (b)
  list(p = 5, mu = rep(1, 5)),          # (c)
  list(p = 5, mu = 1:5),                # (d)
  list(p = 10, mu = rep(1, 10)),        # (e)
  list(p = 10, mu = 1:10)               # (f)
)

M <- 1000  # number of training sets
B <- 1000  # number of bootstrap samples

# Initialize a matrix to store coverage rates for each scenario and method
coverage_rates <- matrix(NA, nrow = length(scenarios), ncol = 3)
colnames(coverage_rates) <- c("Percentile", "Basic", "BCa")

# Loop over each scenario
for (scenario_idx in 1:length(scenarios)) {
  scenario <- scenarios[[scenario_idx]]
  p <- scenario$p
  mu <- scenario$mu

    # Modify sample size for scenario (e) to check stability
# if (scenario_idx == 5) {
#    n <- 1000  # Larger sample size for (e)
#  } else {
#    n <- 100  # Default sample size for other scenarios
#  }


  # Initialize counters for coverage
  coverage_percentile <- 0
  coverage_basic <- 0
  coverage_bca <- 0

  true_mu_max <- max(mu)  # True mu_max for this scenario

  # Loop over M training sets
  for (i in 1:M) {
    training_sample <- generate_training_set(p, mu)

    # Compute the bootstrap confidence intervals
    ci <- bootstrap_ci(training_sample, B = B)

    # Check true mu_max is within the confidence intervals
    if (true_mu_max >= ci$percentile[1] && true_mu_max <= ci$percentile[2]) {
      coverage_percentile <- coverage_percentile + 1
    }
    if (true_mu_max >= ci$basic[1] && true_mu_max <= ci$basic[2]) {
      coverage_basic <- coverage_basic + 1
```

```
    }
    if (true_mu_max >= ci$bca[1] && true_mu_max <= ci$bca[2]) {
      coverage_bca <- coverage_bca + 1
    }
  }

  # Compute coverage rates
  coverage_rates[scenario_idx, "Percentile"] <- coverage_percentile / M
  coverage_rates[scenario_idx, "Basic"] <- coverage_basic / M
  coverage_rates[scenario_idx, "BCa"] <- coverage_bca / M
}

# Print the coverage rates
rownames(coverage_rates) <- c("(a) p=2, mu=1", "(b) p=2, mu=j", "(c) p=5, mu=1",
                              "(d) p=5, mu=j", "(e) p=10, mu=1", "(f) p=10, mu=j")
print(coverage_rates)
```

```
##                Percentile Basic   BCa
## (a) p=2, mu=1       0.892 0.948 0.928
## (b) p=2, mu=j       0.946 0.953 0.947
## (c) p=5, mu=1       0.309 0.883 0.686
## (d) p=5, mu=j       0.946 0.949 0.943
## (e) p=10, mu=1      0.006 0.820 0.182
## (f) p=10, mu=j      0.926 0.920 0.924
```

Question 3

```
prostate <- read.table("https://hastie.su.domains/ElemStatLearn/datasets/prostate.data")
prostate <- na.omit(prostate)

train_X <- prostate |>
  dplyr::filter(train) |>
  dplyr::select(-train, -lpsa)
train_Y <- prostate |>
  dplyr::filter(train) |>
  dplyr::select(lpsa)

train <- prostate |>
  dplyr::filter(train) |>
  dplyr::select(-train)

test_X <- prostate |>
  dplyr::filter(!train) |>
  dplyr::select(-train, -lpsa)
test_Y <- prostate |>
  dplyr::filter(!train) |>
  dplyr::select(lpsa)

X <- as.matrix(train_X)
Y <- as.matrix(train_Y)
```

Part (a): Best-subset linear regression with k chosen by 5-fold cross-validation

```
# 5 folds
k = 5
n = 8 # 8 variables
set.seed(1)
folds = sample(1:k, nrow(X), replace = TRUE)

predict.regsubsets = function(object,newdata,id,...){
      form = as.formula(object$call[[2]]) # Extract the formula used regsubsets()
      mat = model.matrix(form,newdata) # Build the model matrix
      coefi = coef(object,id=id) # Extract the coefficiants of the ith model
      xvars = names(coefi) # Pull out names of predictors used in the ith model
      mat[,xvars] %*% coefi # Make predictions using matrix multiplication
}
```

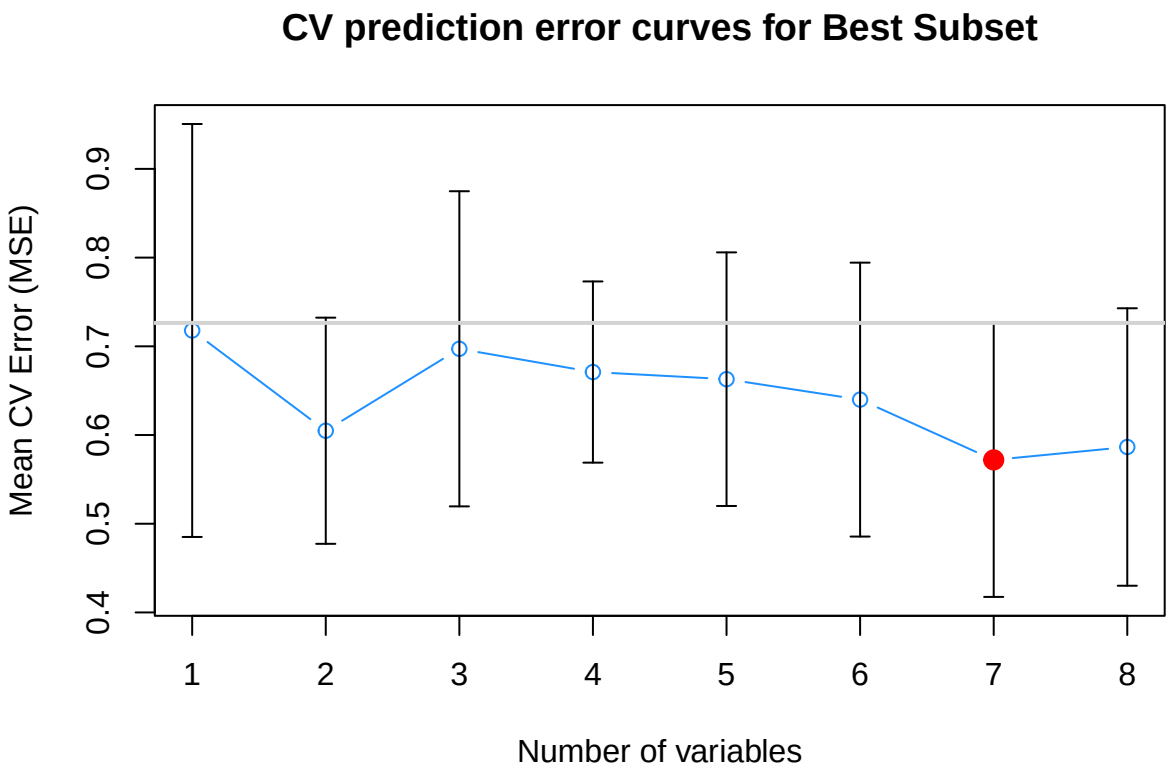
```
# cv_error
cv_errors = matrix(NA, k, n, dimnames = list(NULL, paste(1:n)))
# Outer loop iterates over all folds
for(j in 1:k){
    # best subset selection on the full dataset - the jth fold
    best_fit = leaps::regsubsets(lpsa~., data = train[folds!=j,], nvmax = n, method = "exhaustive")
    # Inner loop iterates over each size i
    for(i in 1:n){
        # Predict the values of the current fold from the "best subset" model on i predictors
```

```
        pred = predict(best_fit, train[folds == j,], id = i)
        # Calculate the MSE, store it in the matrix we created above
        cv_errors[j,i] = mean((Y[folds == j] - pred)^2)
    }
}

# Take the mean of over all folds for each model size
mean_cv_errors = apply(cv_errors, 2, mean)
mean_cv_sd = apply(cv_errors, 2, sd)
mean_cv_up = mean_cv_errors + mean_cv_sd
mean_cv_lo = mean_cv_errors - mean_cv_sd

# Find the model size with the min cv error
min = which.min(mean_cv_errors)

plot(mean_cv_errors, type = 'b',
     ylim = c(min(mean_cv_lo), max(mean_cv_up)),
     col = "dodgerblue",
     xlab = "Number of variables",
     ylab = "Mean CV Error (MSE)",
     main = "CV prediction error curves for Best Subset")
# error bar
arrows(1:n, mean_cv_up, 1:n, mean_cv_lo, length=0.05, angle=90, code=3)
# model size point with min error
points(min, mean_cv_errors[min][1], col = "red", cex = 2, pch = 20)
# mean cv error line
abline(h = (mean_cv_up)[which.min(mean_cv_errors)], col = "lightgray", lwd = 2)
```



Best-subset model shows that best variable size of 7 with the lowest CV error. However, if we use 1se

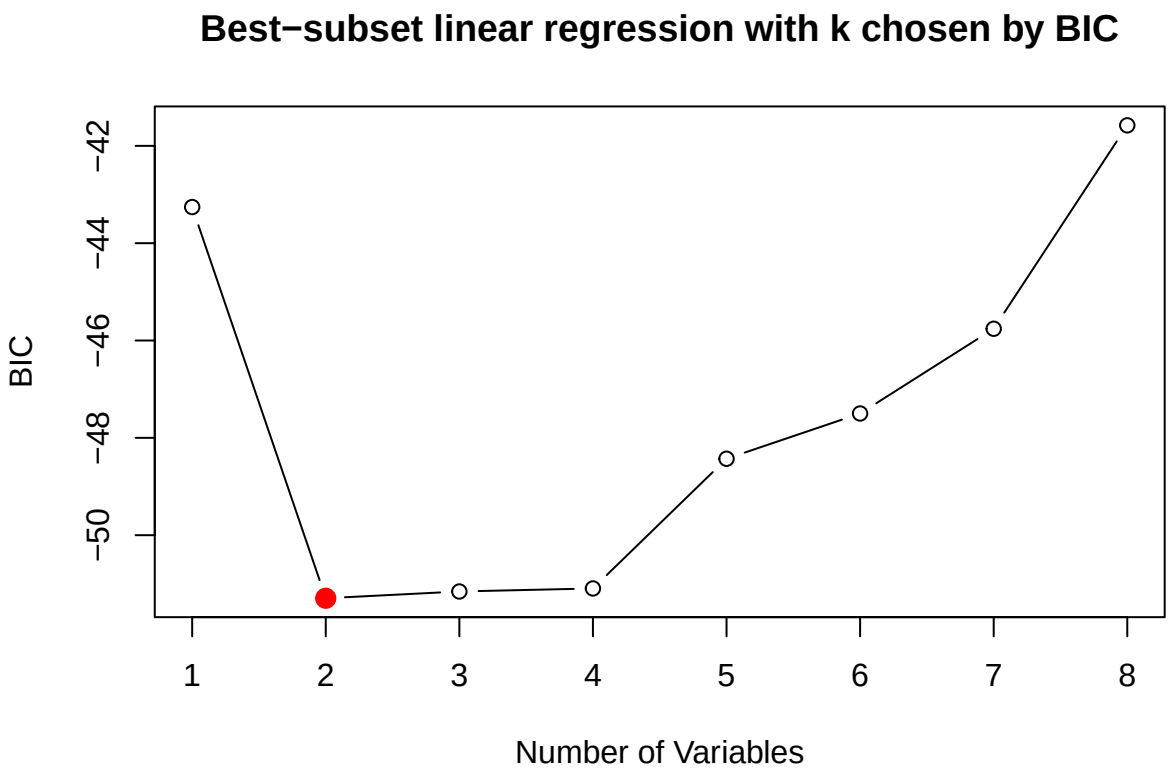
standard, we select the model of size 2.

Part (b): Best-subset linear regression with k chosen by BIC.

```
best <- leaps::regsubsets(x = X, y = Y, method = "exhaustive")
reg_summary <- summary(best);reg_summary
```

```
## Subset selection object
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     " "
## 5  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     "*"
## 6  ( 1 ) "*"    "*"     " " "*"  "*" "*" " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

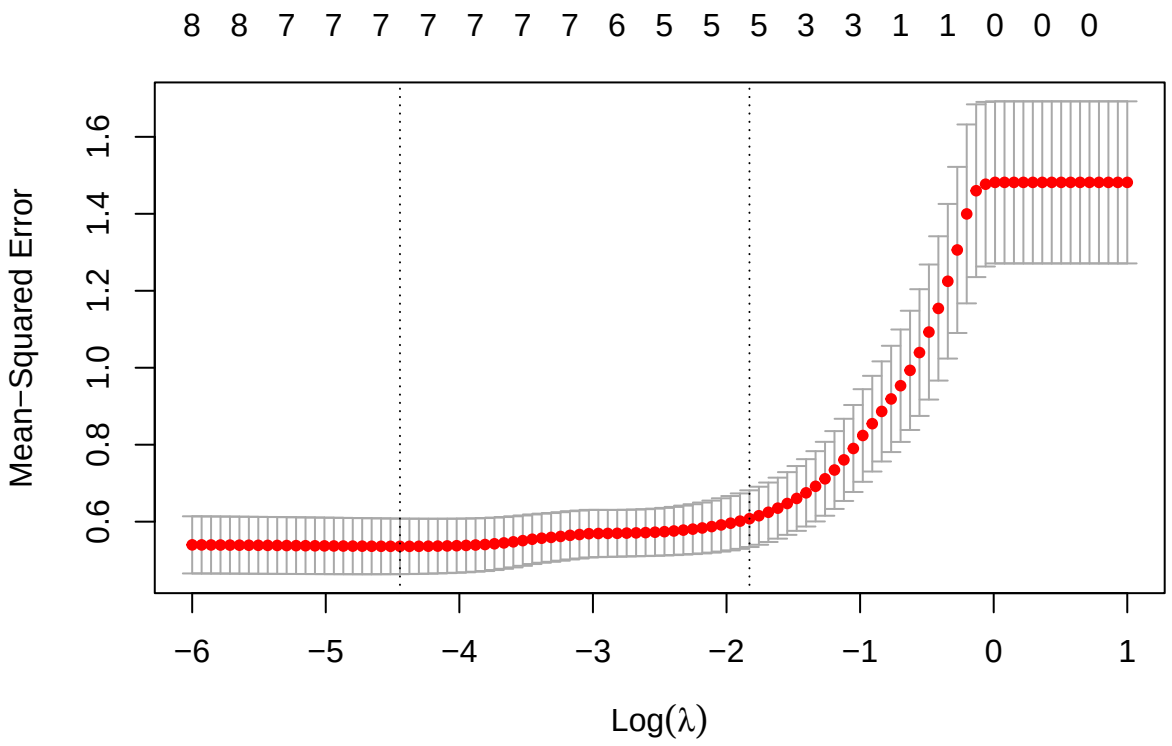
```
plot(reg_summary$bic,
     xlab = "Number of Variables",
     ylab = "BIC",
     type = "b",
     main = "Best-subset linear regression with k chosen by BIC")
bic_min = which.min(reg_summary$bic)
# model size point with min bic
points(bic_min, reg_summary$bic[bic_min], col = "red", cex = 2, pch = 20)
```



Best-subset model shows that best variable size of 2. The BIC decreases at the beginning to the lowest point, but then keep increasing when model complexity increases.Therefore, BIC tends to select the parsimony model.

Part (c): Lasso regression with λ chosen by 5-fold cross-validation.

```
# fit lasso
set.seed(1)
cv.lasso <- glmnet::cv.glmnet(X, Y,
                              alpha = 1, lambda = exp(seq(1, -6, length = 100)),
                              nfolds = 5)
plot(cv.lasso)
```




```
cv.lasso$lambda.min
```

```
## [1] 0.01174363
```

```
cv.lasso$lambda.1se
```

```
## [1] 0.1606893
```

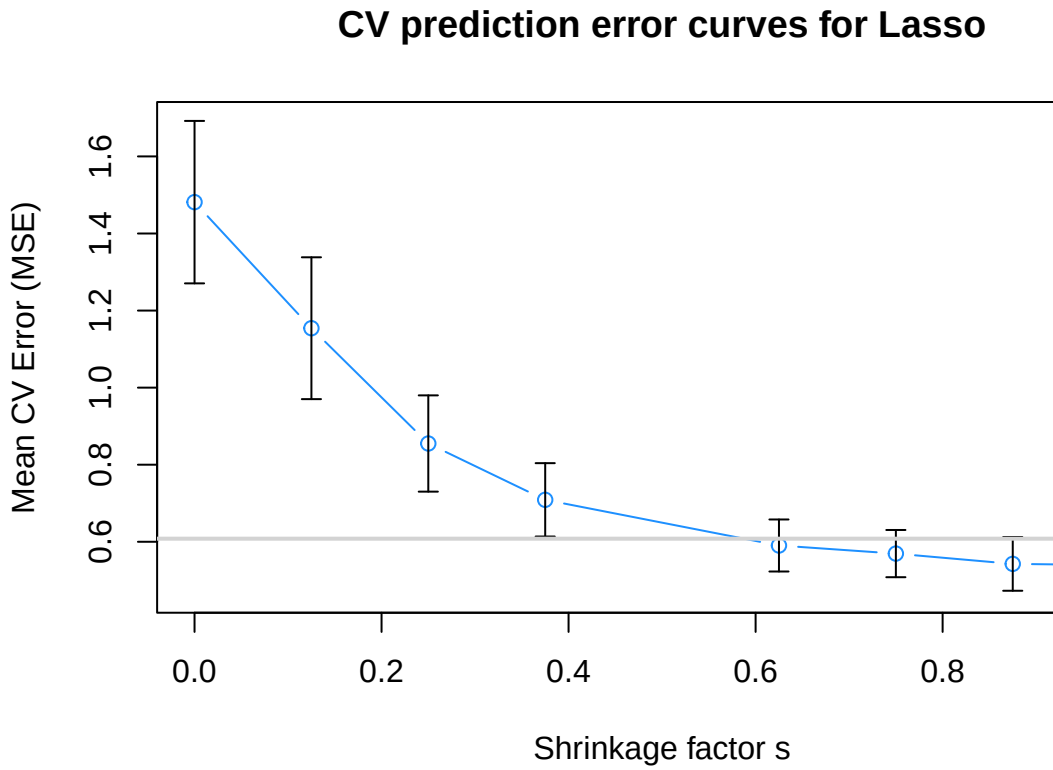
The chosen lambda using 1se rule is 0.0117436. The minimum MSE lambda is 0.1606893.

```
# add complexity
lasso.table <- data.frame(complexity = cv.lasso$glmnet.fit$df/8,
                          mse = cv.lasso$cvm,
                          mse_u = cv.lasso$cvup,
                          mse_l = cv.lasso$cvlo)

lasso.table <- lasso.table |>
  dplyr::group_by(complexity) |>
  dplyr::summarise_at(dplyr::vars(mse, mse_u, mse_l), mean)

# MSE vs model complexity
plot(lasso.table$complexity, lasso.table$mse, type = 'b',
     ylim = c(min(lasso.table$mse_l), max(lasso.table$mse_u)),
     col = "dodgerblue",
     xlab = "Shrinkage factor s",
     ylab = "Mean CV Error (MSE)",
```

```
     main = "CV prediction error curves for Lasso")
# error bar
arrows(lasso.table$complexity, lasso.table$mse_u, lasso.table$complexity,
       lasso.table$mse_l, length=0.05, angle=90, code=3)
# model size point with min mse
points(which.min(lasso.table$mse)/8,
       lasso.table$mse[which.min(lasso.table$mse)],
       col = "red", cex = 2, pch = 20)
# 1se line
abline(h = (cv.lasso$cvm + cv.lasso$cvsd)[which.min(cv.lasso$cvm)],
       col = "lightgray", lwd = 2)
```



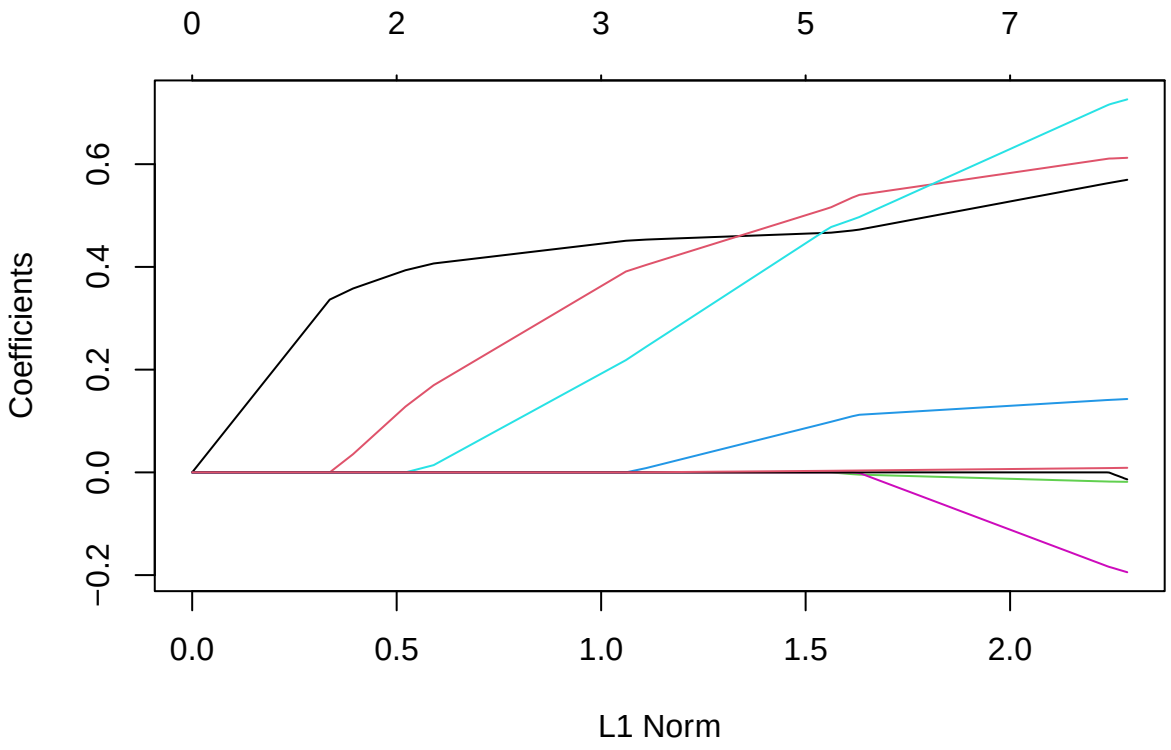
Model complexity increases:

The plot shows 1.0 shrinkage factor with variable subset of 8, but 0.6 with 1se rule corresponding variable subset around 5.

Part (d): Lasso regression with λ chosen by BIC.

```
lasso.bic <- glmnet::glmnet(x = X, y = Y, standardize = TRUE,
                            alpha = 1, lambda = exp(seq(1, -6, length = 100)))
plot(lasso.bic)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

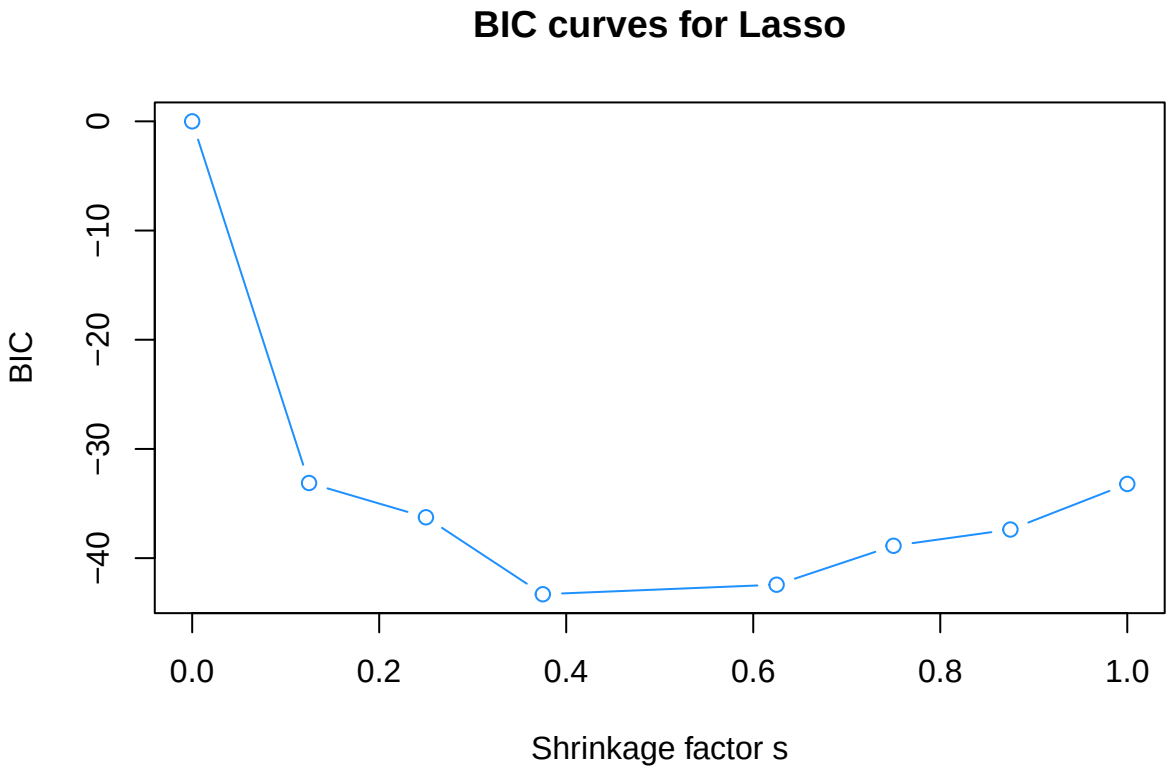



```r
# calculate BIC
tLL <- lasso.bic$nulldev - deviance(lasso.bic)
k <- lasso.bic$df
n <- lasso.bic$nobs
BIC <- log(n)*k - tLL

# mean BIC for each shrinkage factor s
lasso.bic.table <- data.frame(complexity = lasso.bic$df/8, bic = BIC)

lasso.bic.table <- lasso.bic.table |>
  dplyr::group_by(complexity) |>
  dplyr::summarise_at(dplyr::vars(bic), base::min)

plot(x = lasso.bic.table$complexity, y = lasso.bic.table$bic, type = 'b',
     col = "dodgerblue",
     xlab = "Shrinkage factor s",
     ylab = "BIC",
     main = "BIC curves for Lasso")
```




```
lasso.bic$lambda[which.min(BIC)]
```

```
## [1] 0.2132149
```

We select the model with lowest BIC score, so we will have the shrinkage factor around 0.4 and the corresponding lambda is 0.2132149.