Math574M_Hw5

Saheed Adisa, Ganiyu

2023-11-02

6. Download the prostate cancer data set from https://hastie.su.domains/ElemStatLearn/. The data set contains eight predictors (columns 1-8), $\mathbf{X} \in R^8$, and the outcome Y (column 9). The last column (column 10) is the train/test indicator; there are $n = 67$ training data points and $\bar{n} = 30$ test data points. Denote the training set by $\{(\mathbf{x}_i, y_i), i = 1, \cdots, n\}$ and the test set by $\{(\tilde{\mathbf{x}}_i, \tilde{y}_i), i = 1, \cdots, \tilde{n}\}$.

Analysis: Consider the linear regression of Y on $\mathbf{X}$. For any fitted model $\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}^T\mathbf{x}$, define its training error by TrainErr $= \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \hat{f}(\mathbf{x}_i)\right]^2$ and its test error by TestErr $= \frac{1}{n}\sum_{i=1}^{\bar{n}}\left(\tilde{y}_i - \hat{f}(\tilde{\mathbf{x}}_i)\right]^2$.

(a) Fit the standard linear regression model using OLS. Report its R^2, p-values of regression coefficients, the set of significant predictors (at the level $\alpha = 0.05$), TrainErr, and TestErr. Use R functions " lm() " and "summary()" to do the analysis.

```
# loading the dataset from the book site
prostate_data <- read.table(
  "https://hastie.su.domains/ElemStatLearn/datasets/prostate.data")
x_train <- subset(prostate_data, train == TRUE)[,1:9]
y_train <- subset(prostate_data, train == TRUE)[,9]
x_test <- subset(prostate_data, train == FALSE)[,1:9]
y_test <- subset(prostate_data, train == FALSE)[,9]
```

fitting and summary of the linear model

```
lm_model <- lm(lpsa ~ ., data = x_train)
summary(lm_model)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = x_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.429170   1.553588   0.276  0.78334
## lcavol       0.576543   0.107438   5.366 1.47e-06 ***
## lweight      0.614020   0.223216   2.751  0.00792 **
## age         -0.019001   0.013612  -1.396  0.16806
## lbph         0.144848   0.070457   2.056  0.04431 *
## svi          0.737209   0.298555   2.469  0.01651 *
## lcp         -0.206324   0.110516  -1.867  0.06697 .
## gleason     -0.029503   0.201136  -0.147  0.88389
## pgg45        0.009465   0.005447   1.738  0.08755 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12
```

computing the train and test error

```r
# Training error
train_error <- mean((y_train - predict(lm_model,x_train[,1:8]))^2)

# Test error
test_error <- mean((y_test - predict(lm_model, newdata = x_test[,1:8]))^2)
cat("Train Error=", train_error, "\n", "Test Error=", test_error)
```

```
## Train Error= 0.4391998
##  Test Error= 0.521274
```

(b) Apply forward selection to select variables use R function "regsubsets()" in the package "leaps". You should get a sequence of eight models, $\widehat{M}_1, \cdots, \widehat{M}_8$, in the increasing order of model size. For each model $\widehat{M}_j, j = 1, \cdots, 8$, report its regression coefficients, calculate its TrainErr, and calculate its BIC using the formula

$$BIC\left(\widehat{M}_j\right) = n \log(\text{ TrainErr }) + \log(n)\left|\widehat{M}_j\right|,$$

where $\left|\widehat{M}_j\right|$ is the number of variables in the model (including the intercept). Choose the best model by minimizing BIC, and report the set of important variables. Furthermore, use the selected variables to refit the OLS and report TestErr.

```r
library(leaps)
# Forward selection
forward_model <- regsubsets(lpsa ~ ., data = x_train, nvmax = 8, method = "forward")
summary(forward_model)
```

```
## Subset selection object
## Call: regsubsets.formula(lpsa ~ ., data = x_train, nvmax = 8, method = "forward")
## 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: forward
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     " "
## 5  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     "*"
## 6  ( 1 ) "*"    "*"     " " "*"  "*" "*" " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

computing TrainErr and BIC for each model and select the best model

```r
# Get the model with the minimum BIC
best_model <- which.min(summary(forward_model)$bic)

# Retrieve the selected model
selected_model <- summary(forward_model)$which[best_model, ]

# Extract the variables in the selected model
selected_variables <- names(selected_model[selected_model == 1])
selected_variables <- selected_variables[-1]

# Fit the OLS model using the selected variables
selected_lm_model <- lm(lpsa ~ ., data = x_train[, c("lpsa", selected_variables)])

# Calculate TrainErr
selected_train_error <- mean((y_train - predict(selected_lm_model))^2)

# Calculate BIC
n <- nrow(x_train)
bic <- n * log(selected_train_error) + log(n) * (length(selected_variables) + 1)

cat("Selected model var: ", selected_variables,
    "(", length(selected_variables), ")", "\n")
```

```
## Selected model var:  lcavol lweight ( 2 )
```

```r
cat("Selected train error=", selected_train_error, "\n")
```

```
## Selected train error= 0.5536096
```

```r
cat("BIC=", bic)
```

```
## BIC= -27.00272
```

Computing the TestErr using the selected model based on BIC

```r
selected_test_error <- mean((y_test - predict(selected_lm_model, newdata = x_test))^2)
cat("Selected test error=", selected_test_error)
```

```
## Selected test error= 0.4924823
```

Comment: There is a better performance on test data for selected variables with 0.4925 error rate compare to its error rate with 0.521274 when the full variables were used. The reverse is the case for train set under the selected variables.

(c) In part (b), replace BIC by AIC,

$$AIC\left(\widehat{M}_j\right) = n\log(\text{ TrainErr }) + 2\left|\widehat{M}_j\right|.$$

Choose the best model by minimizing AIC and report the set of selected variables. Furthermore, use the selected variables to refit the OLS and report TestErr.

```r
# Forward selection with AIC
forward_model_aic <- regsubsets(lpsa ~ ., data = x_train, nvmax = 8, method = "forward", really.big = T
summary(forward_model_aic)
```

```
## Subset selection object
## Call: regsubsets.formula(lpsa ~ ., data = x_train, nvmax = 8, method = "forward",
##     really.big = TRUE)
## 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: forward
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     " "
## 5  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     "*"
## 6  ( 1 ) "*"    "*"     " " "*"  "*" "*" " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

```
# 8 models in the order of forward selection
m1 <- lm(y_train~x_train[,1])
m2 <- lm(y_train~x_train[,1]+x_train[,2])
m3 <- lm(y_train~x_train[,1]+x_train[,2]+x_train[,5])
m4 <- lm(y_train~x_train[,1]+x_train[,2]+x_train[,4]+x_train[,5])
m5 <- lm(y_train~x_train[,1]+x_train[,2]+x_train[,4]+x_train[,5]+x_train[,8])
m6 <- lm(y_train~x_train[,1]+x_train[,2]+x_train[,4]+x_train[,5]+x_train[,6]+x_train[,8])
m7 <- lm(y_train~x_train[,1]+x_train[,2]+x_train[,3]+x_train[,4]+x_train[,5]+x_train[,6]+x_train[,8])
m8 <- lm(y_train~.,x_train[,1:8])
```

 # compute RSS for the four models

```
rss <- rep(0,8)
rss[1] <- sum((y_train-predict(m1))^2)
rss[2] <- sum((y_train-predict(m2))^2)
rss[3] <- sum((y_train-predict(m3))^2)
rss[4] <- sum((y_train-predict(m4))^2)
rss[5] <- sum((y_train-predict(m5))^2)
rss[6] <- sum((y_train-predict(m6))^2)
rss[7] <- sum((y_train-predict(m7))^2)
rss[8] <- sum((y_train-predict(m8))^2)
```

compute AIC and BIC

```
#bic <- rep(0,8)
aic <- rep(0,8)
for (i in 1:8){
#bic[i] = n*log(rss[i]/n)+log(n)*(1+i)
aic[i] = n*log(rss[i]/n)+2*(1+i)
}

# find the optimal model
#best_model_index_bic<-which.min(bic)
best_model_index_aic<-which.min(aic)

# Retrieve the selected model
selected_model <- summary(forward_model_aic)$which[best_model_index_aic, ]

# Extract the variables in the selected model
selected_variables <- names(selected_model[selected_model == 1])
selected_variables <- selected_variables[-1]

# Fit the OLS model using the selected variables
selected_lm_model_aic <- lm(lpsa ~ ., data = x_train[, c("lpsa", selected_variables)])

# Calculate TrainErr
selected_train_error <- mean((y_train - predict(selected_lm_model_aic))^2)

# Calculate BIC
```

```
n <- nrow(x_train)
aic <- n * log(selected_train_error) + 2 * (length(selected_variables) + 1)

cat("Selected model var: ", selected_variables,
    "(", length(selected_variables), ")", "\n")
```

```
## Selected model var:  lcavol lweight age lbph svi lcp pgg45 ( 7 )
```

```
cat("Selected train error=", selected_train_error, "\n")
```

```
## Selected train error= 0.4393627
```

```
cat("AIC=", aic)
```

```
## AIC= -39.10281
```

Computing the TestErr using the selected model based on AIC

```
selected_test_error <- mean((y_test - predict(selected_lm_model_aic,
                                              newdata = x_test))^2)
cat("Selected test error=", selected_test_error)
```

```
## Selected test error= 0.5165135
```

Comment: we observed that the test error of selected model based on BIC with 0.4925 is slightly better of the one based on AIC with test error 0.5165. Moreover, BIC based selection only used two variables while AIC based make used of seven variables. So, BIC wins in both model simplicity and error rate, hurray!!! :)

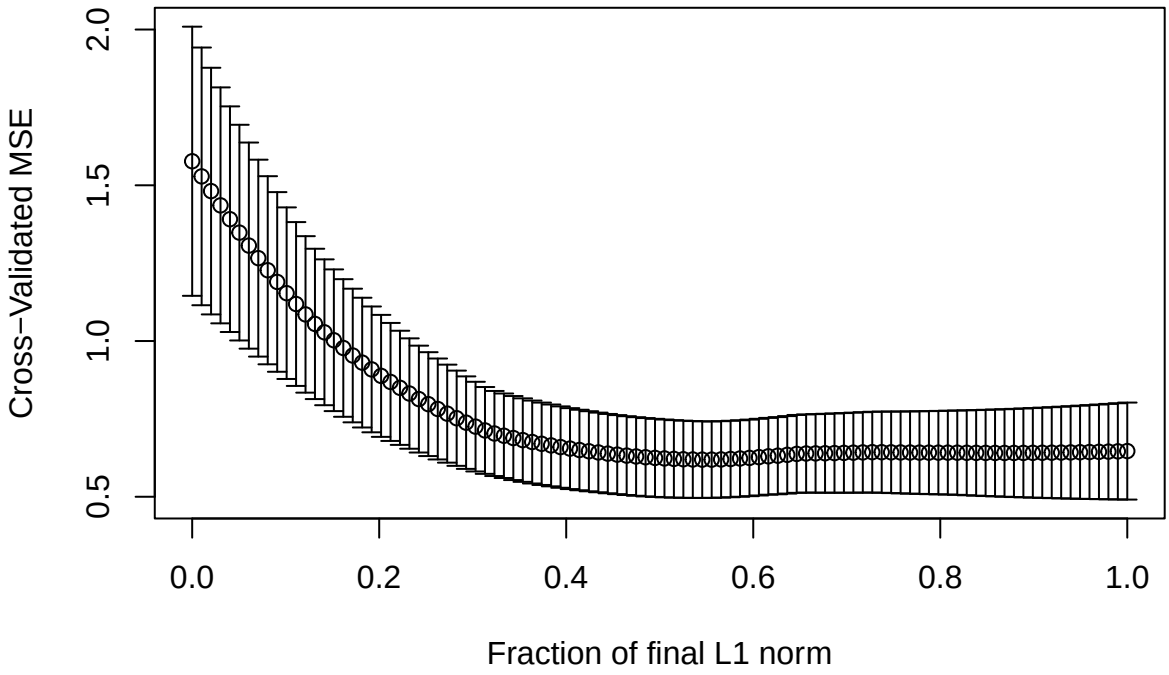
7. Fit the LASSO regression for the prostate cancer data set. You can use R functions "lars()" and "cv.lars()"

(a) Select the parameter with 5 -fold CV, using the minimum CV rule. Report the best s (or λ, or t), the selected model, the estimated regression coefficients, and the TestErr.

```
library(lars)
```

```
## Loaded lars 1.3
```

```
set.seed(1)
x_train2 <- as.matrix(x_train)
x_test2 <- as.matrix(x_test)
lasso.s<-seq(0,1,length=100)
lasso_fit <- lars(x = x_train2[, -9], y = x_train2[,9], type = "lasso")
cv_fit <- cv.lars(x_train2[, -9], x_train2[,9], K = 5,
                  index=lasso.s,mode="fraction")
```




```
lasso.mcv <- which.min(cv_fit$cv)

## minimum CV rule
best_lambda <- lasso.s[lasso.mcv]
lasso.coef1 <- predict(lasso_fit, s=best_lambda, type="coef", mode="frac")
test_predict <- predict(lasso_fit, newx = x_test2[,-9], s=best_lambda, type="fit",mode="frac")
test_error <- mean((x_test2[,9] - test_predict$fit)^2)
cat("Best tuning parameter s/lambda=", best_lambda, ",\n\n",
    "Its test error=",test_error, "\n\n", "Estimated Coeficients: \n" )
```

```
## Best tuning parameter s/lambda= 0.5454545 ,
##
##  Its test error= 0.4569334
##
##  Estimated Coeficients:
```

```
print(lasso.coef1$coefficients)
```

```
##      lcavol     lweight         age        lbph         svi         lcp
## 0.465614258 0.507733637 0.000000000 0.091907327 0.461016168 0.000000000
##     gleason       pgg45
## 0.000000000 0.002876921
```

(b) Select the parameter with 5-fold CV, using the one-standard deviation rule for CV. Report the best s (or λ, or t), the selected model, the estimated regression coefficients, and the TestErr.

```
## one-standard rule
bound<-cv_fit$cv[lasso.mcv]+cv_fit$cv.error[lasso.mcv]
```

```
best_lambda2<-lasso.s[min(which(cv_fit$cv<bound))]
lasso.coef2 <- coef(lasso_fit, s=best_lambda2, mode="frac")
test_predict2 <- predict(lasso_fit, newx = x_test2[,-9],
                         s=best_lambda2, type="fit",mode="frac")
test_error2 <- mean((x_test2[,9] - test_predict2$fit)^2)
cat("Best tuning parameter s/lambda=", best_lambda2, ",\n\n",
    "Its test error=",test_error2, "\n\n", "Estimated Coeficients: \n" )
```

```
## Best tuning parameter s/lambda= 0.2929293 ,
##
##  Its test error= 0.5470205
##
##  Estimated Coeficients:
```

```
print(lasso.coef2)
```

```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
## 0.41923437 0.23260909 0.00000000 0.00000000 0.07230153 0.00000000 0.00000000
##      pgg45
## 0.00000000
```

Comment: We noticed that the test error under minimum CV rule is lower than that of one-standard rule. The test error under minimum CV rule is varies if not set.seed it, and sometimes it could be 0.4912, where there is no such for one-standard rule. Moreover, the one-standard rule keep 3 important variable and make others spares, where minimum CV rule keeps 5 variables out of the 8 variables.

8. Fit the adaptive LASSO regression for the prostate cancer data set with $\gamma = 1$.

(a) Select the parameter with 5 -fold CV, using the minimum CV rule. Report the best tuning parameter, the selected model, the estimated regression coefficients, and the TestErr.

```
# Load the required library
library(glmnet)
```

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

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

```
# Split the data into predictors (X) and the response variable (y)
X <- x_train2[,-9]
y <- y_train

# Define a sequence of lambda values for the adaptive LASSO
lambda_sequence <- seq(0, 10, length = 100)

# Initialize an empty vector to store cross-validation results
cv_results <- matrix(NA, nrow = length(lambda_sequence), ncol = 5)
```

```r
# Perform 5-fold cross-validation for different lambda values
for (i in 1:length(lambda_sequence)) {
  fit <- glmnet(X, y, alpha = 1, lambda = lambda_sequence[i], nfolds = 5)
  cv_results[i, ] <- c(lambda_sequence[i], fit$cvm, fit$index.min)
}

# Find the lambda that minimizes cross-validation error (Minimum CV Rule)
best_lambda_min <- cv_results[which.min(cv_results[, 2]), 1]

# Fit the model with the best lambda using the entire dataset
best_model_min <- glmnet(X, y, alpha = 1, lambda = best_lambda_min)

test_predict3 <- predict(best_model_min, newx = x_test2[,-9])
test_error3 <- mean((x_test2[,9] - test_predict3)^2)

# Display results for Minimum CV Rule
cat("==============Minimum CV Rule===============\n")
```

```
## ==============Minimum CV Rule===============
```

```r
cat("Best Lambda:", best_lambda_min, "\n")
```

```
## Best Lambda: 0
```

```r
cat("Selected Model (Non-zero coefficients):", sum(coef(best_model_min) != 0), "\n")
```

```
## Selected Model (Non-zero coefficients): 9
```

```r
cat("Estimated Regression Coefficients:\n")
```

```
## Estimated Regression Coefficients:
```

```r
print(coef(best_model_min))
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.427726218
## lcavol       0.576509089
## lweight      0.614190231
## age         -0.019008075
## lbph         0.144823829
## svi          0.737188080
## lcp         -0.206331874
## gleason     -0.029295274
## pgg45        0.009462698
```

```r
cat("\n Test error=",test_error3 )
```

```
##
##  Test error= 0.52125
```

(b) Select the parameter with 5-fold CV, using the one-standard rule. Report the best tuning parameter, the selected model, the estimated regression coefficients, and the TestErr.

```
# Find the lambda using the one-standard error rule
set.seed(8)
lambda_sequence <- seq(0, 1, length = 100)
cv_fit2 <- cv.lars(X, y, K = 5, index=lambda_sequence,
                   plot.it = FALSE, mode="fraction")
lasso.mcv2 <- which.min(cv_fit2$cv)
min_error_lambda <- cv_fit2$cv[lasso.mcv2]+cv_fit2$cv.error[lasso.mcv2]
best_lambda_one_std <- lambda_sequence[min(which(cv_fit2$cv<min_error_lambda))]

# Fit the model with the best lambda using the entire dataset
best_model_one_std <- glmnet(X, y, alpha = 1, lambda = best_lambda_one_std)
test_predict4 <- predict(best_model_one_std, newx = x_test2[,-9])
test_error4 <- mean((x_test2[,9] - test_predict4)^2)

# Display results for One-Standard Rule
cat("==============One-Standard Error Rule=============\n")
```

```
## ==============One-Standard Error Rule=============
```

```
cat("Best Lambda:", best_lambda_one_std, "\n")
```

```
## Best Lambda: 0.2727273
```

```
cat("Selected Model (Non-zero coefficients):", sum(coef(best_model_one_std) != 0), "\n")
```

```
## Selected Model (Non-zero coefficients): 4
```

```
cat("Estimated Regression Coefficients:\n")
```

```
## Estimated Regression Coefficients:
```

```
print(coef(best_model_one_std))
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept) 0.7882320
## lcavol      0.4316601
## lweight     0.2945723
## age         .
## lbph        .
## svi         0.1294451
## lcp         .
## gleason     .
## pgg45       .
```

```r
cat("\n Test Error (MSE):", test_error4)
```

```
##
##  Test Error (MSE): 0.5189873
```

(c) Compare the adaptive LASSO with the LASSO and forward selection, in terms of their variable selection results and prediction accuracy.

Comment: In regressing lpsa variable over others variables in the prostate cancer data for an important variables selection, we notice that, forward selection keeps just 2 variables out of 8 variables with 0.4924 error rate based on BIC consideration. Lasso keeps 5 variables out of 8 variables with 0.4569 error rate based on minimum CV consideration. Adaptive Lasso keeps 3 variables out of 8 variables with 0.5190 error rate based on one-standard rule. We observed that all the three models commonly selected those two variables (lcavol, and lweight). So in our case, forward selection method perform best due to its model simplicty and the error rate it produced.

=== Using given hint procedure with lars package ====

(a)

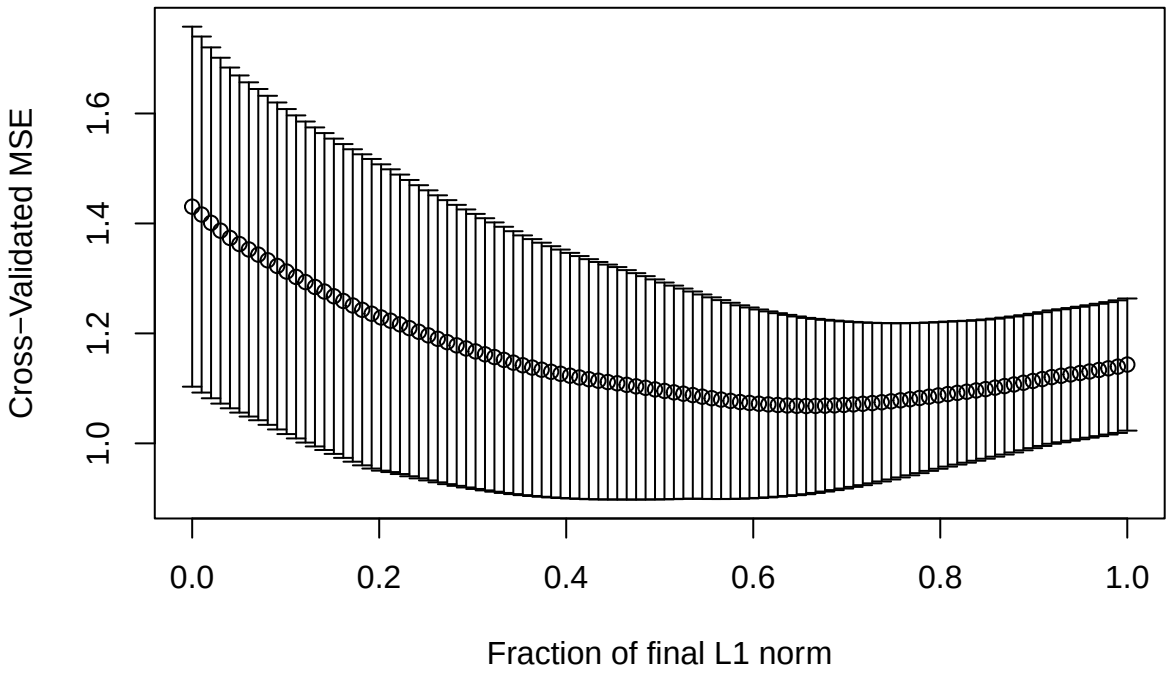
```r
library(lars)
set.seed(1300)

X <- x_train2[,-9]
X_new <- x_test2[,-9]
Y<- x_train2[,9]

# Step 1: Calculate weights
beta_ols <- coef(lm(Y ~ X))
w <- 1 / abs(beta_ols[-1])

# Step 2: Rescale X
#X_star <- scale(X, center = FALSE, scale = w)
X_star <- X/w
X_new_star <- X_new/w

#  Use cv.lars() and lars() to tune the penalty parameter  and
# fit adaptive lasso regression with Y and X_star.
lasso.s<-seq(0,1,length=100)
lasso_fit <- lars(x = X_star, y = Y, type = "lasso", normalize = FALSE, intercept = TRUE  )
cv_fit <- cv.lars(X_star, Y, K = 5,
                  index=lasso.s,mode="fraction")
```




```
lasso.mcv <- which.min(cv_fit$cv)

## minimum CV rule
best_lambda <- lasso.s[lasso.mcv]
beta_hat_star <- predict(lasso_fit, s=best_lambda, type="coef", mode="frac")
beta_hat<- c(beta_ols[1], beta_hat_star$coefficients / w)
test_predict <- predict(lasso_fit, newx = X_new_star, s=best_lambda, type="fit",mode="frac")

## Alternative way of predicting by adding a column with entries of 1
#column_1 <- rep(1, nrow(X_new_star))
#X_new_star <- cbind(column_1, X_new_star)
#predict <- X_new_star %*% beta_hat
#test_error <- mean((x_test2[,9] - predict)^2)

test_error <- mean((x_test2[,9] - test_predict$fit)^2)
cat("Best tuning parameter s/lambda=", best_lambda, ",\n\n",
    "Its test error=",test_error, "\n\n", "Estimated Coeficients: \n" )
```

```
## Best tuning parameter s/lambda= 0.6565657 ,
##
##  Its test error= 0.9744535
##
##  Estimated Coeficients:
```

```
print(beta_hat)
```

```
##  (Intercept)       lcavol      lweight          age         lbph          svi
## 0.4291701328 0.3191314044 0.0383725928 0.0001791118 0.0672707275 0.9038209396
##          lcp      gleason        pgg45
## 0.0554455696 0.0025548008 0.0001135186
```

(b)

```
## one-standard rule
bound<-cv_fit$cv[lasso.mcv]+cv_fit$cv.error[lasso.mcv]
best_lambda2<-lasso.s[min(which(cv_fit$cv<bound))]
beta_hat_star2 <- predict(lasso_fit, s=best_lambda2, mode="frac")
```

```
## Warning in predict.lars(lasso_fit, s = best_lambda2, mode = "frac"): Type=fit
## with no newx argument; type switched to coefficients
```

```
beta_hat2<- c(beta_ols[1], beta_hat_star2$coefficients / w)
test_predict2 <- predict(lasso_fit, newx = X_new_star, s=best_lambda2, type="fit",mode="frac")
test_error2 <- mean((x_test2[,9] - test_predict2$fit)^2)
cat("\n Best tuning parameter s/lambda=", best_lambda2, ",\n\n",
    "Its test error=",test_error2, "\n\n", "Estimated Coeficients: \n" )
```

```
##
##  Best tuning parameter s/lambda= 0.2121212 ,
##
##  Its test error= 0.8754055
##
##  Estimated Coeficients:
```

```
print(beta_hat2)
```

```
##  (Intercept)       lcavol      lweight          age         lbph          svi
## 0.4291701328 0.3134735772 0.0000000000 0.0001650153 0.0221029455 0.0000000000
##          lcp      gleason        pgg45
## 0.0216998186 0.0012528410 0.0001348603
```

Comment: Using given the hint with lars package, and the earlier library (glmnet) used above, we notice that the one-standard rule always slightly performs better of minimum CV rule, and also give a better sparsity.