STAT542, Homework 2

Shuhui Guo

Problem 1

a)

We can first sort out the training and testing datasets:

```r
bitcoin = bitcoin[,-5]
bitcoin$Date = as.numeric(bitcoin$Date) #replace the date variable with integers

X = bitcoin[,-2]
Y = bitcoin$btc_market_price
X_train = X[1:1460,]
X_test = X[1461:1588,]
Y_train = Y[1:1460]
Y_test = Y[1461:1588]
n_train = 1460
n_test = 128
```

Find the best subset selection of the dataset:

```r
x = as.matrix(X_train)
leaps = regsubsets(x, Y_train, nvmax = 22)
sumleaps = summary(leaps, matrix = T)
```

Since the results will be too long if we report the best models of all sizes ranging from 1 to 22, I will report the best models of the first five sizes below.

The variables selected for size1:

```r
names(which(sumleaps$which[1,] == 1)[-1])
```

```
## [1] "btc_market_cap"
```

The variables selected for size2:

```r
names(which(sumleaps$which[2,] == 1)[-1])
```

```
## [1] "btc_market_cap" "btc_difficulty"
```

The variables selected for size3:

```r
names(which(sumleaps$which[3,] == 1)[-1])
```

```
## [1] "btc_market_cap"     "btc_hash_rate"      "btc_miners_revenue"
```

The variables selected for size4:

```r
names(which(sumleaps$which[4,] == 1)[-1])
```

```
## [1] "btc_market_cap"          "btc_hash_rate"
## [3] "btc_miners_revenue"      "btc_cost_per_transaction"
```

The variables selected for size5:

```r
names(which(sumleaps$which[5,] == 1)[-1])
```

```
## [1] "btc_market_cap"          "btc_blocks_size"
## [3] "btc_hash_rate"           "btc_miners_revenue"
## [5] "btc_n_transactions_total"
```

b)

i) We can get the values of Cp, AIC, BIC of each size:

```r
msize = apply(sumleaps$which,1,sum)
n = dim(X_train)[1]
Cp = sumleaps$cp
BIC = sumleaps$bic
AIC = n*log(sumleaps$rss/n) + 2*msize
```

The variables of the best model selected by Cp criteria:

```r
min_Cp = which.min(Cp)
names(which(sumleaps$which[min_Cp,] == 1)[-1])
```

```
##  [1] "Date"                      "btc_total_bitcoins"
##  [3] "btc_market_cap"            "btc_blocks_size"
##  [5] "btc_avg_block_size"        "btc_n_orphaned_blocks"
##  [7] "btc_hash_rate"             "btc_difficulty"
##  [9] "btc_miners_revenue"        "btc_cost_per_transaction"
## [11] "btc_n_transactions_total"
```

The variables of the best model selected by AIC criteria:

```
min_AIC = which.min(AIC)
names(which(sumleaps$which[min_AIC,] == 1)[-1])
```

```
##  [1] "Date"                      "btc_total_bitcoins"
##  [3] "btc_market_cap"            "btc_blocks_size"
##  [5] "btc_avg_block_size"        "btc_n_orphaned_blocks"
##  [7] "btc_hash_rate"             "btc_difficulty"
##  [9] "btc_miners_revenue"        "btc_cost_per_transaction"
## [11] "btc_n_transactions_total"
```

The variables of the best model selected by BIC criteria:

```
min_BIC = which.min(BIC)
names(which(sumleaps$which[min_BIC,] == 1)[-1])
```

```
##  [1] "btc_total_bitcoins"       "btc_market_cap"
##  [3] "btc_blocks_size"          "btc_avg_block_size"
##  [5] "btc_n_orphaned_blocks"    "btc_hash_rate"
##  [7] "btc_difficulty"           "btc_miners_revenue"
##  [9] "btc_cost_per_transaction" "btc_n_transactions_total"
```

ii) Apply the fitted models to the testing dataset and we can get the prediction error.

The prediction error of the model selected by Cp criteria can be calculted by:

```
X_train_Cp = X_train[,which(sumleaps$which[min_Cp,] == 1)-1]
X_test_Cp = X_test[,which(sumleaps$which[min_Cp,] == 1)-1]
lm_train_Cp = lm(Y_train~., data = X_train_Cp)
Yhat_test_Cp = predict(lm_train_Cp, X_test_Cp)
error_Cp = sum((Yhat_test_Cp-Y_test)^2)/n_test
```

Similarly, we can get the prediction error of the models selected by AIC and BIC criteria.

Therefore,

The prediction error of the model selected by Cp criteria is 46876.07.

The prediction error of the model selected by AIC criteria is 46876.07.

The prediction error of the model selected by BIC criteria is 44235.49.

c)

i) Replace Y in a) and b) with $log(1+Y)$, and we can get the new training dataset:

```
Y_train = log(1+bitcoin$btc_market_price)[1:1460]
```

And we can get the values of Cp, AIC, BIC of each size:

```
leaps = regsubsets(x, Y_train, nvmax = 22)
sumleaps = summary(leaps, matrix = T)
msize=apply(sumleaps$which,1,sum)
Cp = sumleaps$cp
BIC = sumleaps$bic
AIC = n*log(sumleaps$rss/n) + 2*msize
```

The variables of the best model selected by Cp criteria:

```
min_Cp = which.min(Cp)
names(which(sumleaps$which[min_Cp,] == 1)[-1])
```

```
##  [1] "Date"
##  [2] "btc_total_bitcoins"
##  [3] "btc_market_cap"
##  [4] "btc_blocks_size"
##  [5] "btc_avg_block_size"
##  [6] "btc_median_confirmation_time"
##  [7] "btc_difficulty"
##  [8] "btc_transaction_fees"
##  [9] "btc_cost_per_transaction"
## [10] "btc_n_unique_addresses"
## [11] "btc_n_transactions_total"
## [12] "btc_n_transactions_excluding_chains_longer_than_100"
## [13] "btc_estimated_transaction_volume"
## [14] "btc_estimated_transaction_volume_usd"
```

The variables of the best model selected by AIC criteria:

```
min_AIC = which.min(AIC)
names(which(sumleaps$which[min_AIC,] == 1)[-1])
```

```
##  [1] "Date"
##  [2] "btc_total_bitcoins"
##  [3] "btc_market_cap"
##  [4] "btc_blocks_size"
##  [5] "btc_avg_block_size"
##  [6] "btc_median_confirmation_time"
##  [7] "btc_difficulty"
##  [8] "btc_transaction_fees"
##  [9] "btc_cost_per_transaction"
## [10] "btc_n_unique_addresses"
## [11] "btc_n_transactions_total"
## [12] "btc_n_transactions_excluding_chains_longer_than_100"
## [13] "btc_estimated_transaction_volume"
## [14] "btc_estimated_transaction_volume_usd"
```

The variables of the best model selected by BIC criteria:

```
min_BIC = which.min(BIC)
names(which(sumleaps$which[min_BIC,] == 1)[-1])
```

```
##  [1] "Date"
##  [2] "btc_total_bitcoins"
##  [3] "btc_market_cap"
##  [4] "btc_blocks_size"
##  [5] "btc_avg_block_size"
##  [6] "btc_median_confirmation_time"
##  [7] "btc_difficulty"
##  [8] "btc_transaction_fees"
##  [9] "btc_cost_per_transaction"
## [10] "btc_n_unique_addresses"
## [11] "btc_n_transactions_total"
## [12] "btc_n_transactions_excluding_chains_longer_than_100"
## [13] "btc_estimated_transaction_volume_usd"
```

ii)

The prediction error of the model selected by Cp criteria can be calculted by:

```
X_train_Cp = X_train[,which(sumleaps$which[min_Cp,] == 1)-1]
X_test_Cp = X_test[,which(sumleaps$which[min_Cp,] == 1)-1]
lm_train_Cp = lm(Y_train~., data = X_train_Cp)
Yhat_test_Cp = predict(lm_train_Cp, X_test_Cp)
Yorihat_test_Cp = exp(Yhat_test_Cp)-1
error_Cp = sum((Yorihat_test_Cp-Y_test)^2)/n_test
```

Similarly, we can get the prediction error of the models selected by AIC and BIC criteria.

Therefore,

The prediction error of the model selected by Cp criteria is 4426691.45.

The prediction error of the model selected by AIC criteria is 4426691.45.

The prediction error of the model selected by BIC criteria is 4426655.95.

Problem 2

Part I

Firstly, generate X and Y from the code in HW2.r file given by the professor. Then start the Lasso:

```
x_center = colMeans(X)
x_scale = apply(X, 2, sd)
X2 = scale(X)

bhat = rep(0, ncol(X2)) # initialize it
ymean = mean(y)
y2 = y - ymean

# prepare the soft thresholding function
soft_th <- function(b, pen)
{
  if(abs(b)>pen) s = sign(b)*(abs(b)-pen)  else s=0
  return(s)
}

# initiate lambda
lambda = exp(seq(log(max(abs(cov(X2, y2)))), log(0.001), length.out = 100))

# prepare the Lasso function
LassoFit <- function(myX, myY, mybeta, mylambda, tol = 1e-10, maxitr = 500)
{
  # initia a matrix to record the objective function value
  N = dim(myX)[1]
  f = rep(0, maxitr)

  for (k in 1:maxitr)
  {
    # compute residual
```

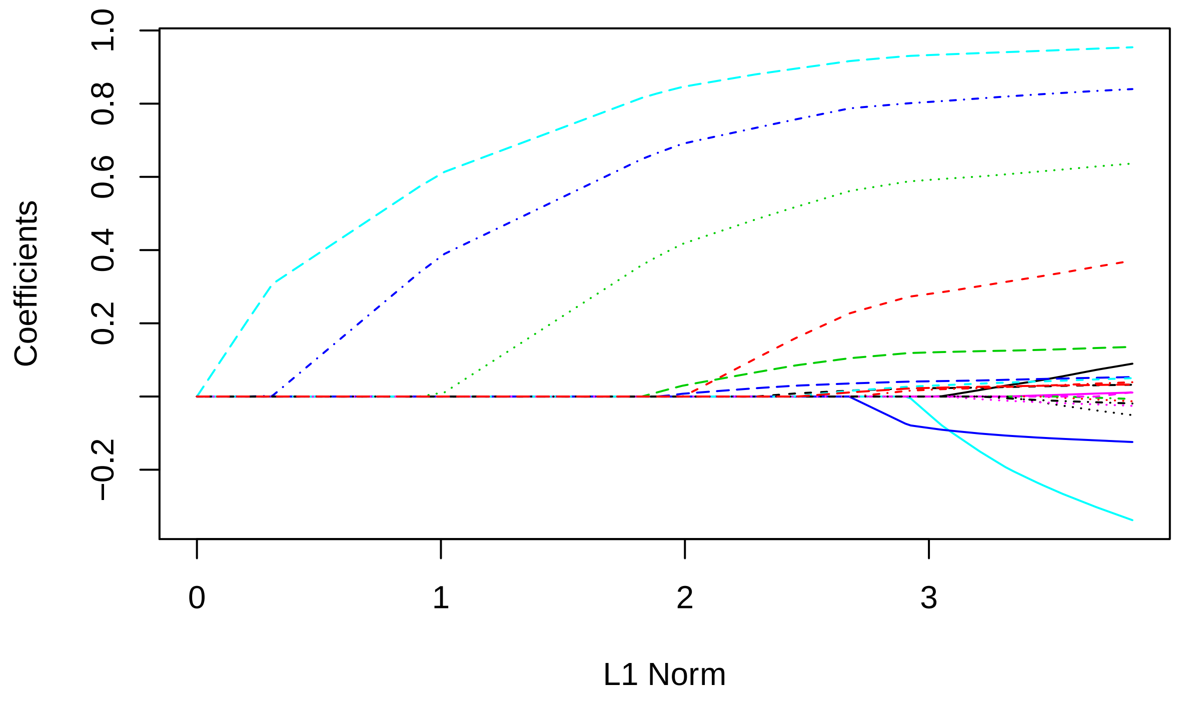
```r
    r = myY - myX %*% mybeta

    # I need to record the residual sum of squares
    f[k] = mean(r*r)

    for (j in 1:ncol(myX))
    {
      # add the effect of jth variable back to r
      r_new = myY - myX %*% mybeta
      beta_new = sum(r_new*myX[,j])/N + mybeta[j]

      # apply the soft_th function to the ols estimate of the jth variable
      mybeta[j] <- soft_th(beta_new, mylambda)
    }

    if (k > 10)
    {
      # stop rule
      if (sum(abs(f[(k-9):k] - mean(f[(k-9):k]))) < tol) break;
    }
  }
  return (mybeta)
}

# test my function
LassoFit(X2, y2, mybeta = rep(0, ncol(X2)), mylambda = lambda[10], tol = 1e-7,
         maxitr = 500)
```

```
##  [1] 0.000000000 0.000000000 0.006769549 0.310409395 0.619233311
##  [6] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [11] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [16] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

```r
# initiate a matrix that records the fitted beta for each lambda value
beta_all = matrix(NA, ncol(X), length(lambda))

# this vecter stores the intercept of each lambda value
beta0_all = rep(NA, length(lambda))

# interate all lambda values
bhat = rep(0, ncol(X2)) # initialize it

for (i in 1:length(lambda)) # loop from the largest lambda value
{
  bhat = LassoFit(X2, y2, bhat, lambda[i])

  # save the correctly scaled beta into the beta matrix
  beta_all[, i] = bhat/x_scale

  # recalculte the intercept
  beta0_all[i] = ymean - sum((x_center/x_scale)*bhat)
}

# coefficient matrix
coef_new = rbind("intercept" = beta0_all, beta_all)
```

The plot of the coefficient trace is:

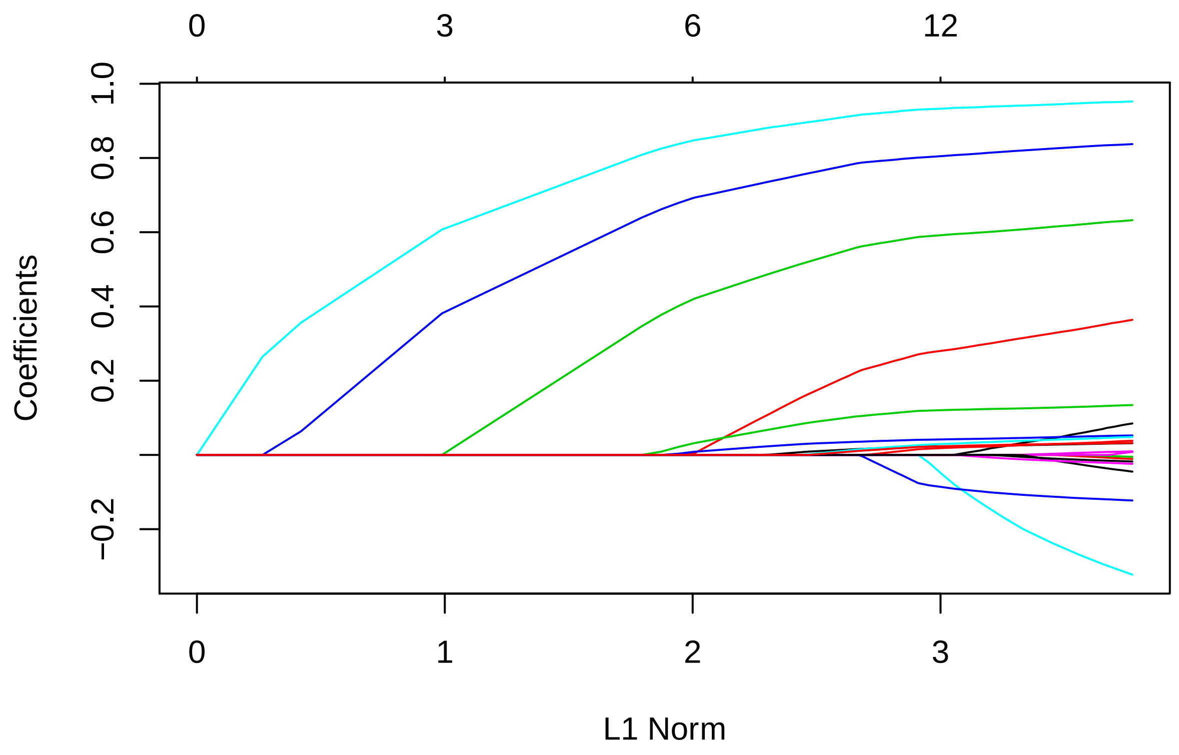
```
matplot(colSums(abs(beta_all)), t(beta_all), type="l",
        xlab = "L1 Norm", ylab = "Coefficients")
```



The following part is how I demonstrate that my code is correct:

Firstly, compare with the plot in glmnet package:

```r
plot(glmnet(X, y))
```



We can see this plot is identical to my previous plot.

Then, set my lambda to their lambda value and rerun my code.

```r
lambda = glmnet(X, y)$lambda
```

Calculate the distance:

```r
max(abs(beta_all - glmnet(X, y)$beta))
```

```
## [1] 0.002058255
```

```r
max(abs(beta0_all - glmnet(X, y)$a0))
```

```
## [1] 0.001375829
```

We can see the two distances are less than 0.01, which indicates that my code is correct.

PartII

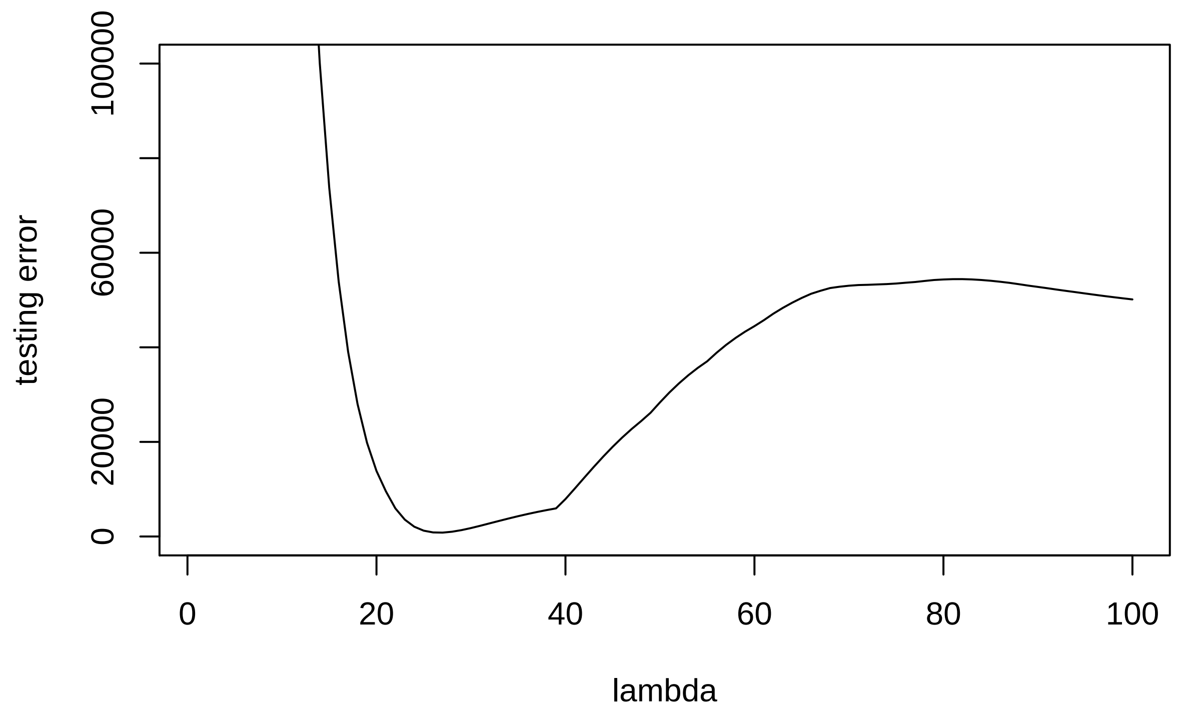
Fit the Lasso model to the training dataset of the bitcoin dataset:

```
bitcoin = read.table("C:/Users/guoshuhui/Desktop/542/homework/hw2/bitcoin_dataset.csv",
bitcoin = bitcoin[,-5]
bitcoin$Date = as.numeric(bitcoin$Date) #replace the date variable with integers

X = bitcoin[,-2]
Y = bitcoin$btc_market_price
X_train = X[1:1460,]
X_test = X[1461:1588,]
Y_train = Y[1:1460]
Y_test = Y[1461:1588]
n_train = 1460
n_test = 128
X_train_center = colMeans(X_train)
X_train_scale = apply(X_train, 2, sd)
X2_train = scale(X_train)

Ymean_train = mean(Y_train)
Y2_train = Y_train - Ymean_train

lambda = exp(seq(log(max(abs(cov(X2_train, Y2_train)))), log(0.001),
                 length.out = 100))

beta_all = matrix(NA, ncol(X_train), length(lambda))
beta0_all = rep(NA, length(lambda))
bhat = rep(0, ncol(X2_train))

for (i in 1:length(lambda))
{
  bhat = LassoFit(X2_train, Y2_train, bhat, lambda[i])
  beta_all[, i] = bhat/X_train_scale
  beta0_all[i] = Ymean_train - sum((X_train_center/X_train_scale)*bhat)
}

coef = rbind("intercept" = beta0_all, beta_all)
```

Then we can report the testing errors on the testing dataset in a labeled graph:

```
X_test_new = cbind(rep(1,n_test), X_test)
coef = as.matrix(coef)
X_test_new = as.matrix(X_test_new)
```

```r
Yhat_test = X_test_new %*% coef

error_test <- c()
for (i in 1:100) {
  error_test[i] = mean((Y_test-Yhat_test[,i])^2)
}

plot(error_test, type = "l", ylim=c(0,100000),
     xlab = "lambda", ylab = "testing error")
```



```r
lambda_best = which.min(error_test)
error_test_min = error_test[lambda_best]
```

And we can get the minimum testing error is 817.11.

The selected variables and the corresponding coefficients are:

```r
colnames(X_test[which(coef[,lambda_best]!= 0)-1])
```

```
## [1] "btc_market_cap"            "btc_miners_revenue"
## [3] "btc_cost_per_transaction"
```

```
coef[which(coef[,lambda_best]!= 0),lambda_best]
```

```
##    intercept
## 5.633080e+00 5.464787e-08 4.250181e-05 1.082220e+00
```

Therefore, the best model is

$$\begin{aligned}\hat{Y} =& 5.63 + 5.46 \times 10^{-8} \times btc\_market\_cap + 4.25 \times 10^{-5} \times btc\_miners\_revenue \\ &+ 1.08 \times btc\_cost\_per\_transaction\end{aligned}$$