STAT542 Statistical Learning Homework 2

Huamin Zhang (huaminz2@illinois.edu)

Oct 6, 2017

Question 1

a) [15 points]

Answer:

```
library(leaps)
# Read data set
data = read.csv('bitcoin_dataset.csv')

# Get the index of train and test data
index1 = which(data[,1] == '2017-01-01 00:00:00')
index2 = which(data[,1] == '2017-09-12 00:00:00')

# Here we treat the variable Date as a continuous variable, say starting
# with value 0 at the first instances, then increase by 1 for each two days.
data$Date = 1:dim(data)[1]

# Split the data set into train and test data
# We remove Date information and treat the 2rd column 'btc_market_price'
# as the outcome variable
train_data = data[1:index1-1,]
test_data = data[index1:index2,]
train_data_x = train_data[,-2]
train_data_y = train_data[,2]
test_data_x = test_data[,-2]
test_data_y = test_data[,2]

# Ignore the variable btc_trade_volume because it contains missing values
train_data_x = subset(train_data_x,select = -btc_trade_volume)
test_data_x = subset(test_data_x,select = -btc_trade_volume)

# Performs an exhaustive search over models, and gives back the best model
# (with low RSS) of each size.
RSSleaps=regsubsets(as.matrix(train_data_x),train_data_y,
                    nvmax = length(train_data_x))
best_subset= summary(RSSleaps, matrix=T)
```

You can find the best model of each size (1-22) from the output from `summary(RSSleaps, matrix=T)`. However, since the output maybe too wide (many columns) to fit a page, we will only report present a part of it.The following is the best model of `size = 10` which means the best model with using only ten features(except Intercept).

```
#best_subset
coef(RSSleaps,10)
```

```
##              (Intercept)        btc_total_bitcoins             btc_market_cap
##             9.204666e-01              -8.102118e-07               6.938252e-08
##           btc_blocks_size         btc_avg_block_size     btc_n_orphaned_blocks
##            -7.289203e-03               2.176334e+01              -1.507009e+00
##             btc_hash_rate             btc_difficulty         btc_miners_revenue
##            -2.436673e-05              -2.104494e-10               1.859082e-05
## btc_cost_per_transaction btc_n_transactions_total
##             5.870465e-01               3.782071e-06
```

You can also get the the result of each size from the plot with their R square

```
plot(RSSleaps,scale='r2')
# The models are ordered by the specified model selection statistic. This plot is
# particularly useful when there are more than ten or so models and the simple
# table produced by summary.regsubsets is too big to read. Here we didn't plot here
# due the limitation of report page.
```

b) [15 points]

Answer:

```
lmfit=lm(train_data_y~as.matrix(train_data_x))
msize=apply(best_subset$which,1,sum)
n=dim(train_data_x)[1]
p=dim(train_data_x)[2]

# Calculate the Cp, AIC, BIC of the best model of each size
Cp = best_subset$rss/(summary(lmfit)$sigma^2) + 2*msize - n
AIC = n*log(best_subset$rss/n) + 2*msize
BIC = n*log(best_subset$rss/n) + msize*log(n)

# Select the best model
result = data.frame(which.min(Cp),which.min(AIC),which.min(BIC),
                    row.names = "The best model")
colnames(result) = c("Cp","AIC","BIC")
result
```

```
##                Cp AIC BIC
## The best model 11  11  10
```

Thus, the best model using C_P and AIC criteria is that with `size = 11`. And the best model using BIC criteria is that with `size = 10` The C_p, AIC and BIC of these two models are showed as below.

```
cbind(Cp,AIC,BIC)[c(10,11),]
```

```
##           Cp      AIC      BIC
## 10 13.164159 5733.904 5792.052
## 11  9.591456 5730.287 5793.721
```

The features used in the best model with `size = 10` and are showed as below.

```
names(train_data_x)[best_subset$which[10,-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"
```

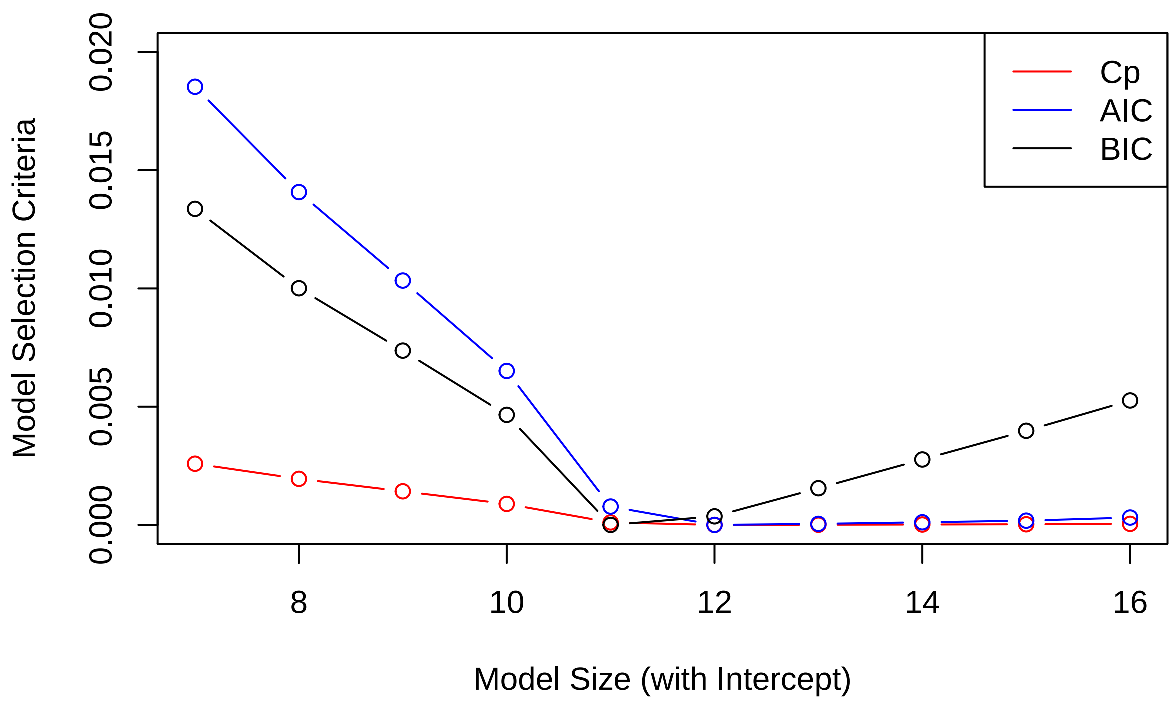
The features used in the best model with `size = 11` and are showed as below.

```
names(train_data_x)[best_subset$which[11,-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"
```

Here we resacle C_p, AIC, BIC to (0,1) and made plots of these criterion of size from 6 to 15 (including Intercept).

```
# Rescale Cp, AIC, BIC to (0,1).
inrange <- function(x) { (x - min(x)) / (max(x) - min(x)) }
Cp = best_subset$cp; Cp = inrange(Cp);
BIC = best_subset$bic; BIC = inrange(BIC);
AIC = n*log(best_subset$rss/n) + 2*msize; AIC = inrange(AIC);
# Since we know when size = 10,11 we can get the minimun, here we
# plot the Model Selection Criteria(Cp, AIC, BIC) of size = 6-15
# zoom in
id=6:15;
plot(range(msize[id]), c(0, 0.02), type="n", xlab="Model Size (with Intercept)",
     ylab="Model Selection Criteria")
points(msize[id], Cp[id], col="red", type="b")
points(msize[id], AIC[id], col="blue", type="b")
points(msize[id], BIC[id], col="black", type="b")
legend("topright", lty=rep(1,3), col=c("red", "blue", "black"),
       legend=c("Cp", "AIC", "BIC"))
```



Then we applied the fitted models to the testing dataset and report the prediction error of $n_{test}^{-1} \sum_{i \in test} (\hat{Y}_i - Y_i)^2$

```
# Apply the fitted models on the test dataset
model_10 = lm(btc_market_price~.,data=train_data[,c('btc_market_price',
              names(which(best_subset$which[10,-1] == TRUE)))])
predict_10_y = predict(model_10,newdata = test_data)

model_11 = lm(btc_market_price~.,data=train_data[,c('btc_market_price',
              names(which(best_subset$which[11,-1] == TRUE)))])
predict_11_y = predict(model_11,newdata = test_data)

# Calculate the prediction error
mse_10 = mean((predict_10_y - test_data_y)^2)
mse_11 = mean((predict_11_y - test_data_y)^2)
# The MSE of the best model with size = 10 (BIC)
mse_10
```

```
## [1] 44235.49
```

```
# The MSE of the best model with size = 11 (Cp, AIC)
mse_11
```

```
## [1] 46876.07
```

So, the prediction error of the best model with `size = 10` which is selected by BIC criteria is 44235.49. And the prediction error of the best model with `size = 11` which

is selected by C_p and AIC criteria is 46876.07.

c) [15 points]

Answer:

```
# redo a-b
# Performs an exhaustive search over models, and gives back the best model
# (with low RSS) of each size.
RSSleaps=regsubsets(as.matrix(train_data_x),log(train_data_y+1),
                    nvmax = length(train_data_x))
best_subset= summary(RSSleaps, matrix=T)
```

The best model of each size (1-22) can be found from the output from `summary(RSSleaps, matrix=T)` Here we will only report present a part of it.The following is the best model of `size = 10` which means the best model with using only ten features(except Intercept).

```
#best_subset
coef(RSSleaps,10)
```

```
##                 (Intercept)                        Date
##               -9.552736e-02               -5.264735e-03
##          btc_total_bitcoins              btc_market_cap
##                5.145858e-07               -8.651644e-11
##             btc_blocks_size          btc_avg_block_size
##               -7.743416e-04                1.628053e+00
## btc_median_confirmation_time              btc_difficulty
##               -2.552420e-02               -1.296557e-11
##     btc_cost_per_transaction      btc_n_unique_addresses
##                4.430324e-02                3.466938e-06
##     btc_n_transactions_total
##                4.536846e-07
```

```
# You can also get the the result of each size from the plot with their R square
# plot(RSSleaps,scale='r2')
```

Then we use C_p, AIC and BIC criteria to select the best model.

```
# Calculate the Cp, AIC, BIC of the best model of each size
lmfit=lm(log(train_data_y+1)~as.matrix(train_data_x))
msize=apply(best_subset$which,1,sum)
n=dim(train_data_x)[1]
p=dim(train_data_x)[2]
Cp = best_subset$rss/(summary(lmfit)$sigma^2) + 2*msize - n
AIC = n*log(best_subset$rss/n) + 2*msize
BIC = n*log(best_subset$rss/n) + msize*log(n)

# Select the best model
```

```r
result = data.frame(which.min(Cp),which.min(AIC),which.min(BIC),
                    row.names = "The best model")
colnames(result) = c("Cp","AIC","BIC")
result
```

```
##                Cp AIC BIC
## The best model 14  14  13
```

Thus, the best model using C_P and AIC criteria is that with `size = 14`. And the best model using BIC criteria is that with `size = 13` The C_p, AIC and BIC and the features of these two models are showed as below.

```r
cbind(Cp,AIC,BIC)[c(13,14),]
```

```
##          Cp       AIC       BIC
## 13 17.23428 -3362.928 -3288.921
## 14 15.22976 -3364.968 -3285.675
```

```r
# Selected features in the best model
names(train_data_x)[best_subset$which[13,-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"
```

```r
names(train_data_x)[best_subset$which[14,-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"
```

Then we applied the fitted models to the testing dataset and report the prediction error.

```r
# Apply the fitted models on the test dataset
model_log_13 = lm(log(btc_market_price+1)~.,data=train_data[,c('btc_market_price',
              names(which(best_subset$which[13,-1] == TRUE)))])
predict_log_13_y = predict(model_log_13,newdata = test_data)

model_log_14 = lm(log(btc_market_price+1)~.,data=train_data[,c('btc_market_price',
              names(which(best_subset$which[14,-1] == TRUE)))])
predict_log_14_y = predict(model_log_14,newdata = test_data)

# Calculate the prediction error
e = 2.718281828459
mse_log_13 = mean(((e^(predict_log_13_y)-1) - test_data_y)^2)
mse_log_14 = mean(((e^(predict_log_14_y)-1) - test_data_y)^2)
# The MSE of the best model with size = 13 (BIC)
mse_log_13
```

```
## [1] 4426656
```

```r
# The MSE of the best model with size = 14 (Cp, AIC)
mse_log_14
```

```
## [1] 4426691
```

So, the prediction error of the best model with `size = 13` which is selected by BIC criteria is 4426656. And the prediction error of the best model with `size = 14` which is selected by C_p and AIC criteria is 4426691.

Question 2

Part I [35 points]

Answer:

Firstly, we will not penalize the intercept term β_0 which means that we will center both your X and Y first and perform the algorithm in the question. Then let's derive the soft thresholding function/update rule of Lasoo using coordinate descent algorithm.

In the question, assume we have a scaled and centered dataset X with N instances, each of which consists of p features, and a centered outcome Y. We will use the objective function

$$f(\beta) = \frac{1}{2n}||Y - X\beta||_2^2 + \lambda \sum_{j=1}^{p} |\beta_j| = \frac{1}{2n}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$$

Consider the update rule of β_j, we take the derivative with respect to β_j of the objective function. Here we can split the objective function into two part: 1) Loss function term $g(\beta) = \frac{1}{2n}\sum_{i=0}^{N}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2$ 2) Regularization function term $h(\beta) = \lambda\sum_{j=1}^{p}|\beta_j|$

Thus, the derivative with respect to β_j of the Loss function term is

$$\begin{aligned}
\frac{\partial g(\beta)}{\partial \beta_j} &= -\frac{1}{n}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})x_{ij} \\
&= -\frac{1}{n}\sum_{i=1}^{N}(y_i - \sum_{k\neq j}^{p}\beta_k x_{ik} - \beta_j x_{ij})x_{ij} \\
&= -\frac{1}{n}\sum_{i=1}^{N}(y_i - \sum_{k\neq j}^{p}\beta_k x_{ik})x_{ij} + \frac{1}{n}\beta_j\sum_{i=1}^{N}x_{ij}^2 \\
&= -\frac{1}{n}p_j + \frac{1}{n}\beta_j z_j
\end{aligned}$$

where

$$\begin{aligned}
p_j &= \sum_{i=1}^{N}(y_i - \sum_{k\neq j}^{p}\beta_k x_{ik})x_{ij} \\
&= \sum_{i=1}^{N}(y_i - \sum_{k=1}^{p}\beta_k x_{ik} + \beta_j x_{ij})x_{ij} \\
&= \sum_{i=1}^{N}(r_i + \beta_j x_{ij})x_{ij} \\
r_i &= y_i - \sum_{k=1}^{p}\beta_k x_{ik} \\
z_j &= \sum_{i=1}^{N}x_{ij}^2
\end{aligned}$$

For the regularization term, we can take the derivative with respect to β_j using subgradient method. So the subdifferential is

$$\partial_{\beta_j}\lambda\sum_{j=1}^{p}|\beta_j| = \partial_{\beta_j}\lambda|\beta_j| = \begin{cases} \lambda & \text{if } \beta_j > 0 \\ [-\lambda, \lambda] & \text{if } \beta_j = 0 \\ -\lambda & \text{if } \beta_j < 0 \end{cases}$$

So the have the the derivative with respect to β_j of the objective function.

$$\frac{\partial f(\beta)}{\partial \beta_j} = \frac{\partial g(\beta)}{\partial \beta_j} + \frac{\partial h(\beta)}{\partial \beta_j} = \begin{cases} -\frac{1}{n}p_j + \frac{1}{n}\beta_j z_j + \lambda & \text{if } \beta_j > 0 \\ [-\frac{1}{n}p_j + \frac{1}{n}\beta_j z_j - \lambda, -\frac{1}{n}p_j + \frac{1}{n}\beta_j z_j + \lambda] & \text{if } \beta_j = 0 \\ -\frac{1}{n}p_j + \frac{1}{n}\beta_j z_j - \lambda & \text{if } \beta_j < 0 \end{cases}$$

Let the derivative to be zero $\partial_{\beta J} f(\beta) = 0$ and solve for $\hat{\beta}_j$. We can get

$$\begin{aligned}
\hat{\beta}_j &= \begin{cases} \frac{p_j - n\lambda}{z_j} & \text{if } p_j > n\lambda \\ 0 & \text{if } p_j \in [-n\lambda, n\lambda] \\ \frac{p_j + n\lambda}{z_j} & \text{if } p_j < n\lambda \end{cases} \\
&= \frac{1}{z_j} sign(p_j)(|p_j| - n\lambda)_+
\end{aligned}$$

According to the soft thresholding function. we can write the Lasso fitting function using coordinate descent algorithm as below. The detail about the function can be found in the comments.

```
# Function LassoFit
# Usage:
# LassoFit(X, y, mybeta = rep(0, ncol(X)), lambda, tol = 1e-10, maxitr = 500)

# Arguments
# myX: input matrix, each row is an observation vector (don't need to be
#      scaled and centered)
# myY: response variable (don't need to be centered)
# mybeta: the initialization of beta
# mylambda: tuning parameter (penalty level), which controls the amount of
#      shrinkage. Usually, mylambda >= 0.
# tol: Convergence threshold for coordinate descent. Each inner coordinate-
#      descent loop continues until the change in the objective function value
#      after any update is less than tol (or run maxitr iterations). Defaults
#      value is 1E-10.
# maxitr: The maximum number of iteration in coordinate-descent loop. Defaults
#      value is 500.

# Value(Output)
# beta_0: The intercept term in our fitted model of the original scale of X.
# beta: The coefficient in our fitted model of the original scale of X.

LassoFit <- function(myX, myY, mybeta, mylambda, tol = 1e-10, maxitr = 500){
  # First we scale and center X, and record them. Also center y and record it.
  # dont scale it. Now since both y and X are centered at 0, we don't need to worry
  # about the intercept anymore.
  x_center = colMeans(myX)
  x_scale = apply(myX, 2, sd)
  X = scale(myX)
  y_mean = mean(myY)
  Y = myY - y_mean
  # Calculate the number of instances
  n = nrow(X)
  # Calculate zj = \sum (x_ij)^2 in the formula
  z = apply(X,2,function(x) sum(x^2))
  # Initia a matrix to record the objective function value
  f = rep(0, maxitr)
  # Start the iteration unless meet the stopping rule or run maxitr iterations
  for (k in 1:maxitr){
    # Compute the residual using current beta
    r = Y - X %*% mybeta
    # Record the objective function value
    f[k] = sum(r*r) / (2*n) + mylambda * sum(abs(mybeta))
    # Calculate the stopping rule
```

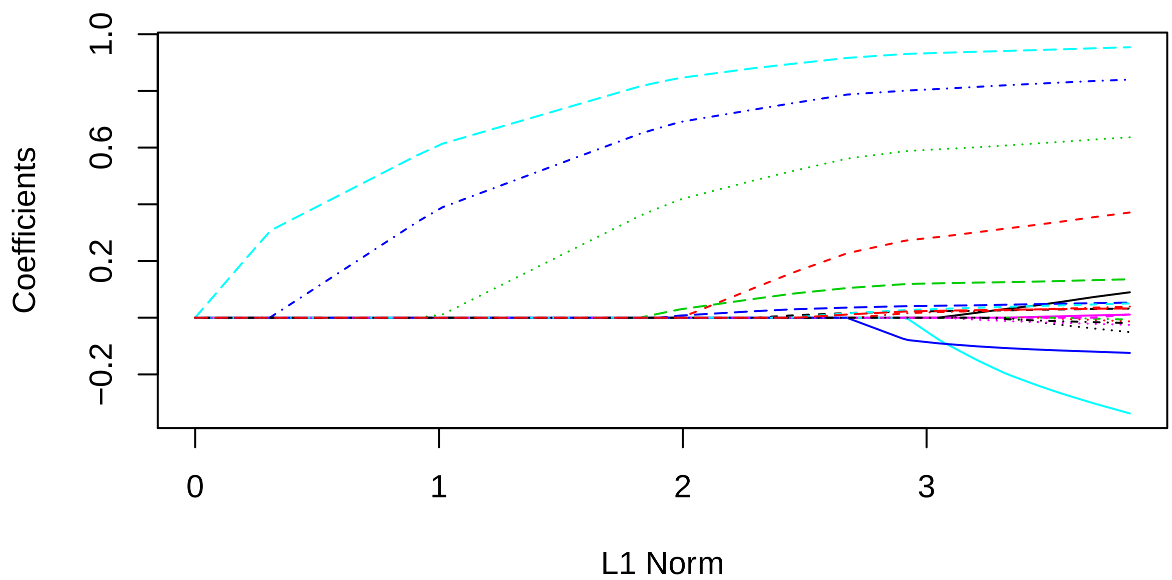
```r
    if (k > 10){
      if (abs(f[k] - f[k-1]) < tol) break;
    }
    # Use coordinate descent to update beta
    for (j in 1:ncol(X)){
      # Add the effect of jth variable back to r
        r = r + X[,j] * mybeta[j]
      # Calculate the pj in the formula
        p = sum(X[,j] * r)
      # Using the soft thresholding function
        mybeta[j] = sign(p)*ifelse(abs(p)>n*mylambda,abs(p)-n*mylambda,0) / z[j]
      # Remove the new effect of jth varaible out of r
        r = r - X[,j] * mybeta[j]
      # You can also write as the follow structure
      # Caluculate the pj in the formula
      # p = sum(X[,j] * (Y - X[,-j] %*% mybeta[-j]))
      # Using the update rule / soft thresholding function
      # mybeta[j] = sign(p) * ifelse(abs(p)>n*mylambda,abs(p)-n*mylambda,0) / z[j]
    }
  }
  # Scale the beta back
  mybeta = mybeta / x_scale
  # Recalculte the intercept term in the original, uncentered and unscaled X
  beta_0 = mean(myY - myX %*% mybeta)
  # Return the beta and intercept
  return(list("beta" = mybeta,"beta_0" = beta_0))
}
```

Here, to demonstrate that the code is correct, we compare the results to glmnet. And compare the fitted beta values. First, we generate a matrix X and outcome Y.

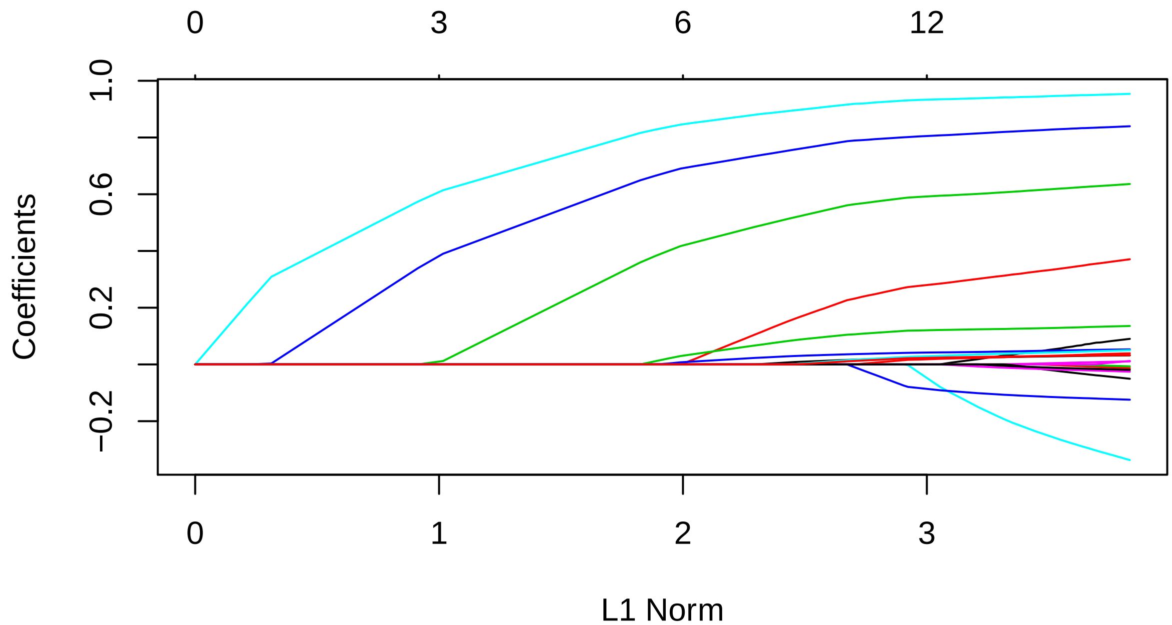
```r
library(MASS)
library(glmnet)
set.seed(1)
N = 400
P = 20
Beta = c(1:5/5, rep(0, P-5))
Beta0 = 0.5
# Genrate X
V = matrix(0.5, P, P)
diag(V) = 1
X = as.matrix(mvrnorm(N, mu = 3*runif(P)-1, Sigma = V))
# Create artifical scale of X
X = sweep(X, 2, 1:10/5, "*")
# Genrate Y
y = Beta0 + X %*% Beta + rnorm(N)
```

Now we want to compare the result from our algorithm and glmnet

```r
# Set the lambda sequence
lambda = exp(seq(log(max(abs(cov(scale(X), (y-mean(y)))))), log(0.001),
                 length.out = 100))
# 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))
# Initialize beta
bhat = rep(0, ncol(X))
# Loop from the largest lambda value
for (i in 1:length(lambda)){
  lasso_beta = LassoFit(X, y, bhat, lambda[i])
  bhat = lasso_beta$beta
  beta_all[, i] = bhat
  beta0_all[i] = lasso_beta$beta_0
}
# Here we make a plot of the L1 Norm versus Coefficients
matplot(colSums(abs(beta_all)), t(beta_all), type="l",xlab = "L1 Norm",
        ylab = "Coefficients")
```




```r
# We can also make a plot from the result of glmnet
plot(glmnet(X, y, lambda = lambda, alpha = 1))
```



We can find the two plots from our function and glmnet result are almost the same. In the other hand, we use the lambda sequence which is used in the glmnet result and rerun our algotithms. Then we compare the coefficients and intercept from our algorithm and glmnet to demonstrate that the code is correct.

```
# Set our lambda to the lambda value from glmnet and rerun your algorithm
lambda = glmnet(X, y)$lambda
beta_all = matrix(NA, ncol(X), length(lambda))
beta0_all = rep(NA, length(lambda))
# Initialize beta
bhat = rep(0, ncol(X))

# loop from the largest lambda value
for (i in 1:length(lambda)){
  lasso_beta = LassoFit(X, y, bhat, lambda[i])
  bhat = lasso_beta$beta
  beta_all[, i] = lasso_beta$beta
  beta0_all[i] = lasso_beta$beta_0
}
# Then this distance should be pretty small which is no more than 0.01
glmnet_result = glmnet(X, y, alpha = 1)
max(abs(beta_all - glmnet_result$beta))
```

```
## [1] 0.002095227
```

```
max(abs(beta0_all - glmnet_result$a0))
```

```
## [1] 0.001404763
```

According to the result, we can say that the code is correct.

Part II [30 points]

Answer:

Here we use the lambda sequence from part of the result of glmnet, and use mean squared error as the test error to select the best model.

```r
# Set our lambda to the lambda value from glmnet
result = glmnet(as.matrix(train_data_x),train_data_y,alpha = 1, nlambda = 35)
lambda = result$lambda[9:22]
# Initiate a matrix that records the fitted beta for each lambda value
beta_all = matrix(NA, ncol(train_data_x), length(lambda))
# This vecter stores the intercept of each lambda value
beta0_all = rep(NA, length(lambda))
# This vecter stores the test error of each lambda value
MSE = rep(NA, length(lambda))
# Initialize beta
bhat = rep(0, ncol(train_data_x))

# Loop from the largest lambda value
for (i in 1:length(lambda)){
  # Train the Lasso model
  lasso_beta = LassoFit(as.matrix(train_data_x), train_data_y, bhat, lambda[i])
  # Store the coefficients and intercepts
  beta_all[, i] = lasso_beta$beta
  beta0_all[i] = lasso_beta$beta_0
  # Apply the fitted model on the test dataset
  test_y_hat = as.matrix(test_data_x) %*% lasso_beta$beta + lasso_beta$beta_0
  # Calculate the test error
  MSE[i] = mean((test_y_hat-test_data_y)^2)
}
```

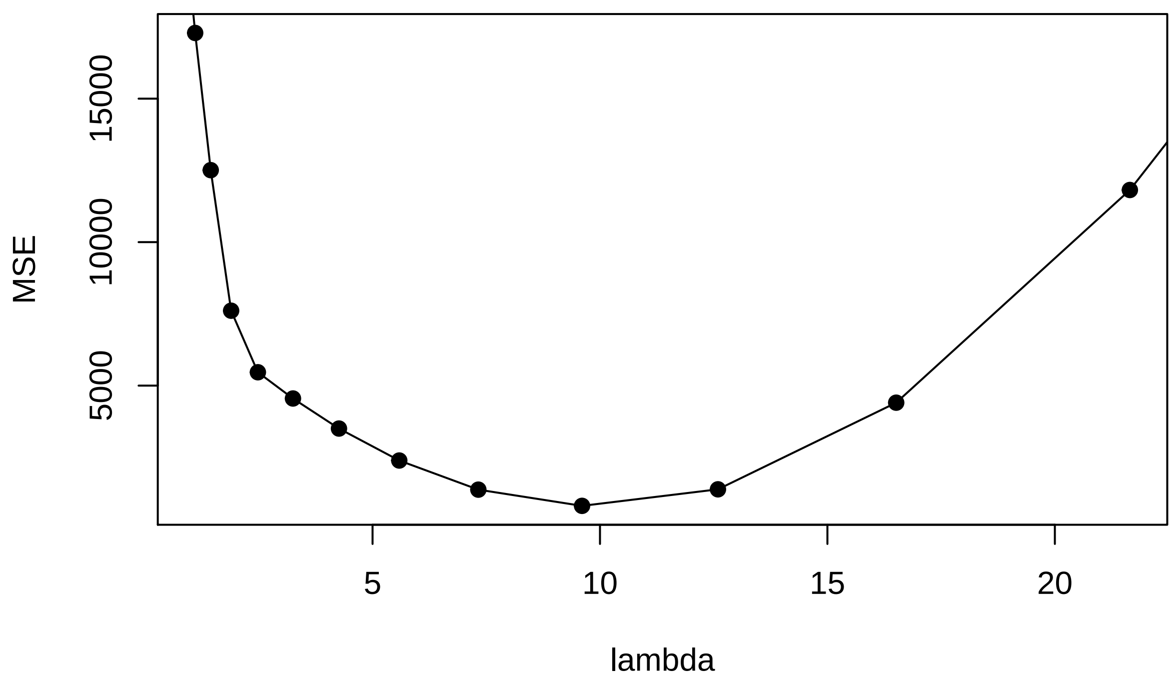
Then, the test error of each lambda is showed as below. We can also make plots of lambda versus MSE.

```r
# The test error of each lambda
cbind(lambda,MSE)
```

```
##           lambda        MSE
##  [1,] 28.3854431 25487.0375
##  [2,] 21.6495373 11816.6553
##  [3,] 16.5120715  4405.5510
##  [4,] 12.5937336  1386.3619
##  [5,]  9.6052228   810.1388
##  [6,]  7.3258898  1374.9794
##  [7,]  5.5874458  2390.0072
##  [8,]  4.2615370  3504.0176
##  [9,]  3.2502683  4551.3638
## [10,]  2.4789750  5465.1725
```

```
## [11,]  1.8907107  7610.5805
## [12,]  1.4420424 12508.4709
## [13,]  1.0998437 17288.9823
## [14,]  0.8388492 21551.4500
```

```
# Make plots of lambda versus MSE
sort_result = cbind(lambda,MSE)[order(cbind(lambda,MSE)[,2],decreasing=F),]
plot(MSE~lambda,data = sort_result[1:12,],pch=19)
lines(MSE~lambda)
```



We select the model with minimum test error as the best model. The test error, lambda and coefficient are showed as below.

```
# The test error and lambda of the best model
cbind(lambda,MSE)[which.min(MSE),]
```

```
##     lambda        MSE
##   9.605223 810.138841
```

```
# Report the coefficient and intercept of the best model.
best_model = data.frame(c(beta_all[,which.min(MSE)],beta0_all[which.min(MSE)]),
                        row.names = c(names(train_data_x),"Intercept"))
r = data.frame(best_model[best_model[,1] != 0,],
            row.names = rownames(best_model)[best_model[,1] != 0])
colnames(r) = c("Coefficient")
r
```

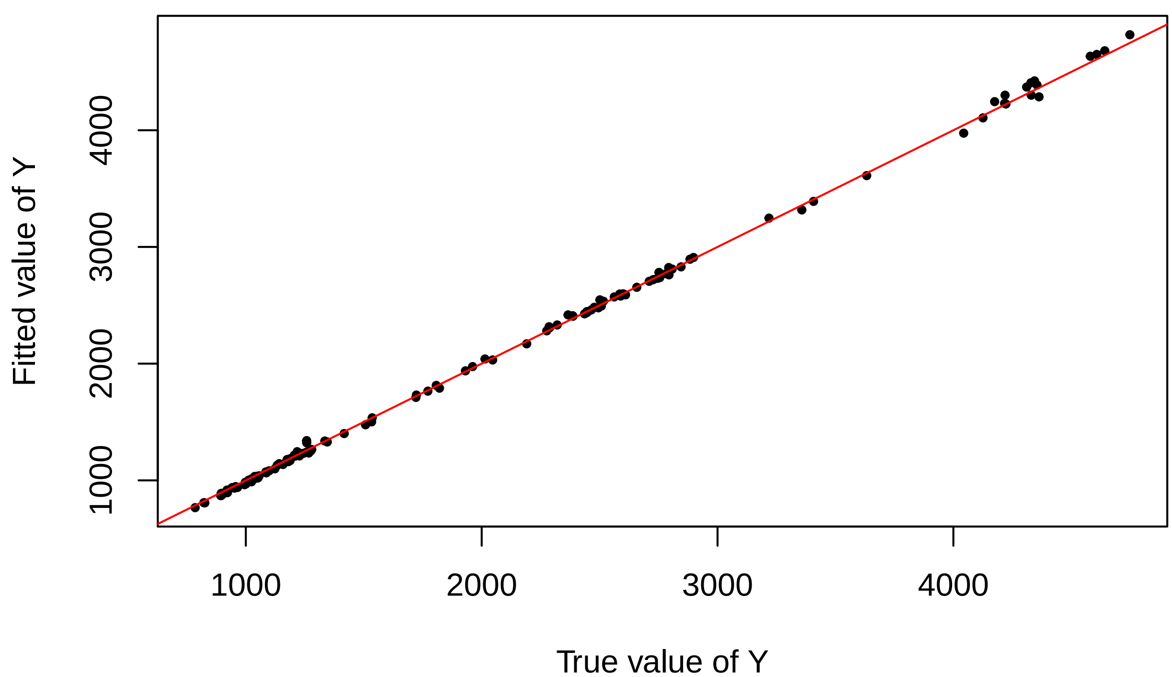
```
##                           Coefficient
## btc_market_cap           5.460582e-08
## btc_miners_revenue       4.265463e-05
## btc_cost_per_transaction 1.071608e+00
## Intercept                5.740412e+00
```

Thus, this is the coefficient and features in the best model. Now we can make a plot of the L1 Norm versus Coefficients (We don't show here due to the limitation of pages.)

```r
matplot(colSums(abs(beta_all)), t(beta_all), type="l",xlab = "L1 Norm",
        ylab = "Coefficients",lty = 1)
```

Finally, we make a plot of fitted Y values versus the true value of y. We can find that lasso perform pretty well on the data set.

```r
fitted_y_best_model = as.matrix(test_data_x) %*% best_model[1:22,1] + best_model[23,1]
plot(test_data_y,fitted_y_best_model,xlab = "True value of Y",
     ylab = "Fitted value of Y",pch = 19,cex=0.5)
abline(0,1,col="red")
```



There will be a **More thought** part to analysis why the performance of lasso on the test data is much better than the best subset selection. The reason is about the distribution difference between train and test data, and it may be overfitting in Problem 1 since there is an enormous search space. However, we will not show the analysis here due to the limitation of report page. You can find more detail in the Rmd file.