HW2 - STAT 542

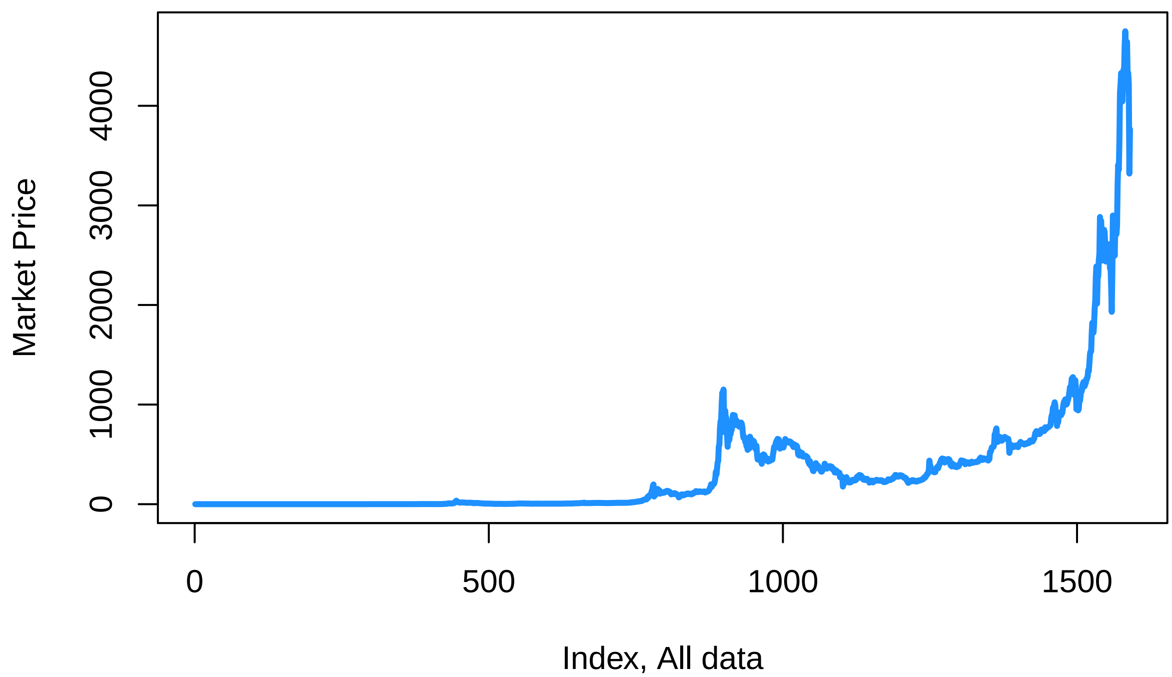
Josh Liu

September 21, 2017

```
btc <- read.csv("D:/Zhao/Documents/fall_2017/542/hw2/bitcoin_dataset.csv")
colnames(btc) <- c("Date","mktPrice","total","mktCap","tradeVol","blkSize","avgBlkSize","orphanedBlk","
```

Question 1

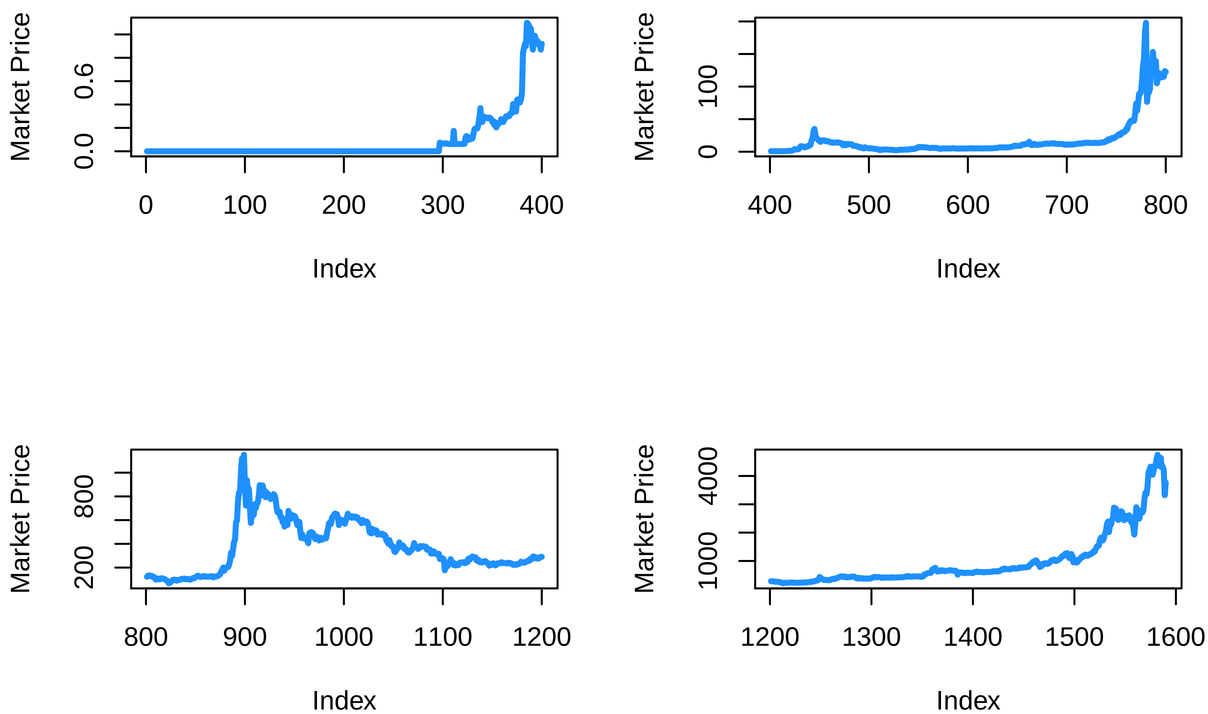
```
plot(btc$mktPrice , xlab = "Index, All data", type = "l", lwd = 3, ylab = "Market Price", col = "dodgerb
```



After inspecting the data, I take a closer look at the `price` variable:

```
par(mfrow = c(2,2))

plot(y = btc$mktPrice[1: 400] , x = 1:400, xlab = "Index", type = "l", lwd = 3, ylab = "Market Price", c

plot(y = btc$mktPrice[401: 800] , x = 401:800, xlab = "Index", type = "l", lwd = 3, ylab = "Market Pric

plot(y = btc$mktPrice[801: 1200]  , x = 801:1200, xlab = "Index", type = "l", lwd = 3, ylab = "Market Pr
```

```
plot(y = btc$mktPrice[1201: 1590] , x = 1201:1590, xlab = "Index", type = "l", lwd = 3, ylab = "Market
```




```
par(mfrow = c(1, 1))
```

The chart shows that prior to around Index 300, the market price of bitcoin remained identically at 0. My first thought is that I will not use observations indexed less than 300. However, since Professor Zhu said the training dataset should incluce `ALL` observations before 1/1/2017, I will keep them.

I will truncate data after 1/1/2017 in our training dataset.

In addition, observations with any variable being `NA` are removed.

```
b <- which(btc$Date == "2017-01-01 00:00:00")
btc_truncated <- btc[1: b, ]

# drop the variable of tradeVol, as instructed:
btc_truncated <- subset(btc_truncated, select = names(btc_truncated) != "tradeVol")
btc_training <- na.omit(btc_truncated)

btc_testing <- btc[b + 1: length(btc), ]
```

a)

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 3.3.3
```

```r
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.3.3
```

```r
bestsubset <- leaps(x = btc_training[, -c(1, 2, 3)],
      y = btc_training[, "mktPrice"],
      int = TRUE,
      method = c("Cp", "bic"),
      nbest = 1,
      names = colnames(btc_training[, -c(1, 2, 3)]), df=NROW(x))

bestsubset$which
```

```
##    mktCap blkSize avgBlkSize orphanedBlk transPerBlk medianConfTime hashRate dfcty minerRev transFee
## 1    TRUE   FALSE      FALSE       FALSE       FALSE          FALSE    FALSE FALSE    FALSE    FALSE
## 2    TRUE   FALSE      FALSE       FALSE       FALSE          FALSE    FALSE  TRUE    FALSE    FALSE
## 3    TRUE   FALSE      FALSE       FALSE       FALSE          FALSE     TRUE FALSE     TRUE    FALSE
## 4    TRUE   FALSE      FALSE       FALSE       FALSE          FALSE     TRUE FALSE     TRUE    FALSE
## 5    TRUE    TRUE      FALSE       FALSE       FALSE          FALSE     TRUE FALSE     TRUE    FALSE
## 6    TRUE    TRUE      FALSE       FALSE       FALSE          FALSE     TRUE FALSE     TRUE    FALSE
## 7    TRUE    TRUE      FALSE        TRUE       FALSE          FALSE     TRUE FALSE     TRUE    FALSE
## 8    TRUE    TRUE      FALSE        TRUE       FALSE          FALSE     TRUE  TRUE     TRUE    FALSE
## 9    TRUE    TRUE       TRUE        TRUE       FALSE          FALSE     TRUE  TRUE     TRUE    FALSE
## 10   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE    FALSE
## 11   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 12   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE    FALSE
## 13   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 14   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 15   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 16   TRUE    TRUE       TRUE        TRUE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 17   TRUE    TRUE       TRUE        TRUE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 18   TRUE    TRUE       TRUE        TRUE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 19   TRUE    TRUE       TRUE        TRUE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 20   TRUE    TRUE       TRUE        TRUE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
```

For best model with size 1: predictor is `btc_market_cap`; Size 2: `mktCap` and `difficulty`; Size 3: `mktCap`, `hashRate` and `minersRevenue`; Size 4: `mktCap`, `hashrate`, `costPerTrans` and `minersRevenue`; Size 5: `mktCap`, `hashRate`, `transTotal`, `blockSize`, and `minersRevenue`.

b)

In this part, I will use a package called `bestglm` to fit the best subset based on AIC and BIC.

In order to use this package, the dataset needs to be further cleaned, leaving no extraneous variables there.

I will drop the variable `Date`. In addition, the reponse variable `mktPrice` will be renamed as `y`.

```r
bestCPmodel <- which.min(bestsubset$Cp)
bestsubset$which[13,]
```

```
##                         mktCap                        blkSize                     avgBlkSize
##                           TRUE                           TRUE                           TRUE
```

```r
btc.training.for.bestglm <- subset(btc_training, select = names(btc_training) != "Date")
colnames(btc.training.for.bestglm)[1] <- "y"
```

```
bestCPmodel <- lm(y ~ mktCap + blkSize + avgBlkSize + orphanedBlk + medianConfTime + hashRate + dfcty +


library(bestglm)
```

```
## Warning: package 'bestglm' was built under R version 3.3.3
```

```
bestmodel.aic <-
    bestglm(Xy = btc.training.for.bestglm,
            family = gaussian,
            IC = "AIC",
            method = "exhaustive")
kable(ifelse(bestmodel.aic$BestModels[1,]== TRUE, "*", ""))
```

y	total	mktCap	blkSize	avgBlkSize	orphanedBlk	transPerBlk	medianConfTime	hashRate	dfcty	minerRe
*	*		*	*			*	*	*	*

```
bestAICmodel <- lm(y ~ total + blkSize + avgBlkSize + medianConfTime + hashRate + dfcty + minerRev + tra



bestmodel.bic <-
    bestglm(Xy = btc.training.for.bestglm,
            family = gaussian,
            IC = "BIC",
            method = "exhaustive")
kable(ifelse(bestmodel.bic$BestModels[1,]== TRUE, "*", ""))
```

y	total	mktCap	blkSize	avgBlkSize	orphanedBlk	transPerBlk	medianConfTime	hashRate	dfcty	minerRe
*			*							*

```
bestBICmodel <- lm(y ~ blkSize + minerRev + transFee + costPerTrans + uniqueAdd + transTotal + transExc

predictedCP <- predict(bestCPmodel, btc_testing)
predictedAIC <- predict(bestAICmodel, btc_testing)
predictedBIC <- predict(bestBICmodel, btc_testing)

predErrCP <- mean((predictedCP - btc_testing$mktPrice) ^ 2)
predErrAIC <- mean((predictedAIC - btc_testing$mktPrice) ^ 2)
predErrBIC <- mean((predictedBIC - btc_testing$mktPrice) ^ 2)

result2b <- as.matrix(c(predErrCP, predErrAIC, predErrBIC))
rownames(result2b) <- c("CP", "AIC", "BIC")
colnames(result2b) <- "Prediction Error"
kable(result2b)
```

	Prediction Error
CP	233.2786
AIC	2566.4795
BIC	31079.4935

c)

This part repeats a) and b), with response variable as $log(1 + Y)$, where Y is the variable `mktPrice`.

```
bestsubset <- leaps(x = btc_training[, -c(1, 2, 3)],
      y = log(btc_training[, "mktPrice"] + 1),
      int = TRUE,
      method = c("r2"),
      nbest = 1,
      names = colnames(btc_training[, -c(1, 2, 3)]), df=NROW(x))




bestsubset$which
```

```
##    mktCap blkSize avgBlkSize orphanedBlk transPerBlk medianConfTime hashRate dfcty minerRev transFee
## 1    TRUE   FALSE      FALSE       FALSE       FALSE          FALSE    FALSE FALSE    FALSE    FALSE
## 2   FALSE    TRUE      FALSE       FALSE       FALSE          FALSE    FALSE FALSE    FALSE    FALSE
## 3   FALSE    TRUE      FALSE       FALSE       FALSE          FALSE    FALSE  TRUE    FALSE    FALSE
## 4   FALSE    TRUE      FALSE       FALSE       FALSE          FALSE    FALSE  TRUE    FALSE    FALSE
## 5    TRUE    TRUE       TRUE       FALSE       FALSE          FALSE    FALSE FALSE    FALSE    FALSE
## 6    TRUE    TRUE       TRUE       FALSE       FALSE          FALSE    FALSE  TRUE    FALSE    FALSE
## 7    TRUE    TRUE       TRUE       FALSE       FALSE          FALSE    FALSE  TRUE    FALSE    FALSE
## 8    TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE    FALSE
## 9    TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE     TRUE
## 10   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE     TRUE
## 11   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE     TRUE
## 12   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE     TRUE
## 13   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE    FALSE  TRUE    FALSE     TRUE
## 14   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE     TRUE  TRUE    FALSE     TRUE
## 15   TRUE    TRUE       TRUE       FALSE       FALSE           TRUE     TRUE  TRUE     TRUE     TRUE
## 16   TRUE    TRUE       TRUE       FALSE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 17   TRUE    TRUE       TRUE       FALSE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 18   TRUE    TRUE       TRUE       FALSE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 19   TRUE    TRUE       TRUE       FALSE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
## 20   TRUE    TRUE       TRUE        TRUE        TRUE           TRUE     TRUE  TRUE     TRUE     TRUE
```

To answer question in 1c, please refer to the table above.

```
bestCPmodel <- which.min(bestsubset$Cp)
bestsubset$which[13,]
```

```
##                          mktCap                          blkSize                       avgBlkSize
##                            TRUE                             TRUE                             TRUE
```

```
btc.training.for.bestglm <- subset(btc_training, select = names(btc_training) != "Date")
colnames(btc.training.for.bestglm)[1] <- "y"

bestCPmodel <- lm(y ~ mktCap , data = btc.training.for.bestglm)


library(bestglm)

btc.training.logged <- btc.training.for.bestglm
btc.training.logged$y <- log(btc.training.logged$y + 1)
```

```r
bestmodel.aic <-
    bestglm(Xy = btc.training.logged,
            family = gaussian,
            IC = "AIC",
            method = "exhaustive")
kable(ifelse(bestmodel.aic$BestModels[1,]== TRUE, "*", ""))
```

y	total	mktCap	blkSize	avgBlkSize	orphanedBlk	transPerBlk	medianConfTime	hashRate	dfcty	minerRe
*		*	*	*			*	*	*	*

```r
bestAICmodel <- lm(y ~ mktCap + blkSize + avgBlkSize + medianConfTime + hashRate + dfcty + minerRev + t
```

```r
bestmodel.bic <-
    bestglm(Xy = btc.training.logged,
            family = gaussian,
            IC = "BIC",
            method = "exhaustive")
kable(ifelse(bestmodel.bic$BestModels[1,]== TRUE, "*", ""))
```

y	total	mktCap	blkSize	avgBlkSize	orphanedBlk	transPerBlk	medianConfTime	hashRate	dfcty	minerRe
*		*	*				*			*

```r
bestBICmodel <- lm(y ~ mktCap + blkSize + medianConfTime  + minerRev + transFee + costPerTrans + transT

predictedCP <- predict(bestCPmodel, btc_testing)
predictedAIC <- predict(bestAICmodel, btc_testing)
predictedBIC <- predict(bestBICmodel, btc_testing)

btc.testing.logged <- btc_testing
btc.testing.logged$y <- log(btc.testing.logged$mktPrice + 1)

predErrCP <- mean((predictedCP - btc.testing.logged$mktPrice) ^ 2)
predErrAIC <- mean((predictedAIC - btc.testing.logged$mktPrice) ^ 2)
predErrBIC <- mean((predictedBIC - btc.testing.logged$mktPrice) ^ 2)

result2b <- as.matrix(c(predErrCP, predErrAIC, predErrBIC))
rownames(result2b) <- c("CP", "AIC", "BIC")
colnames(result2b) <- "Prediction Error"
kable(result2b)
```

	Prediction Error
CP	13169.0
AIC	894189.1
BIC	892334.7

Question 2

Part I.

```
library(MASS)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.3.3
## Warning: package 'Matrix' was built under R version 3.3.3
## Warning: package 'foreach' was built under R version 3.3.3
```

```
set.seed(200)
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)

# check OLS
lm(y ~ X)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Coefficients:
## (Intercept)           X1           X2           X3           X4           X5           X6           X
##    0.586100     0.796638     0.490602     0.618338     0.701075     0.997051     0.033797    -0.0145
```

This is the β in OLS regression.

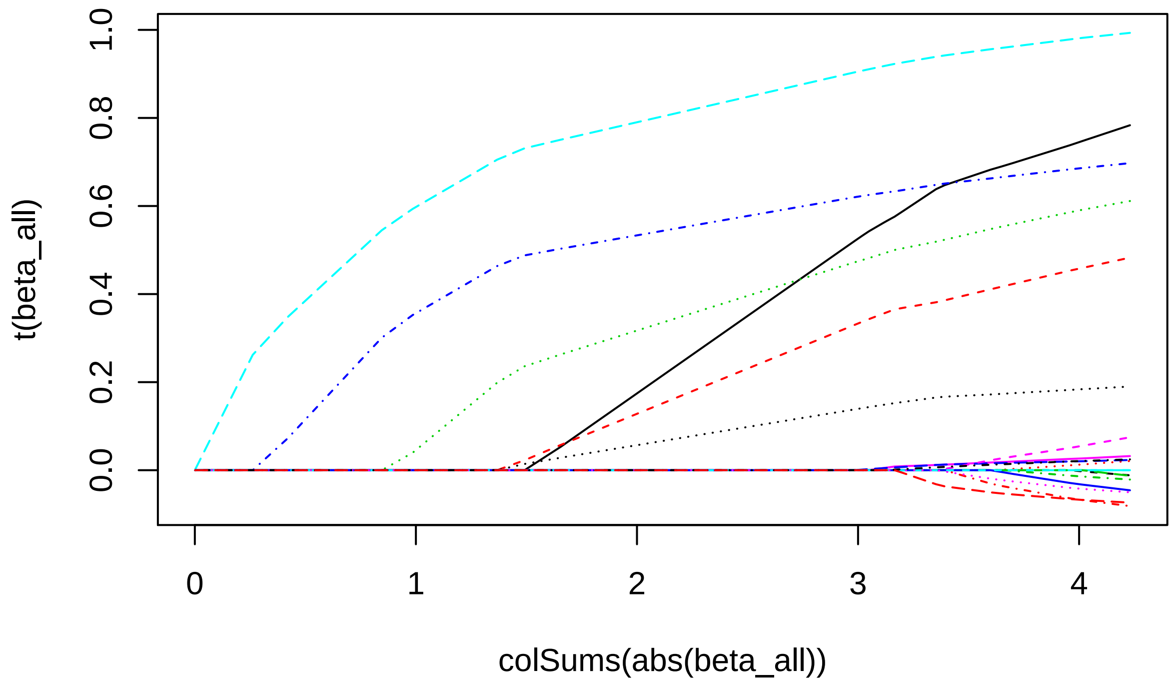
```
# now start the Lasso
# First we scale and center X, and record them.
# Also center y and record it. dont scale it.
# now since both y and X are centered at 0, we don't need to worry about the intercept anymore.
# this is because for any beta, X %*% beta will be centered at 0, so no intercept is needed.
# However, we still need to recover the real intercept term after we are done estimating the beta.
# The real intercept term can be recovered by using the x_center, x_scale, y2, and the beta parameter y
# There are other simpler ways to do it too, if you think carefully.

x_center  <-colMeans(X)
x_scale <- apply(X, 2, sd)
```

```r
X2 <- scale(X)

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

# now start to write functions
# prepare the soft thresholding function (should be just one line, or a couple of)

soft_th <- function(b, pen)
{
    ifelse((abs(b)-pen)<0, 0, sign(b)*(abs(b)-pen))
}

# initiate lambda. This is one way to do it, the logic is that I set the first lambda as the largetst g
# if you use this formula, you will need to calculate this for the real data too.

#lambda = exp(seq(log(max(abs(cov(X2, y2)))), log(0.001), length.out = 100))
lambda = glmnet(X, y)$lambda

# you should write the following function which can be called this way
# LassoFit(X2, y2, mybeta = rep(0, ncol(X2)), mylambda = lambda[10])

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

    for (k in 1:maxitr)
    {
        # compute residual
        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
            # so that the residual is now the residual after fitting all other variables
          r=r + myX[,j]*mybeta[j]

            # apply the soft thresholding function to the ols estimate of the jth variable

          mybeta[j]=soft_th(b=sum(myX[,j]*r)/sum((myX[,j]*myX[,j])),pen=mylambda)

            # remove the new effect of jth varaible out of r
          r = r - myX[,j]*mybeta[j]
        }

        if (k > 10)
        {
            # this is just my adhoc way of stoping rule, you dont have to use it
```

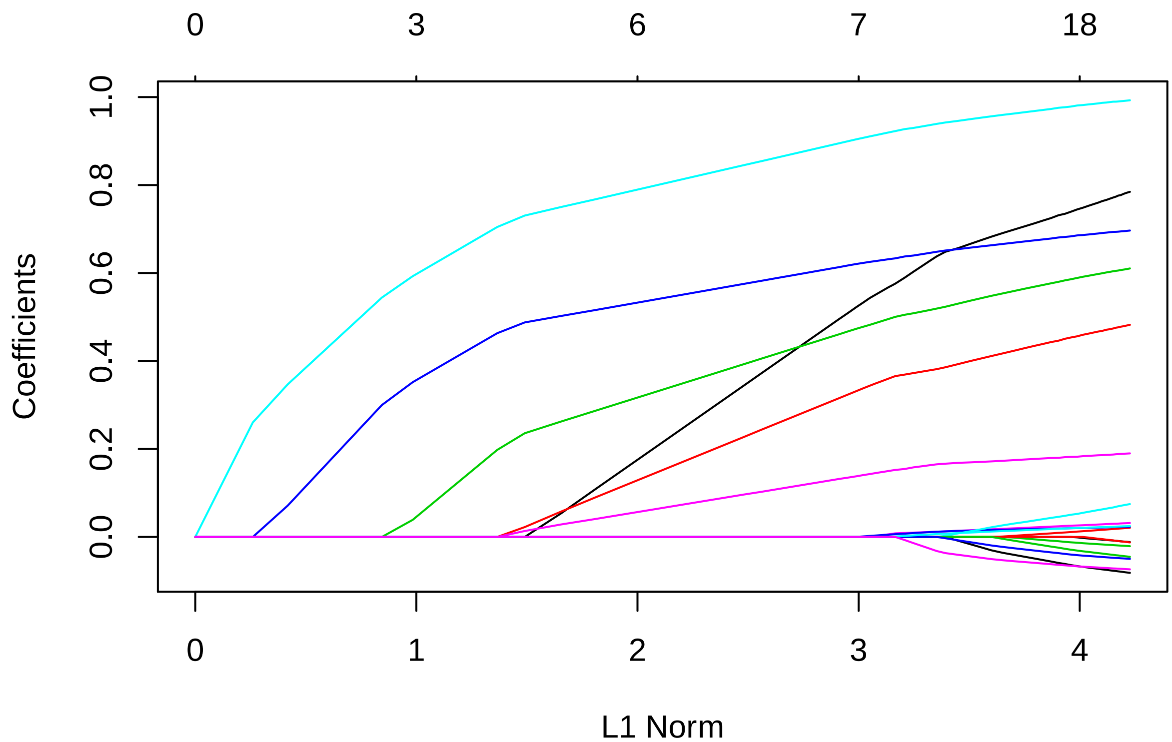
```r
            if (sum(abs(f[(k-9):k] - mean(f[(k-9):k]))) < tol) break;
        }
    }
    return (mybeta)
}

# you should test your function on a large lambda (penalty) level.
# this should produce a very spase model.
# keep in mind that these are not the beta in the original scale of X

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

```
##  [1] 0.00000000 0.00000000 0.09011326 0.34380663 0.62227913 0.00000000 0.00000000 0.00000000 0.000000
```

```r
# now 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))

# this part gets pretty tricky: you will initial a zero vector for bhat,
# then throw that into the fit function using the largest lambda value.
# that will return the fitted beta, then use this beta on the next (smaller) lambda value
# iterate until all lambda values are used

bhat <- rep(0, ncol(X2)) # initialize it

for (i in 1:length(lambda)) # loop from the largest lambda value
{
    # if your function is correct, this will run pretty fast

    bhat <- LassoFit(X2, y2, bhat, lambda[i])

    # this is a tricky part, since your data is scaled, you need to figure out how to scale that back
    # save the correctly scaled beta into the beta matrix

    beta_all[, i] <- bhat/x_scale

    # here, you need to figure out a way to recalculte the intercept term in the original, uncentered a

    beta0_all[i]  <- ymean-mean(apply(X%*%beta_all[,i],1,sum))
}


# now you have the coefficient matrix
# each column correspond to one lambda value
#rbind("intercept" = beta0_all, beta_all)


# you should include a similar plot like this in your report
# feel free to make it look better
matplot(colSums(abs(beta_all)), t(beta_all), type="l")
```




```r
# The following part provides a way to check your code.
# You do not need to include this part in your report.

# However, keep in mind that my original code is based on formula (3)
# if you use other objective functions, it will be different, and the results will not match

# load the glmnet package and get their lambda
library(glmnet)

# this plot should be identical (close) to your previous plot
plot(glmnet(X, y))
```




```r
# set your lambda to their lambda value and rerun your algorithm
#lambda = glmnet(X, y)$lambda

# then this distance should be pretty small
# my code gives distance no more than 0.01
max(abs(beta_all - glmnet(X, y)$beta))
```

```
## [1] 0.002338841
```

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

```
## [1] 0.002204841
```

Part II.

```r
X2 <- as.matrix(btc_training[,-which(names(btc_training) %in% c("mktPrice","Date"))])
Y2 <- as.matrix(btc_training[,which(names(btc_training) == "mktPrice")])
#lambda <- seq(20,30,by=0.1)


x_center <- colMeans(X2)
x_scale <- apply(X2, 2, sd)
X3 <- scale(X2)

ymean <- mean(Y2)
Y3 <- Y2 - ymean
```

```
lambda <- glmnet(X2, Y2)$lambda

beta_all <- matrix(NA, ncol(X3), length(lambda))

beta0_all <- rep(NA, length(lambda))

bhat <- rep(0, ncol(X3)) # initialize it
for (i in 1:length(lambda))
{
    #bhat <- LassoFit(X3, Y3, mybeta = rep(0, ncol(X3)), mylambda = lambda[i], tol = 1e-7, maxitr = 500
    bhat <- LassoFit(X3,Y3, bhat, lambda[i])
    beta_all[, i] <- bhat / x_scale
    beta0_all[i] <- ymean-mean(apply(X2%*%beta_all[,i],1,sum))
}
```

```
Training=btc[1:1460,]
Test=btc[1461:1588,]
Training=Training[,-which(names(Training) == "tradeVol")]
Test=Test[,-which(names(Test) == "tradeVol")]


Test_dataset=as.matrix(Test[,-which(names(Training) %in% c("mktPrice","Date"))])
Test_Y=Test[,which(names(Training) == c("mktPrice"))]
error= matrix(0,length(lambda))
for (i in 1:length(lambda))
{
error[i]=mean((Test_Y-as.vector(beta0_all[i]+Test_dataset%*%matrix(beta_all[,i],ncol=1)))^2)
}

error
```

```
##              [,1]
##  [1,] 4649208.3076
##  [2,] 3748822.9388
##  [3,] 3012722.9293
##  [4,] 2412001.3684
##  [5,] 1922749.3244
##  [6,] 1525199.2722
##  [7,] 1203014.6628
##  [8,]  942700.7584
##  [9,]  741557.7991
## [10,]  606356.7509
## [11,]  495028.8624
## [12,]  403439.3026
## [13,]  328162.5102
## [14,]  266361.1283
## [15,]  215685.5507
## [16,]  174190.5760
## [17,]  140266.2594
## [18,]  112580.5488
## [19,]   90031.6983
## [20,]   71708.7987
## [21,]   56859.0402
## [22,]   44860.5645
```

```
## [23,]   35199.9528
## [24,]   27453.5605
## [25,]   21272.0438
## [26,]   16367.5340
## [27,]   12503.0078
## [28,]    9476.6587
## [29,]    6712.4414
## [30,]    4654.2135
## [31,]    3161.0848
## [32,]    2117.9538
## [33,]    1430.9632
## [34,]    1023.7416
## [35,]     834.2971
## [36,]     812.4504
## [37,]     917.7148
## [38,]    1117.5463
## [39,]    1385.9005
## [40,]    1702.0418
## [41,]    2049.5640
## [42,]    2415.5830
## [43,]    2790.0736
## [44,]    3165.3241
## [45,]    3535.4895
## [46,]    3896.2251
## [47,]    4244.3864
## [48,]    4577.7844
## [49,]    4894.9858
## [50,]    5195.1507
## [51,]    5477.9008
## [52,]    5743.2125
## [53,]    6621.3200
## [54,]    8219.5255
## [55,]    9860.4568
## [56,]   11530.3955
## [57,]   13197.0895
## [58,]   14836.1902
## [59,]   16429.6951
## [60,]   17964.6736
## [61,]   19432.2270
## [62,]   20826.6405
## [63,]   22144.6945
## [64,]   23385.1052
## [65,]   24550.3954
## [66,]   25773.7483
## [67,]   27419.2046
## [68,]   28971.1687
## [69,]   30423.5405
## [70,]   31778.6348
## [71,]   33039.7013
## [72,]   34205.0542
## [73,]   35278.6201
```

```
lambda[which.min(error)] # This outputs the lowest testing error.
```

```
## [1] 9.582842
```

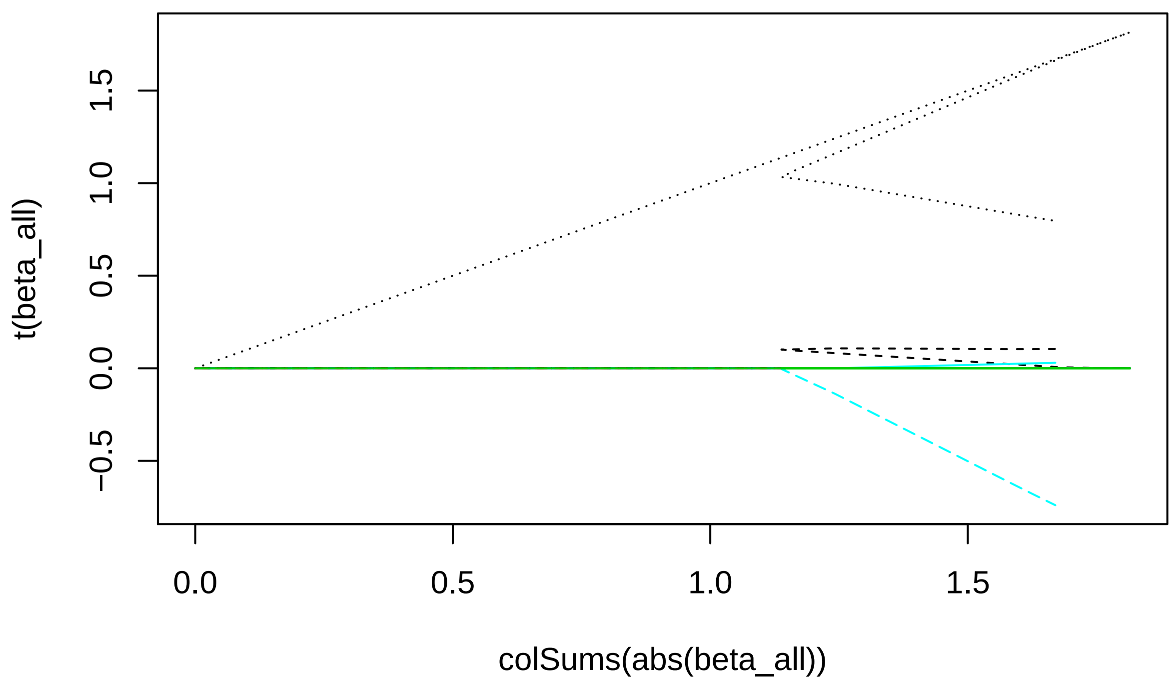
```r
model=glmnet(X2,Y2)
test_pred=predict(model,as.matrix(Test_dataset),type='response')
error=mean((test_pred-Test[2])^2)
max(abs(beta_all - glmnet(X2, Y2)$beta))
```

```
## [1] 0.07641398
```

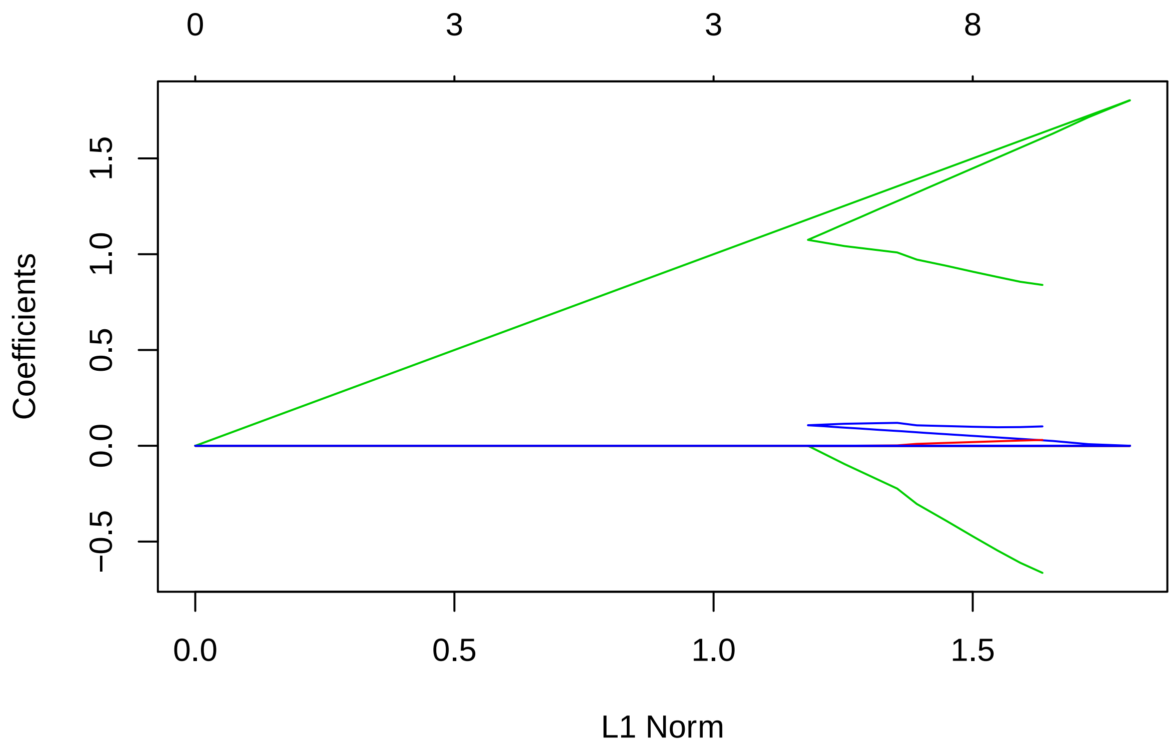
```r
max(abs(beta0_all - glmnet(X2, Y2)$a0))
```

```
## [1] 0.1367954
```

```r
matplot(colSums(abs(beta_all)), t(beta_all), type="l")
```




```r
plot(glmnet(X2,Y2))
```


0
3
3
8
Coefficients
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
L1 Norm