Name:Jiahong Hu
UNI:jh3561
HW03
STAT W 4400

Problem 1

Part 1 See Code starting at Page 3
Part 2 See Code starting at Page 3

Part 3
- Test result for USPS data when B= 60

Code:

```
x<-read.table("uspsdata.txt")
y<-read.table("uspscl.txt")[,1]
n=nrow(x)
set.seed=1
ada_60=AdaBoost(x,y,60)
allpars_60=ada_60$allpars
alpha_60=ada_60$alpha
c_hat_60=agg_class(x,alpha_60,allpars_60)
```

sample of alpha and allpars

```
> ada_60
$alpha
 [1] 1.6214863 1.0633568 1.1600821 0.8927868 0.6038988 0.6408494 0.4421925 0.6084815
 [9] 0.5356946 0.4858256 0.4617793 0.5093852 0.4886119 0.4162222 0.3850488 0.4431448
[17] 0.3986194 0.4600670 0.3619578 0.3006652 0.4583975 0.3954913 0.3167800 0.3619256
[25] 0.3594495 0.3720204 0.3334995 0.2616918 0.2536257 0.3418120 0.2324946 0.3424301
[33] 0.3370463 0.3250948 0.3663054 0.2715065 0.3673252 0.2686530 0.3924605 0.3322247
[41] 0.2475923 0.3166657 0.3238810 0.2786936 0.2169385 0.2938265 0.3311058 0.2837596
[49] 0.3041426 0.2685484 0.3341568 0.2811435 0.3225015 0.1898453 0.3190682 0.2763163
[57] 0.2317246 0.1950018 0.2858904 0.3213934

$allpars
$allpars[[1]]
$allpars[[1]]$j
[1] 165

$allpars[[1]]$theta
[1] 65.59

$allpars[[1]]$m
[1] 1
```

classification y

```
> c_hat_60
  [1] -1 -1  1  1  1 -1  1  1  1 -1  1  1  1 -1  1  1 -1 -1  1  1 -1  1  1  1  1  1 -1 -1 -1 -1
 [31]  1  1  1 -1 -1  1  1 -1  1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1  1  1 -1 -1  1 -1 -1  1 -1
 [61] -1  1  1 -1  1 -1  1  1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1
 [91] -1 -1  1  1 -1  1  1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1 -1
[121]  1  1  1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1  1 -1  1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1 -1
[151] -1  1  1  1  1  1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1  1  1  1 -1 -1  1 -1 -1  1  1  1 -1 -1
[181]  1  1  1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1  1  1 -1  1  1
```

- cross validation using B=100

test error matrix

```
> test_error_matrix
       [,1]  [,2]  [,3]  [,4]  [,5]
 [1,] 0.200 0.200 0.250 0.325 0.250
 [2,] 0.200 0.200 0.250 0.325 0.250
 [3,] 0.150 0.200 0.225 0.325 0.150
 [4,] 0.150 0.225 0.225 0.275 0.150
 [5,] 0.175 0.175 0.225 0.225 0.150
 [6,] 0.125 0.175 0.225 0.275 0.150
 [7,] 0.200 0.150 0.200 0.200 0.175
 [8,] 0.175 0.200 0.225 0.225 0.175
 [9,] 0.175 0.175 0.200 0.250 0.125
[10,] 0.150 0.125 0.200 0.225 0.125
[11,] 0.150 0.125 0.200 0.225 0.150
[12,] 0.150 0.100 0.200 0.225 0.125
[13,] 0.175 0.150 0.200 0.225 0.150
[14,] 0.175 0.125 0.225 0.225 0.125
[15,] 0.175 0.125 0.200 0.225 0.125
[16,] 0.200 0.125 0.200 0.225 0.125
[17,] 0.175 0.125 0.200 0.225 0.125
[18,] 0.175 0.100 0.200 0.225 0.100
[19,] 0.175 0.100 0.200 0.225 0.100
```

train error matrix

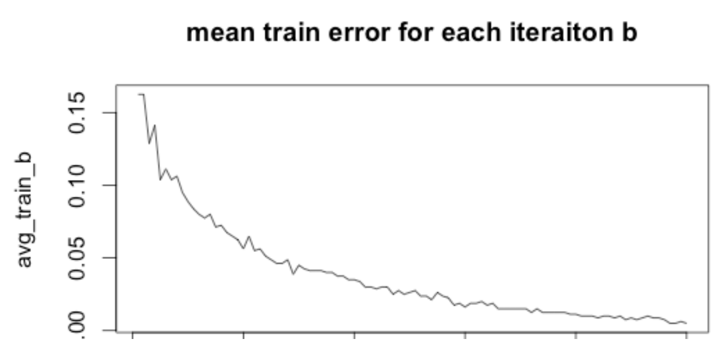
```
> train_error_matrix
         [,1]    [,2]    [,3]    [,4]    [,5]
 [1,] 0.18125 0.18125 0.14375 0.16250 0.14375
 [2,] 0.18125 0.18125 0.14375 0.16250 0.14375
 [3,] 0.15000 0.13750 0.10625 0.12500 0.12500
 [4,] 0.15000 0.19375 0.11875 0.12500 0.11875
 [5,] 0.13125 0.13125 0.09375 0.07500 0.08750
 [6,] 0.10625 0.13750 0.07500 0.13125 0.10625
 [7,] 0.13125 0.13750 0.06875 0.07500 0.10625
 [8,] 0.11875 0.14375 0.06875 0.08750 0.11250
 [9,] 0.12500 0.14375 0.05625 0.06250 0.08750
[10,] 0.10000 0.11875 0.05000 0.06875 0.10625
[11,] 0.11250 0.12500 0.03125 0.06250 0.08750
[12,] 0.08750 0.12500 0.04375 0.05625 0.08750
[13,] 0.08125 0.11250 0.04375 0.06250 0.08750
[14,] 0.06250 0.13125 0.05000 0.06875 0.08750
[15,] 0.08125 0.10625 0.03125 0.05625 0.08125
[16,] 0.05000 0.12500 0.04375 0.04375 0.10000
```

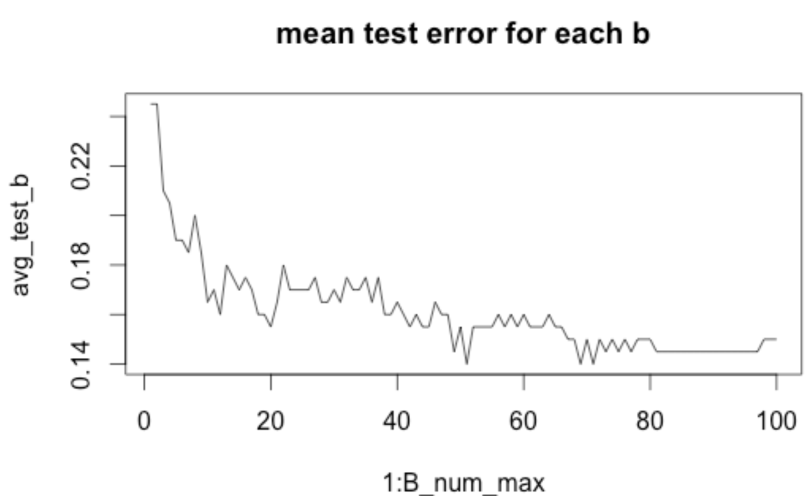
average of test error and train error under cross validation when B =100

```
> avg_test_b
 [1] 0.205 0.205 0.220 0.205 0.195 0.165 0.175 0.160 0.165 0.175 0.160 0.190 0.170 0.170 0.155
[16] 0.165 0.160 0.155 0.175 0.155 0.175 0.145 0.155 0.160 0.160 0.150 0.145 0.160 0.155 0.145
[31] 0.155 0.145 0.160 0.150 0.145 0.150 0.150 0.160 0.160 0.155 0.165 0.150 0.155 0.155 0.155
[46] 0.155 0.150 0.150 0.140 0.155 0.145 0.155 0.145 0.150 0.135 0.150 0.150 0.150 0.140 0.155
```

```
> avg_train_b
  [1] 0.16125 0.16125 0.13125 0.12000 0.11500 0.10750 0.10125 0.09375 0.09750 0.09000 0.08375
 [12] 0.08000 0.08000 0.07250 0.08000 0.06375 0.06625 0.06375 0.06375 0.06125 0.06250 0.05500
 [23] 0.05625 0.05625 0.05375 0.05250 0.05000 0.04875 0.04750 0.04375 0.05000 0.04875 0.04750
 [34] 0.05000 0.04625 0.04500 0.04375 0.04375 0.04125 0.04375 0.04375 0.04000 0.04375 0.03625
 [45] 0.04000 0.03750 0.04000 0.03750 0.03875 0.03875 0.03875 0.03750 0.03500 0.03500 0.03625
 [56] 0.03375 0.03625 0.03250 0.03125 0.03000 0.03000 0.02750 0.03000 0.02500 0.02875 0.02500
 [67] 0.02750 0.02750 0.02875 0.02500 0.02125 0.02125 0.02125 0.02125 0.02250 0.02250 0.01750
 [78] 0.02000 0.01625 0.01625 0.01500 0.01750 0.01625 0.01500 0.01500 0.01500 0.01625 0.01375
 [89] 0.01375 0.01250 0.01250 0.01375 0.01125 0.01125 0.01375 0.01375 0.01250 0.01250 0.01000
[100] 0.01125
```

Part 4





As we can see from photo above, the training error reduces when B becomes larger and larger. When the training error is approximately zero, it means the model is overfitting. when d>40, training error is almost zero.

The testing error is reduced a lot at first,especially before b=40 but later it become stable at a low level. It shows the adaboost method is very rubust.

so I suggest to use b between 15-40.

Problem 2 - Lq regression

Part 1:

q=0.5 encourages spare estimate but q =4 does not encourage spare estimate.

when q=0.5, the minimum l(o.5) distance to the origin is located at a point on the axis for every eclipse iso-line of the square loss that has an intersection with an axis.Therefore, the entry for the other axis must be zero. However, in terms of q =4 , it encourages beta(i) if approximately same size.

Part 2: which achieve the smallest cost under the l-q constrained least square cost function?

when q =0.5, the cost optimal is x3 because it is at the beta(2) axis.

when q =4, the cost optimal is x4 because the penalty isoline could be shrieked until it meets the ellipse at x4, which is the only choice.

Code

Part 1:

AdaBoost Function

```
# x is the n*p data matrix
# y is the responce variable, -1 and 1
# B_num is the number of weak classifiers

## part a ##

## adaboost function is aimed to find alpha, allpars under the adaboost
AdaBoost=function(x,y,B_num){

    n=nrow(x)

    # start with an intial weight vector
    w_original=1/n
    w_vector=rep(w_original,times=n)

    ##pre-allocate en empty vector and list
    alpha=rep(NA,times=B_num)
    allpars=rep(list(list()),length=B_num)


    for(i in 1:B_num){
       # step 1 Training parameters using train function
       pars=train(x,w_vector,y)
       allpars[[i]]=pars
       # step 2  classification y function
       label=Classify(x,pars)
       # step 3 check errors
       error_ind=(y!=label)
       # step 4 compute errors
       w_error_vector=w_vector[error_ind]
       error=sum(w_error_vector)/sum(w_vector)
       # step 5 compute alpha
       alpha_value=log((1-error)/error)
       alpha[i]=alpha_value
       #step 6 compute the new weight
       w_vector=w_vector*exp(alpha_value*error_ind)


}
    return (list(alpha=alpha,allpars=allpars))
}
```

agg_class function

```
## part b ##
## agg_class function make the final y classfication under the adaboost, which is c_hat

agg_class=function(x,alpha,allpars){

    n=nrow(x)
    B_num=length(alpha)
    ## Create an empty list
    lable_matrix=matrix(NA,nrow=B_num,ncol=n)

    for(i in 1:B_num){
      lable_matrix[i,]=alpha[i]*Classify(x,allpars[[i]])
    }

    hat=apply(lable_matrix,2,sum)
    c_hat=sign(hat)
    return(c_hat)
}
```

Part 2

Train and classify function

```
#### problem 2 ####

#part a #

train=function (x, w, y){
     p=ncol(x)
     n=nrow(x)

     y_hat=vector()
     error=vector()
     best_i=vector()
     best_i_error=vector()


     for (j in 1:p) {

          #reoder x, y,w from small to large
          x_col_j=x[,j]
          ind_order=order(x_col_j)
          x_col_j_new=x_col_j[ind_order]
          y_new=y[ind_order]
          w_new=w[ind_order]
```

```
            # check for duplicate
            x_col_j_new_single=unique(x_col_j_new)
            n_new=length(x_col_j_new_single)

            for(i in 1:n_new){
                  ind_1=x_col_j_new <= x_col_j_new_single[i]
                  ind_2=x_col_j_new > x_col_j_new_single[i]
                  y_hat[ind_1]=-1
                  y_hat[ind_2]=1
                  error[i]=sum(w_new*(y_new !=y_hat))/sum(w_new)

            }

            best_i[j]= x_col_j_new_single[which.min(error)]
            best_i_error[j]=min(error)
      }
      best_j=which.min(best_i_error)
      best_i=best_i[best_j]
      pars<-list(j = best_j, theta = best_i,m = 1)
      return(pars)
}




#part b #
Classify=function(x,pars){
  label=vector()
  j_ind=pars$j
  x_j=x[,j_ind]
  ind_1=x_j <= pars$theta
  ind_2=x_j > pars$theta
  label[ind_1]=-1
  label[ind_2]=1
  return(label)
}
```

```
#### problem 3 ####

# test if the algorithm works
# B_num=8
x<-read.table("uspsdata.txt")
y<-read.table("uspscl.txt")[,1]
n=nrow(x)
set.seed=1
ada_60=AdaBoost(x,y,60)
allpars_60=ada_60$allpars
alpha_60=ada_60$alpha
c_hat_60=agg_class(x,alpha_60,allpars_60)


## implement 5-fold cross validation


B_num_max=100

test_error_matrix=matrix(NA,nrow=100,ncol=5)
train_error_matrix=matrix(NA,nrow=100,ncol=5)


set.seed=50
ind=sample.int(200)
ind_1=which(ind<=40)
ind_2=which(ind>=41 & ind<=80)
ind_3=which(ind>=81 & ind<=120)
ind_4=which(ind>=121 & ind<=160)
ind_5=which(ind>=161 & ind<=200)

ind_matrix=matrix(NA,nrow=5,ncol=40)
ind_matrix[1,]=ind_1
ind_matrix[2,]=ind_2
ind_matrix[3,]=ind_3
ind_matrix[4,]=ind_4
ind_matrix[5,]=ind_5


for(i in 1:5){
	test_ind=ind_matrix[i,]
	test_ind=as.vector(test_ind)
	train_ind=ind_matrix[-i,]
	train_ind=as.vector(train_ind)

	x_test=x[test_ind,]
	x_train=x[train_ind,]
```

```
    y_test=y[test_ind]
    y_train=y[train_ind]

    result=AdaBoost(x_train,y_train,B_num_max)
    allpars=result$allpars
    alpha=result$alpha

    for(j in 1:B_num_max){


        c_hat_test=agg_class(x_test,alpha[1:j],allpars[1:j])
        test_error_ind=(c_hat_test != y_test)
        test_error_matrix[j,i]=mean(test_error_ind)

        c_hat_train=agg_class(x_train,alpha[1:j],allpars[1:j])
        train_error_ind=(c_hat_train != y_train)
        train_error_matrix[j,i]=mean(train_error_ind)




    }


}


# vector of length b that represent the average test error for each iteration b
avg_test_b=apply(test_error_matrix,1,mean)
which.min(avg_test_b)
# vector of length b that represent the average train error for each iteration b
avg_train_b=apply(train_error_matrix,1,mean)






### Problem 4 ###

plot(1:B_num_max,avg_test_b,type="l",main="mean test error for each b")
plot(1:B_num_max,avg_train_b,type="l",main="mean train error for each iteraiton b")
```