Decision Trees Homework

Dmitrii Dunin, ITU ID 94739

International Technological University

Question 1

```
> rm(list=ls())
> require(graphics)
> require(stringr)
> require(rpart)
> require(ISLR)
> setwd("F:/Workspace/R/Homework3")
```

1) Once again check out wine quality data set described in the web page below: http://archive.ics.uci.edu/ml/machine-learning-databases/winequality/winequality.names Remember the Red Wine data set (winequality-red.csv) contains 1599 observations of 11 attributes. The median score of the wine tasters is given in the last column. Note also that the delimiter used in this file is a semi colon and not a comma. This problem is to create an ordinary least squares linear model (use the lm function in R) for this data set using the first 1400 observations. Don't forget to scale each column before you create the model. Next check the model's performance on the last 199 observations. How well did the model predict the results for the last 199 observations? What measure did you use to evaluate how well the model did this prediction? Next use the model to predict the results for the whole data set and measure how well your model worked. (hint: use the r function lm and the regression example from class)

Answer 1

```
> set.seed(pi)
> wine_data<-read.csv("winequality-red.csv",header = TRUE, sep=";")
> # Scaling data before constructing models
> scaled_wine_data<-scale(wine_data)
> # First 1400 as train data
> wine_train<-scaled_wine_data[1:1400,]
> x_wine_train<-wine_train[,1:11]
> y_wine_train<-wine_train[,12]
> # Last 199 as test data
> wine_test<-scaled_wine_data[1401:dim(wine_data)[1],]
> x_wine_test<-wine_test[,1:11]
> y_wine_test<-wine_test[,12]
> # Contructing model, data = 1400 rows with first 11 columns
> Predicted_OLS<-lm(y_wine_train~., data = as.data.frame(x_wine_train))
```

```
> OLS_coef <- coef(Predicted_OLS)
> # Using model to predict 199 last records
> predicted_OLS_quality<-predict(Predicted_OLS, newdata = as.data.frame(x_wine_test))
> # Calculating error
> dY<-y_wine_test - predicted_OLS_quality
> testErr_199 <- sqrt(sum(dY*dY))/(length(y_wine_test))
> paste("199 Last records predticiton error = ", testErr_199)

[1] "199 Last records predticiton error =  0.061249097889956"

> # Taking full data set
> wine_data_x<-scaled_wine_data[,1:11]
> wine_data_y<-scaled_wine_data[,12]
> # Predicting last columng for all data set
> lm_wine_data<-predict(Predicted_OLS, newdata = as.data.frame(wine_data_x))
> dYData <- wine_data_y - lm_wine_data
> # Calculating error for full set
> dataErr <- sqrt(sum(dYData*dYData))/(length(wine_data_y))
> paste("Full set prediciton error = ",dataErr)

[1] "Full set prediciton error =  0.0200136387039995"

> summary(lm_wine_data)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.69121 -0.45265 -0.03970  0.01544  0.46124  2.17223
```

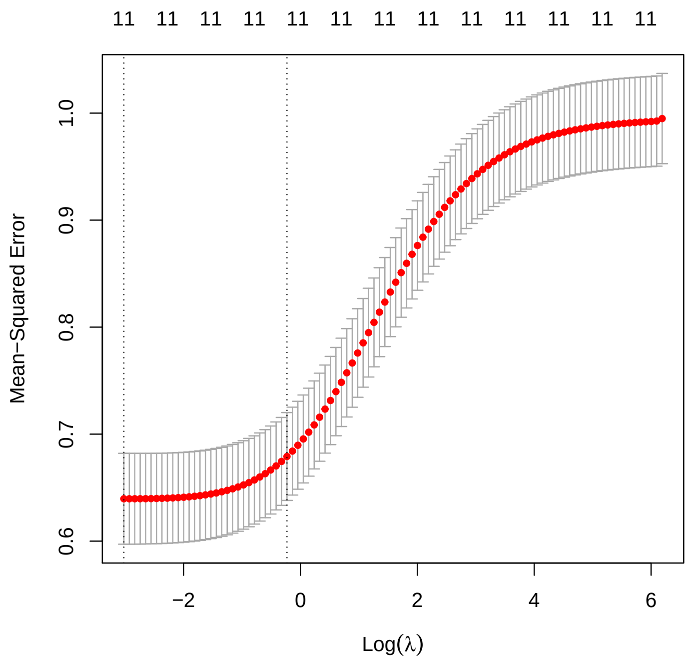
Question 2

2) Perform a ridge regression on the wine quality data set from problem 1 using only the first 1400 observations. Compare the results of applying the ridge regression model to the last 199 observations with the results of applying the ordinary least square model to these observations. Compare the coefficients resulting from the ridge regression with the coefficients that were obtained in problem 1. What conclusions can you make from this comparison?

Answer 2

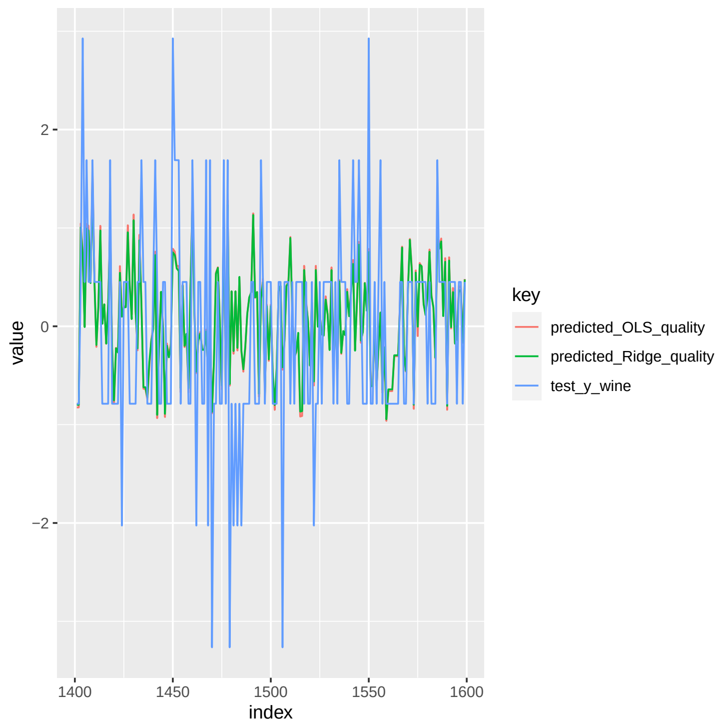
```
> library(glmnet)
> # Train data for Ridge
> ridge_wine_train = scaled_wine_data[1:1400,]
> ridge_x_wine_train = ridge_wine_train[,1:11]
> ridge_y_wine_train = ridge_wine_train[,12]
> # Test data for Ridge (199 last)
> test_wine_ridge = scaled_wine_data[1401:dim(wine_data)[1],]
> test_x_wine = test_wine_ridge[,1:11]
```

```
> test_y_wine = test_wine_ridge[,12]
> # Using glmnet on train data to get a model to predict quality
> cv.out=cv.glmnet(as.matrix(ridge_x_wine_train), ridge_y_wine_train, alpha = 0 )

> plot(cv.out)
```




```
> # Taking minimum lambda to use in next prediction
> bestlambda=cv.out$lambda.min
> bestlambda

[1] 0.04874846

> #Make fair comraison of Error
> ridgeMod=glmnet(as.matrix(ridge_x_wine_train), ridge_y_wine_train, alpha = 0, lambda = bes
> predicted_Ridge_quality= predict(ridgeMod, newx = as.matrix(test_x_wine))[1:dim(test_x_win
> ridge_testErr = sqrt(sum((test_y_wine - predicted_Ridge_quality)^2))/length(predicted_Ridg
> ridge_coef<-coef(ridgeMod)
> paste("Ridge error = ", ridge_testErr)

[1] "Ridge error =  0.0613576627146581"

> paste("Ridge lambda = ", bestlambda, "alpha = 0")

[1] "Ridge lambda =  0.0487484578909308 alpha = 0"
```

```
> # Predicted_OLS vs Predicted_Ridge
> library(ggplot2)
> library(dplyr)
> library(tidyr)
> index = c(1401:dim(wine_data)[1])
> df=data.frame(index,test_y_wine, predicted_OLS_quality, predicted_Ridge_quality)
> dfplot <- df %>% gather(key, value, -index)

> ggplot(dfplot, mapping = aes(x = index, y = value, color = key) ) + geom_line()
```




```
> diff_OLS_Ridge = predicted_OLS_quality - predicted_Ridge_quality
> OLS_coef

        (Intercept)       fixed.acidity    volatile.acidity
         0.01543785          0.05023145         -0.23862339
        citric.acid      residual.sugar           chlorides
        -0.03504472          0.01146886         -0.10638580
free.sulfur.dioxide total.sulfur.dioxide            density
         0.04423963         -0.13961796         -0.03494001
                 pH           sulphates             alcohol
        -0.05580298          0.18328213          0.36639494

> ridge_coef
```

```
12 x 1 sparse Matrix of class "dgCMatrix"
                              s0
(Intercept)           0.01474809
fixed.acidity         0.06597727
volatile.acidity     -0.22457003
citric.acid          -0.01413464
residual.sugar        0.01971215
chlorides            -0.10214615
free.sulfur.dioxide   0.03599691
total.sulfur.dioxide -0.13224343
density              -0.06164116
pH                   -0.03701236
sulphates             0.17689476
alcohol               0.33633856
```

```
> summary(ridge_coef)
```

```
12 x 1 sparse Matrix of class "dgCMatrix", with 12 entries
    i j           x
1   1 1  0.01474809
2   2 1  0.06597727
3   3 1 -0.22457003
4   4 1 -0.01413464
5   5 1  0.01971215
6   6 1 -0.10214615
7   7 1  0.03599691
8   8 1 -0.13224343
9   9 1 -0.06164116
10 10 1 -0.03701236
11 11 1  0.17689476
12 12 1  0.33633856
```

```
> mean(ridge_coef)
```

```
[1] 0.00649333
```

```
> median(ridge_coef)
```

```
[1] 0.000306721
```

```
> summary(OLS_coef)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.238623 -0.068449 -0.011736  0.005053  0.045738  0.366395
```

Question 3

3) This problem uses the Iris Data Set. It only involves the Versicolor and Virginica species (rows 51 through 150). Use cross validated ridge regression to

classify these two species. Create and plot a ROC curve for this classification method.

Answer 3

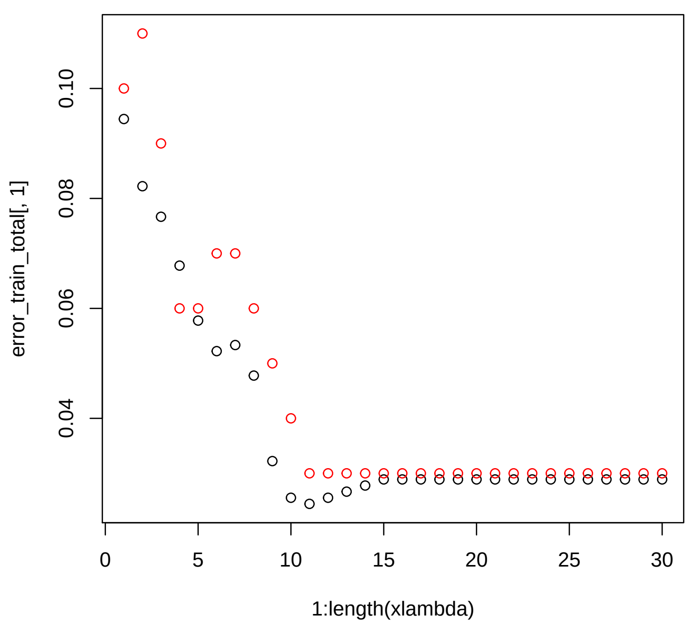
```
> rm(list=ls())
> library(datasets)
> library(dplyr)
> library(MASS)
> library("ridge")
> iris_orig_data = as.data.frame(iris)
> xlambda=rep(0, times = 30)
> for(i in seq(from = 0, to = 29)){
+   #
+   exp <- (+3 -4*(i/20))
+   xlambda[i+1] <- 10^exp
+ }
> iris_data = iris_orig_data[51:150,]
> target = rep(0,100)
> str(iris_data)

'data.frame':        100 obs. of  5 variables:
 $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
 $ Sepal.Width : num  3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
 $ Petal.Length: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
 $ Petal.Width : num  1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 2 2 2 2 2 2 2 2 2 2 ...

> target[iris_data[,5]=="versicolor"] = 1
> target[iris_data[,5]!="versicolor"] = -1
> row.names(iris_data)<-NULL
> iris_data = cbind(iris_data, target)
> set_seed <- function(i) {
+   set.seed(i)
+   if (exists(".Random.seed"))  oldseed <- get(".Random.seed", .GlobalEnv)
+   if (exists(".Random.seed"))  assign(".Random.seed", oldseed, .GlobalEnv)
+ }
> k = 10
> # 10-fold cross valiation is used.
> num_sample = nrow(iris_data)
> set_seed(123)
> iris_data = iris_data[sample(num_sample, num_sample, replace=FALSE),]
> iris_train = iris_data[1:(num_sample*((k-1)/k)),]
> iris_cross = iris_data[1:(num_sample*(1/k)),]
> error_train_total = matrix(0, nrow = length(xlambda), ncol = 1)
> error_cross_total = matrix(0, nrow = length(xlambda), ncol = 1)
```

```
> for(ilambda in 1:length(xlambda)){
+   pick = k #pick kth set
+
+   error_train = 0
+   error_cross = 0
+
+   for(j in 1:k){
+     i_tmp = 1
+     for(i in 1:k){
+
+       #choose training set, and cross validation set
+       if(i == pick){
+         iris_cross = iris_data[((i-1)*num_sample/k+1):(num_sample*(i/k)),]
+       } else {
+         iris_train[((i_tmp-1)*num_sample/k+1):(num_sample*(i_tmp/k)),
+                    ] = iris_data[((i-1)*num_sample/k+1):(num_sample*i/k),]
+         i_tmp = i_tmp + 1
+       }
+     }
+     pick = pick - 1
+     y_iris_train = iris_train[,6]
+     x_iris_train = iris_train[,1:4]
+     yx_iris_train = cbind(x_iris_train, y_iris_train)
+
+     y_iris_cross = iris_cross[,6]
+     x_iris_cross = iris_cross[,1:4]
+
+     iris_model = lm.ridge(y_iris_train~., yx_iris_train, lambda=xlambda[ilambda])
+     A = as.array(iris_model$coef[1:4]/iris_model$scales)
+     X_train = x_iris_train
+     for( i in seq(from = 1, to = ncol(x_iris_train))){
+       X_train[,i] = x_iris_train[,i] - iris_model$xm[i]
+     }
+     X_train=as.matrix(X_train)
+     yh = X_train%*%A + iris_model$ym
+
+     yhP = (yh >= 0.0)
+     yp = (y_iris_train >= 0.0)
+     error_train = error_train + sum(yhP != yp)/(length(y_iris_train)*k*0.00001/0.00001)
+     X_cross = x_iris_cross
+     for( i in seq(from = 1, to = ncol(x_iris_cross))){
+       X_cross[,i] = x_iris_cross[,i] - iris_model$xm[i]
+     }
+     X_cross=as.matrix(X_cross)
+     yh = X_cross%*%A + iris_model$ym
+
```

```
+
+     yhP = (yh >= 0.0)
+     yp = (y_iris_cross >= 0.0)
+
+     error_cross = error_cross + sum(yhP != yp)/(length(y_iris_cross)*k*0.00001/0.00001)
+
+   }
+
+   error_train_total[ilambda,1] = error_train
+   error_cross_total[ilambda,1] = error_cross
+ }
> min_iris_lambda <- xlambda[min(which(min(error_cross_total) == error_cross_total))]
> th_lambda = min(which(min(error_cross_total) == error_cross_total))
> cat(th_lambda, "th lambda", min_iris_lambda, "is optimal.")
```

```
11 th lambda 10 is optimal.
```

```
> plot(1:length(xlambda),error_train_total[,1],
+      ylim=c(min(error_train_total, error_cross_total),
+             max(error_train_total, error_cross_total)))
> points(1:length(xlambda),error_cross_total[,1], col='red')
```



Question 4

4) See if you can improve on regression-based classification of the iris data that we did in class. Classify the iris data set with second degree terms added using a ridge regression. (ie supplement the original 4 attributes x1, x2, x3, and x4 by including the 10 second degree terms (x1*x1, x1*x2, x1*x3, .) for a total of 14 attributes.) Use multiclass to classify the data and then compare the results with the results obtained in class. It is fine to use brute force to add these attributes. For those who are adventurous, investigate the function mutate in the package plyr

Answer 4

```
> rm(list=ls())
> library(datasets)
> library(dplyr)
> iris_orig_data = as.data.frame(iris)
> orig_lm = lm(iris_orig_data$Species~., data = iris_orig_data[1:4])
> # Getting the list of columns for further mutation
> # Sepal.Length Sepal.Width Petal.Length Petal.Width
> columns_for_mutation = attributes(iris_orig_data)$names[
+                                                        1:length(
+                                                          attributes(iris_orig_data)$names
+                                                        ]
> # Adding squared column values
> iris_mutated_data = mutate_at(iris_orig_data,
+                               .vars=columns_for_mutation,
+                               .funs=list(Squared = ~.^2))
> # Adding values multiplied by 2
> iris_mutated_data = mutate_at(iris_mutated_data,
+                               .vars=columns_for_mutation,
+                               .funs=list(Doubled = ~.*2))
> # Adding values of columns 1-2 multiplied by each other
> iris_mutated_data = mutate(iris_mutated_data,
+                            "Sepal.Length_x_Sepal.Width" = Sepal.Length * Sepal.Width)
> # Adding values of columns 3-4 multiplied by each other
> iris_mutated_data = mutate(iris_mutated_data,
+                            "Petal.Length_x_Petal.Width" = Petal.Length * Petal.Width)
> # Reordering data set
> iris_mutated_data_ordered <- iris_mutated_data[,
+                                               c(which(
+                                                      colnames(
+                                                          iris_mutated_data) !="Species"),
+                                                 which(
+                                                      colnames(
```

```
+                                                     iris_mutated_data)=="Species"))
+                                               ]
> # Train data
> ridge_x_iris = iris_mutated_data_ordered[,1:14]
> ridge_y_iris = iris_mutated_data_ordered[,15]
> str(ridge_x_iris)
```

```
'data.frame':	150 obs. of  14 variables:
 $ Sepal.Length             : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width              : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length             : num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width              : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Sepal.Length_Squared     : num  26 24 22.1 21.2 25 ...
 $ Sepal.Width_Squared      : num  12.25 9 10.24 9.61 12.96 ...
 $ Petal.Length_Squared     : num  1.96 1.96 1.69 2.25 1.96 2.89 1.96 2.25 1.96 2.25 ...
 $ Petal.Width_Squared      : num  0.04 0.04 0.04 0.04 0.04 0.16 0.09 0.04 0.04 0.01 ...
 $ Sepal.Length_Doubled     : num  10.2 9.8 9.4 9.2 10 10.8 9.2 10 8.8 9.8 ...
 $ Sepal.Width_Doubled      : num  7 6 6.4 6.2 7.2 7.8 6.8 6.8 5.8 6.2 ...
 $ Petal.Length_Doubled     : num  2.8 2.8 2.6 3 2.8 3.4 2.8 3 2.8 3 ...
 $ Petal.Width_Doubled      : num  0.4 0.4 0.4 0.4 0.4 0.8 0.6 0.4 0.4 0.2 ...
 $ Sepal.Length_x_Sepal.Width: num  17.8 14.7 15 14.3 18 ...
 $ Petal.Length_x_Petal.Width: num  0.28 0.28 0.26 0.3 0.28 0.68 0.42 0.3 0.28 0.15 ...
```

```
> str(ridge_y_iris)
```

```
 Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> summary(ridge_x_iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
 Sepal.Length_Squared Sepal.Width_Squared Petal.Length_Squared
 Min.   :18.49        Min.   : 4.000      Min.   : 1.00       
 1st Qu.:26.01        1st Qu.: 7.840      1st Qu.: 2.56       
 Median :33.64        Median : 9.000      Median :18.93       
 Mean   :34.83        Mean   : 9.536      Mean   :17.22       
 3rd Qu.:40.96        3rd Qu.:10.890      3rd Qu.:26.01       
 Max.   :62.41        Max.   :19.360      Max.   :47.61       
 Petal.Width_Squared Sepal.Length_Doubled Sepal.Width_Doubled
 Min.   :0.010       Min.   : 8.60        Min.   :4.000      
 1st Qu.:0.090       1st Qu.:10.20        1st Qu.:5.600      
 Median :1.690       Median :11.60        Median :6.000      
```

```
 Mean   :2.016       Mean   :11.69        Mean   :6.115
 3rd Qu.:3.240       3rd Qu.:12.80        3rd Qu.:6.600
 Max.   :6.250       Max.   :15.80        Max.   :8.800
 Petal.Length_Doubled Petal.Width_Doubled Sepal.Length_x_Sepal.Width
 Min.   : 2.000       Min.   :0.200       Min.   :10.00
 1st Qu.: 3.200       1st Qu.:0.600       1st Qu.:15.66
 Median : 8.700       Median :2.600       Median :17.66
 Mean   : 7.516       Mean   :2.399       Mean   :17.82
 3rd Qu.:10.200       3rd Qu.:3.600       3rd Qu.:20.32
 Max.   :13.800       Max.   :5.000       Max.   :30.02
 Petal.Length_x_Petal.Width
 Min.   : 0.110
 1st Qu.: 0.420
 Median : 5.615
 Mean   : 5.794
 3rd Qu.: 9.690
 Max.   :15.870
```

```
> summary(ridge_y_iris)
```

```
   setosa versicolor  virginica
       50         50         50
```

```
> # Making a model
> grid=10^seq(10,-2,length=100)
> flowers = c(rep(1,50),rep(2,50), rep(3,50))
> library(data.table)
> library(mltools)
> x=one_hot(as.data.table(iris$Species))
> cv.out=cv.glmnet(
+   as.matrix(ridge_x_iris),
+   x$V1_setosa, alpha = 0,
+   labmda=grid)
> fit <- cv.out$glmnet.fit
> summary(fit)
```

```
          Length Class     Mode
a0         100   -none-    numeric
beta      1400   dgCMatrix S4
df         100   -none-    numeric
dim          2   -none-    numeric
lambda     100   -none-    numeric
dev.ratio  100   -none-    numeric
nulldev      1   -none-    numeric
npasses      1   -none-    numeric
jerr         1   -none-    numeric
offset       1   -none-    logical
```

```
call         5   -none-    call
nobs         1   -none-    numeric

> opt_lambda = cv.out$lambda.min
>
```

Question 5

5) This is a multi-class problem. Consider the Glass Identification Data Set from the UC Irvine Data Repository. The Data is located at the web site: http://archive.ics.uci.edu/ml/datasets/GlassThis problem will only work with building and vehicle window glass (classes 1,2 and 3), so it only uses the first 163 rows of data. (Ignore rows 164 through 214) With this set up this is a three class problem. Use ridge regression to classify this data into the three classes: building windows float processed, building windows non float processed, and vehicle windows float processed