Linear Methods 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

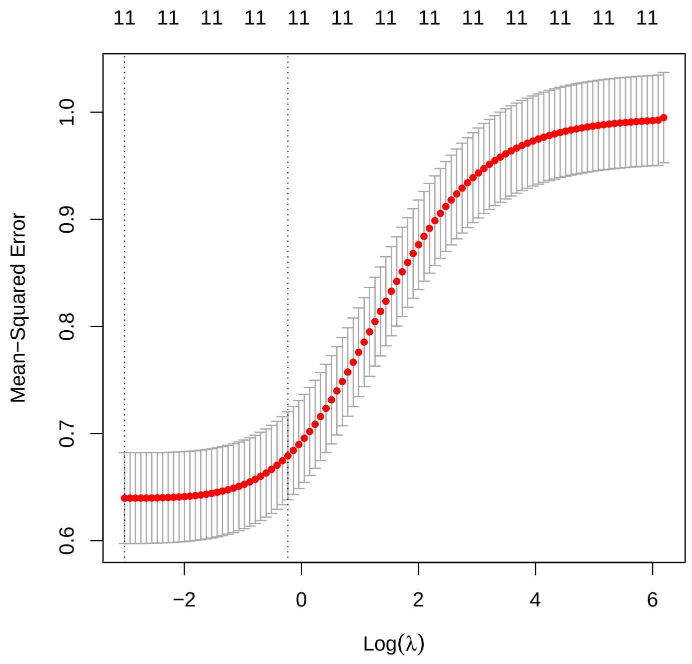
```
> set.seed(pi)
> wine_data<-read.csv("winequality-red.csv",header = TRUE, sep=";")
> # Scaling data before constructing models
> scaled_wine_data<-scale(wine_data)
> # First 1400 as train data
> wine_train<-scaled_wine_data[1:1400,]
> x_wine_train<-wine_train[,1:11]
> y_wine_train<-wine_train[,12]
> # 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?

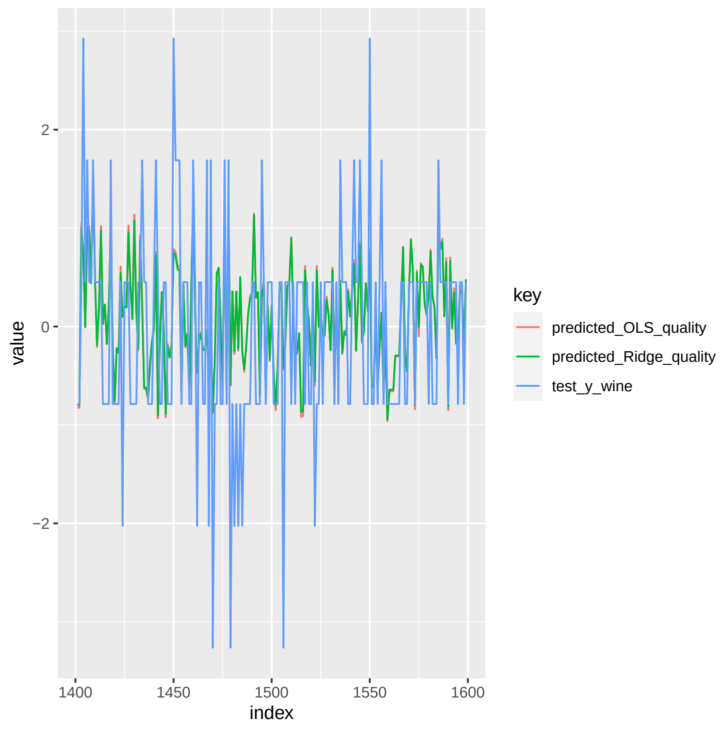
Answer 2

```
> 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_wir
> 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

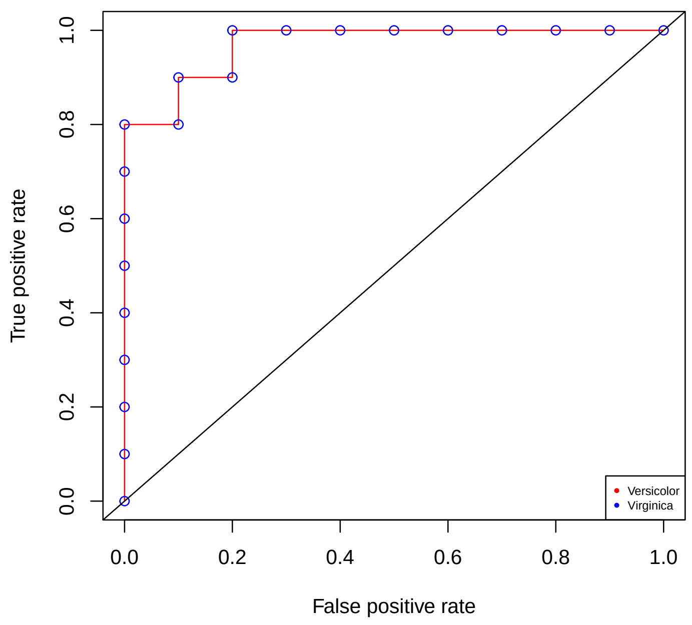
classify these two species. Create and plot a ROC curve for this classification method.

Answer 3

```
> rm(list=ls())
> set.seed(pi)
> library(MASS)
> library(ridge)
> library(glmnet)
> require(rpart)
> library(ROCR)
> iris_orig = as.data.frame(iris)
> iris_data<-iris_orig[51:150,]
> Y1 <- c(rep(-1,100))
> Y1
```

```
  [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
 [26] -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
 [51] -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
 [76] -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
```

```
> Y2 <- Y1
> Y1[1:50] <- 1
> Y2[51:100] <- 1
> I <- 1:100
> target <- cbind(Y1,Y2)
> X <- iris_data[,-5]
> x_trainIn1<-0.0
> x_trainOut1<-0.0
> x_trainIn2<-0.0
> x_trainOut2<-0.0
> nxval=5
> ntol=nrow(iris_data)
> y_trainIn<-matrix(nrow = (ntol - ntol/nxval), ncol = 2)
> y_trainOut<-matrix(nrow = ntol/nxval, ncol = 2)
> testErr <- matrix(nrow = nxval, ncol = 2)
> lambda <- matrix(nrow = nxval, ncol = 2)
> prolambda <- c(rep(0,2))
> for(ident in seq(1,2)){
+     grid=10^seq(10,-2,length=100)
+     trainErr <- 0.0
+     bestlam <-0.0
+
+     for(ixval in seq(from = 1, to = nxval)){
+         Iout <- which(I%%nxval== ixval - 1)
+         Xin <- X[-Iout,]
+         Xout <- X[Iout,]
+         Yin <- target[-Iout,ident]
+         Yout <- target[Iout,ident]
+         dataIn <- cbind(Xin,Yin)
+
+         if(ident==1){
+             y_trainIn[, 1]<-target[-Iout,ident]
+             y_trainOut[, 1]<-target[Iout,ident]
+             x_trainIn1<-X[-Iout,]
+             x_trainOut1<-X[Iout,]
+             }
+         else if(ident==2) {
+             y_trainIn[, 2]<-target[-Iout,ident]
+             y_trainOut[, 2] <- target[Iout,ident]
+             x_trainIn2<-X[-Iout,]
+             x_trainOut2<-X[Iout,]
+             }
+
+         ## to get best lambda
+         cv.out=cv.glmnet(as.matrix(Xin),as.matrix(Yin),alpha=0,lambda=grid)
+         bestlam=cv.out$lambda.min
+
+         #
+         ridgeMod =glmnet(as.matrix(Xin), as.matrix(Yin),alpha=0,lambda=bestlam)
+         yEst <- predict(ridgeMod, newx = as.matrix(Xout))
+
+         testErr[ixval, ident] <-sqrt( sum((Yout-yEst)^2))/length(yEst)
+         lambda[ixval, ident]<-bestlam
+
+         }
+
+     MintestErr<-apply(testErr, 2, min)
+     MintestErr
+     index<- which(min(testErr[, ident])==testErr[, ident])
+     print(index)
+     prolambda[ident]<- lambda[index, ident]
+
+ }

[1] 2
[1] 2

> xlambda1 <- prolambda[1]
> Xin1 <- x_trainIn1
> Yin1 <-y_trainIn[, 1]
> dataIn1 <- cbind(Xin1,Yin1)
> ridgeMod1 =glmnet(as.matrix(Xin1), as.matrix(Yin1),alpha=0,lambda=xlambda1)
> pred_y1 <- predict(ridgeMod1, newx = as.matrix(x_trainOut1))
> testErr1 <-sqrt( sum((y_trainOut[, 1] - pred_y1)^2))/length(pred_y1)
> testErr1

[1] 0.1430037

> pred1<-prediction(pred_y1, as.matrix(y_trainOut[, 1]))
> perf1<-performance(pred1,"tpr","fpr")
> xlambda2 <- prolambda[2]
> Xin2 <- x_trainIn2
> Yin2 <- y_trainIn[, 2]
> dataIn2 <- cbind(Xin2,Yin2)
> ridgeMod2 =glmnet(as.matrix(Xin2), as.matrix(Yin2),alpha=0,lambda=xlambda2)
> pred_y2 <- predict(ridgeMod2, newx = as.matrix(x_trainOut2))
> testErr2 <-sqrt( sum((y_trainOut[, 2] -pred_y2)^2))/length(pred_y2)
> testErr2

[1] 0.1430037

> pred2<-prediction(pred_y2, as.matrix(y_trainOut[, 2]))
> perf2<-performance(pred2,"tpr","fpr")
> x.values= data.frame(perf2@x.values)
> y.values= data.frame(perf2@y.values)
>
>
>

> plot(perf1,col="red", type = "l")
> points(as.matrix(x.values), as.matrix(y.values),col="blue", type = "p")
> abline(0,1)
> legend("bottomright", c("Versicolor","Virginica"), cex = 0.6,
+        pch = c(20,20), col = c("red", "blue"))# lty = 1:2,
```

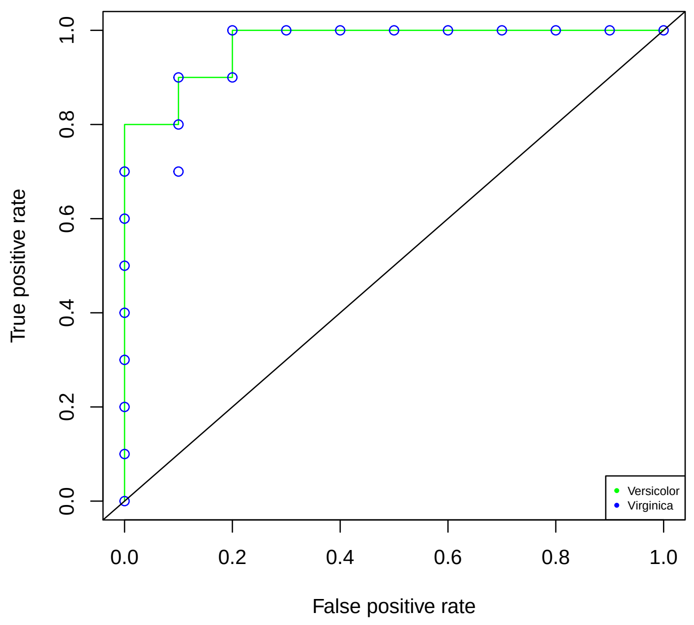


Question 4

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)
> iris_test<-mutate(iris_orig_data, X5 =Sepal.Length^2, X6= Sepal.Length* Sepal.Width,
+                      X7=Sepal.Length*Petal.Length, X8=Sepal.Length*Petal.Width,
+                      X9 =Sepal.Width^2, X10=Sepal.Width*Petal.Length,
+                     X11=Sepal.Width*Petal.Width, X12= Petal.Length^2,
+                     X13=Petal.Length*Petal.Width, X14=Petal.Width^2)
> iris_data<-iris_orig_data[51:150,]
> iris_trans1<-iris_test[, -5]
> iris_trans2<-iris_test[, 5]
> iris_trans<-cbind(iris_trans1, iris_trans2)
> iris_data<-iris_trans[51:150,]
> Y1 <- c(rep(-1,100))
> Y2 <- Y1
> Y1[1:50] <- 1
> Y2[51:100] <- 1
> I <- 1:100
> target <- cbind(Y1,Y2)
> head(target)

     Y1 Y2
[1,]  1 -1
[2,]  1 -1
[3,]  1 -1
[4,]  1 -1
[5,]  1 -1
[6,]  1 -1

> X <- iris_data[,-15]
> x_trainIn1<-0.0
> x_trainOut1<-0.0
> x_trainIn2<-0.0
> x_trainOut2<-0.0
> nxval=5
> ntol=nrow(iris_data)
> y_trainIn<-matrix(nrow = (ntol - ntol/nxval), ncol = 2)
> y_trainOut<-matrix(nrow = ntol/nxval, ncol = 2)
> testErr <- matrix(nrow = nxval, ncol = 2)
> lambda <- matrix(nrow = nxval, ncol = 2)
> prolambda <- c(rep(0,2))
> for(ident in seq(1,2)){
+     grid=10^seq(10,-2,length=100)
+     trainErr <- 0.0
+     bestlam <-0.0
+
+     for(ixval in seq(from = 1, to = nxval)){
+         Iout <- which(I%%nxval== ixval - 1)
+         Xin <- X[-Iout,]
+         Xout <- X[Iout,]
+         Yin <- target[-Iout,ident]
+         Yout <- target[Iout,ident]
+         dataIn <- cbind(Xin,Yin)
+
+         if(ident==1){
+             y_trainIn[, 1]<-target[-Iout,ident]
+             y_trainOut[, 1]<-target[Iout,ident]
+             x_trainIn1<-X[-Iout,]
+             x_trainOut1<-X[Iout,]
+             }
+
+         else if(ident==2) {
+             y_trainIn[, 2]<-target[-Iout,ident]
+             y_trainOut[, 2] <- target[Iout,ident]
+             x_trainIn2<-X[-Iout,]
+             x_trainOut2<-X[Iout,]
+             }
+
+
+     cv.out=cv.glmnet(as.matrix(Xin),as.matrix(Yin),alpha=0,lambda=grid)
+     bestlam=cv.out$lambda.min
+
+
+     ridgeMod =glmnet(as.matrix(Xin), as.matrix(Yin),alpha=0,lambda=bestlam)
+     yEst <- predict(ridgeMod, newx = as.matrix(Xout))
+
+     testErr[ixval, ident] <-sqrt( sum((Yout-yEst)^2))/length(yEst)
+     lambda[ixval, ident]<-bestlam
+
+ }
+
+   testErr
+   MintestErr<-apply(testErr, 2, min)
+   MintestErr
+   index<- which(min(testErr[, ident])==testErr[, ident])
+   print(index)
+   prolambda[ident]<- lambda[index, ident]
+   library(ROCR)
+
+
+
+
+ xlambda1 <- prolambda[1]
+ Xin1 <- x_trainIn1
+ Yin1 <-y_trainIn[, 1]
+ dataIn1 <- cbind(Xin1,Yin1)
+ ridgeMod1 =glmnet(as.matrix(Xin1), as.matrix(Yin1),alpha=0,lambda=xlambda1)
+
+ pred_y1 <- predict(ridgeMod1, newx = as.matrix(x_trainOut1))
+ testErr1 <-sqrt( sum((y_trainOut[, 1] - pred_y1)^2))/length(pred_y1)
+ testErr1
+
+ }

[1] 2
[1] 2

> pred1<-prediction(pred_y1, as.matrix(y_trainOut[, 1]))
> perf1<-performance(pred1,"tpr","fpr")
> xlambda2 <- prolambda[2]
> Xin2 <- x_trainIn2
> Yin2 <- y_trainIn[, 2]
> dataIn2 <- cbind(Xin2,Yin2)
> ridgeMod2 =glmnet(as.matrix(Xin2), as.matrix(Yin2),alpha=0,lambda=xlambda2)
> pred_y2 <- predict(ridgeMod2, newx = as.matrix(x_trainOut2))
> testErr2 <-sqrt( sum((y_trainOut[, 2] -pred_y2)^2))/length(pred_y2)
> testErr2

[1] 0.1417557

> pred2<-prediction(pred_y2, as.matrix(y_trainOut[, 2]))
> perf2<-performance(pred2,"tpr","fpr")
> x.values= data.frame(perf2@x.values)
> y.values= data.frame(perf2@y.values)

> plot(perf1,col="green", type = "l")
> points(as.matrix(x.values), as.matrix(y.values),col="blue", type = "p")
> abline(0,1)
> legend("bottomright", c("Versicolor","Virginica"), cex = 0.6,
+        pch = c(20,20), col = c("green", "blue"))# lty = 1:2,
```



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

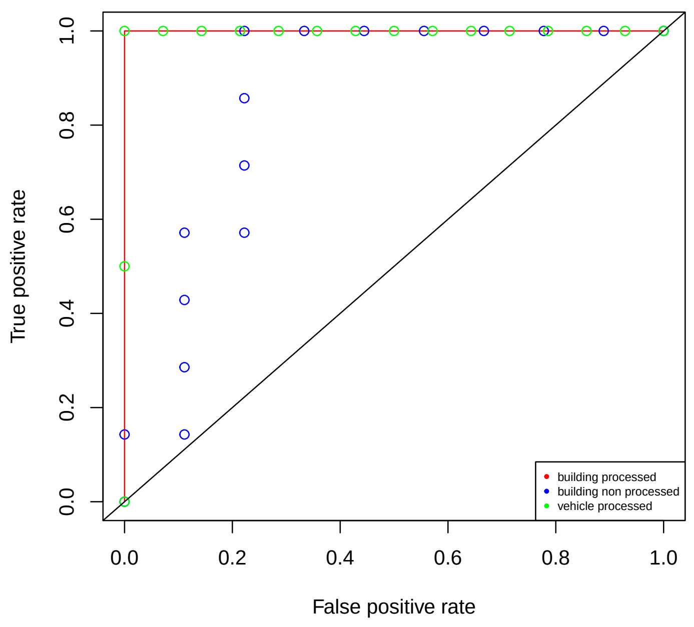
Answer 5

```
> glass <- read.csv("glass.txt",sep = ",", header = FALSE)
> names(glass)

 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"

> glass_data<-glass[1:163,]
> Y1 <- c(rep(-1,163))
> Y1
```

```
  [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
 [26] -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
 [51] -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
 [76] -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
[101] -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
[126] -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
```

```
> Y2 <- Y1
> Y3 <-Y1
> Y1[1:70] <- 1        #Y1 = building windows float processed
> Y2[71:146] <- 1      #Y2 = building windows non float processed
> Y3[147:163] <- 1     #Y3 = vehicle windows float processed
> I <- 1:163
> target <-cbind(Y1, Y2, Y3)
> X <- glass_data[,-11]
> x_trainIn<-0.0
> x_trainOut<-0.0
> nxval=10
> ntol=nrow(glass_data)
> y_trainIn<-matrix(nrow = 147, ncol = 3)
> y_trainOut<-matrix(nrow = 16, ncol = 3)
> testErr <- matrix(nrow = 10, ncol = 3)
> lambda <- matrix(nrow = 10, ncol = 3)
> y_trainIn1<-0.0
> y_trainIn2<-0.0
> y_trainIn3<-0.0
> y_trainOut1<-0.0
> y_trainOut2<-0.0
> y_trainOut3<-0.0
> x_trainIn1<-0.0
> x_trainIn2<-0.0
> x_trainIn3<-0.0
> x_trainOut1<-0.0
> x_trainOut2<-0.0
> x_trainOut3<-0.0
> prolambda <- c(rep(0,3))
> index<-matrix(nrow = 3, ncol=3)
> for(ident in seq(1,3)){
+     grid=10^seq(10,-2,length=50)
+     trainErr <- 0.0
+     bestlam <-0.0
+
+     for(ixval in seq(from = 1, to = 10)){
+         Iout <- which(I%%10== ixval - 1)
+
+         Xin <- X[-Iout,]
+         Xout <- X[Iout,]
+         Yin <- target[-Iout,ident]
+         Yout <- target[Iout,ident]
+         dataIn <- cbind(Xin,Yin)
+         x_trainIn<-Xin
+         x_trainOut<-Xout
+
+
+         cv.out=cv.glmnet(as.matrix(Xin),as.matrix(Yin),alpha=0,lambda=grid)
+         bestlam=cv.out$lambda.min
+
+
+      ridgeMod =glmnet(as.matrix(Xin), as.matrix(Yin),alpha=0,lambda=bestlam)
+      yEst <- predict(ridgeMod, newx = as.matrix(Xout))
+
+      testErr[ixval, ident] <-sqrt( sum((Yout-yEst)^2))/length(yEst)
+      lambda[ixval, ident]<-bestlam
+
+      if(ident==1){
+          y_trainIn1<-target[-Iout, ident]
+          y_trainOut1<-target[Iout, ident]
+          x_trainIn1<-X[-Iout,]
+          x_trainOut1<-X[Iout,]
+          }
+   else if(ident==2) {
+       y_trainIn2<-target[-Iout, ident]
+       y_trainOut2<-target[Iout, ident]
+       x_trainIn2<-X[-Iout,]
+       x_trainOut2<-X[Iout,]
+       }
+
+   else if(ident==3){
+       y_trainIn3<-target[-Iout, ident]
+       y_trainOut3<-target[Iout, ident]
+       x_trainIn3<-X[-Iout,]
+       x_trainOut3<-X[Iout,]
+     }
+   }
+
+   testErr
+   MintestErr<-apply(testErr, 2, min)
+   MintestErr
+   index<- min(which(min(testErr[, ident])==testErr[, ident]))
+
+
+   index
+
+
+   prolambda[ident]<- lambda[index, ident]
+
+
+  }
> prolambda

[1] 0.16768329 0.29470517 0.01757511

> library(ROCR)
> xlambda1 <- prolambda[1]
> Xin1 <- x_trainIn1
> Yin1 <-y_trainIn1
> ridgeMod1 =glmnet(as.matrix(Xin1), as.matrix(Yin1),alpha=0,lambda=xlambda1)
> pred_y1 <- predict(ridgeMod1, newx = as.matrix(x_trainOut1))
> testErr1 <-sqrt( sum((y_trainOut1 - pred_y1)^2))/length(pred_y1)
> testErr1

[1] 0.106027

> pred1<-prediction(pred_y1, as.matrix(y_trainOut1))
> perf1<-performance(pred1,"tpr","fpr")
> xlambda2 <- prolambda[2]
> Xin2 <-x_trainIn2
> Yin2 <-y_trainIn2
> ridgeMod2 =glmnet(as.matrix(Xin2), as.matrix(Yin2),alpha=0,lambda=xlambda2)
> pred_y2 <- predict(ridgeMod2, newx = as.matrix(x_trainOut2))
> testErr2 <-sqrt( sum((y_trainOut2 - pred_y2)^2))/length(pred_y2)
> testErr2

[1] 0.1935913

> pred2<-prediction(pred_y2, as.matrix(y_trainOut2))
> perf2<-performance(pred2,"tpr","fpr")
> x2.values= data.frame(perf2@x.values)
> y2.values= data.frame(perf2@y.values)
> xlambda3 <- prolambda[3]
> Xin3 <-x_trainIn3
> Yin3 <-y_trainIn3
> ridgeMod3 =glmnet(as.matrix(Xin3), as.matrix(Yin3),alpha=0,lambda=xlambda3)
> pred_y3 <- predict(ridgeMod3, newx = as.matrix(x_trainOut3))
> testErr3 <-sqrt( sum((y_trainOut3 - pred_y3)^2))/length(pred_y3)
> testErr3

[1] 0.1248874
```

```
> pred3<-prediction(pred_y3, as.matrix(y_trainOut3))
> perf3<-performance(pred3,"tpr","fpr")
> x3.values= data.frame(perf3@x.values)
> y3.values= data.frame(perf3@y.values)
> plot(perf1,col="red", type = "l")
> points(as.matrix(x2.values), as.matrix(y2.values),col="blue", type = "p")
> points(as.matrix(x3.values), as.matrix(y3.values),col="green", type = "p")
> abline(0,1)
> legend("bottomright", c("building processed","building non processed","vehicle processed"
+         pch = c(20,20), col = c("red", "blue", "green"))# lty = 1:2,
```


True positive rate
False positive rate
building processed
building non processed
vehicle processed