COMPUTATIONAL METHODS IN STATISTICS
Final Project
REGRESSION ANALYSIS

April 30, 2018

Author: Bita Nezamdoust
Student ID: 002332793
Instructor: Dr. Xin Qi

Georgia State University
Department of Mathematics and Statistics

Contents

Introduction 2
Section 1: Linear Regression Analysis 2
    Data Description 2
    Regression Models 4
Section 2: Logistic Regression Analysis 7
    Data Description 7
    Classification Models 8
Conclusion 10
References 11
Appendix 12

INTRODUCTION

The present project deals with regression analysis and data classification as two separate tasks that are arranged in two distinct sections. The first section is composed of analysis of *Parkinsons Telemonitoring Data Set* and is aimed at predicting two desired response variables, based on sixteen predictor variables that have been collected. The second section deals with a classification problem on the *Human Activity Recognition Using smartphones Data Set* and builds a classification model for the desired response variable. All three models will be tested against appropriate testing data and best models with highest rate of accuracy are selected, among several model-fitting techniques that have been employed.

SECTION 1: LINEAR REGRESSION ANALYSIS

This section considers *Parkinsons Telemonitoring Data Set* to predict two linear regression models for two separate response variable, namely *motor UPDRS* and *total UPDRS*. The goal is to find the best fitting model that provides the lowest prediction error and hence, the highest rate of prediction accuracy.

Data Description

The dataset was created at the University of Oxford, in collaboration with 10 medical centers in the US and Intel Corporation who developed the telemonitoring device to record the speech signals. The dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring.[1]

The data consists of twenty-two variables, among which variables *motor_UPDRS* and *total_UPDRS* are the two response variables. The first four variables represent subject ID, age, gender, and test time, and the last sixteen variables are biomedical voice measures which are those of interest to form the predictor variables of the regression model that is going to be built. The data is split into a training set with 1000 observations and a test set with 4875 observations for further investigation of the model's accuracy.

The predictor variables appear to be strongly correlated, as seen in the provided figures. Fig. 1 presents the correlation between every two response variables. The collection shown by the pie charts represents the strength of the linear relationship between the variables. The associated correlation coefficients are provided on the left side figure. The strong upward and downward correlation of some of the predictors is apparent in the scatter plots provided

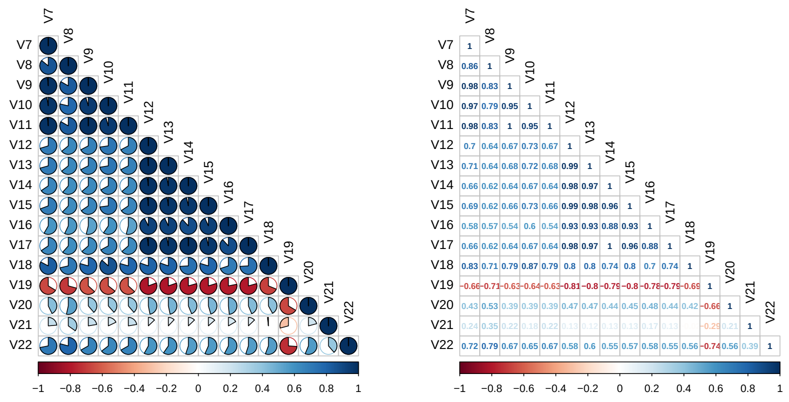

Figure 1: Multicollinearity between the predictor variables

in Fig. 2. The multicollinearity that exists between the predictors could impact the prediction based on these variables negatively, which calls for appropriate measures to be taken to tackle the issue. I will address the issue in the model-fitting section.

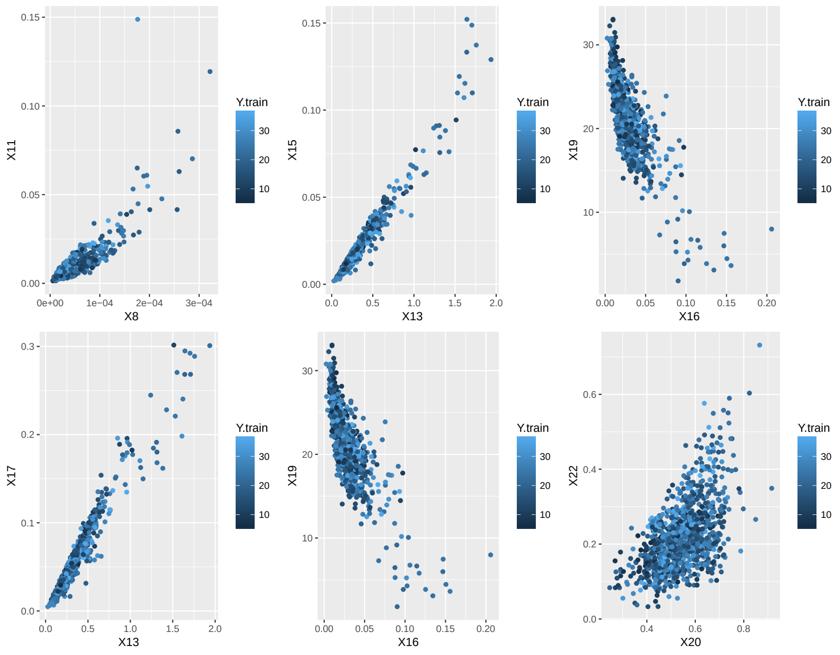

Figure 2: The scatter plot of some of the correlated predictor variables

Regression Models

The first response variable to consider is *motor_UPDRS*. Initially a *Usual Linear Regression Model* is employed to fit a regression line to the data and create the prediction line. The obtained model's performance is tested on the testing data set. As the first column in the summary table below shows, the mean square error, a.k.a the prediction error, from the linear regression fit is equal to 0.069[1], which means the model is able to predict the *motor_UPDRS* variable with 93% accuracy. While the obtained accuracy level is relatively good, further models are employed to reach higher accuracy, especially as the predictor variables are quite strongly correlated and can impact the prediction results, as has been discussed.

Linear Regression Prediction Error (motor UPDRS)			
Linear Regression	Ridge Regression	Lasso Regression	Ridge Sqrt transform
0.069	0.055	0.056	0.055

To tackle the issue of multicollinearity in multiple regression analysis, two of the common regularity methods to utilize are *Ridge Regression* and *Lasso Regression*. Strong correlation between predictor variables can lead to large variation of the estimated parameters in the model. By adding a degree of bias as penalty to the regression estimates, the Ridge and Lasso models attempt to restrict the range of the parameters and so reduce the standard error. As the table indicates, the two models have improved the accuracy by about 2% as the Ridge penalty results in the an error of 0.055 and Lasso penalty 0.056.

Another way to deal with large number of variables and possible multicollinearity is to use dimension reduction methods that reduce the number of parameters and can improve the model accuracy. *Principal Component Analysis (PCA)* is applied as the variable selection method. 3 The new resulting variables (a.k.a PC's) were used to refit the previous models. The accuracy improved only slightly, as the mean square errors were 0.0546 and 0.0540, respectively for Ridge and Lasso fit. Additionally, A square root data transformation was also applied to the analysis as shown in the table, but it was observed that it did not make a significant difference in the accuracy of prediction.

[1]The data set of the present section has been *normalized* for more precision and the ease of interpretation of the obtained mean square errors

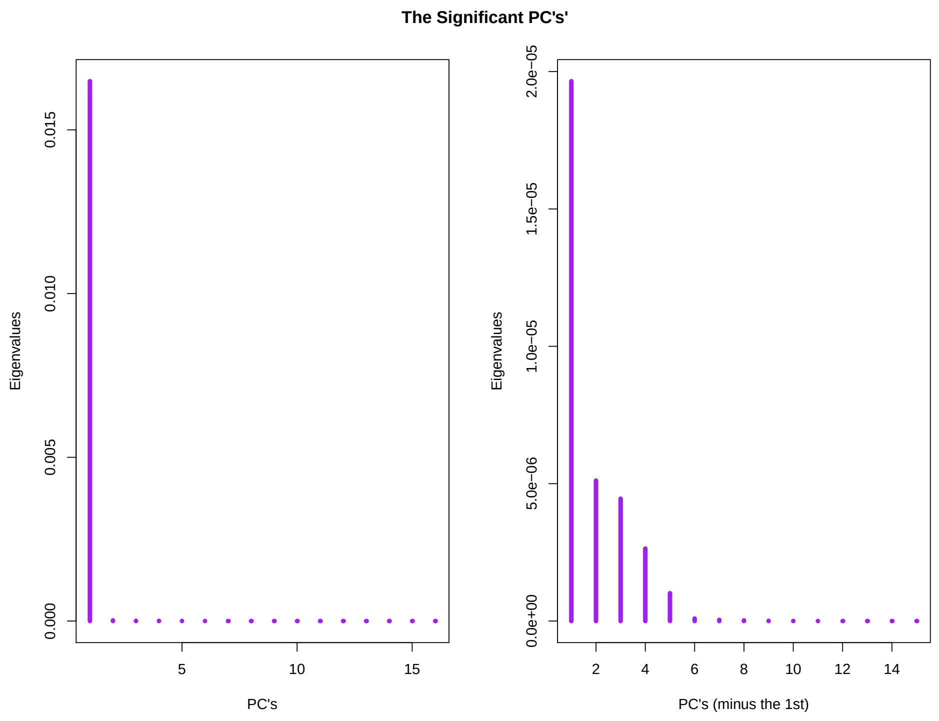

Figure 3: The amount of variability in the data explained by the PC's

Linear Regression Prediction Error (motor UPDRS)- PCA applied	
Ridge Regression	Lasso Regression
0.0546	0.0540

Therefore, the best model for the prediction of motor UPDRS is **Linear Regression with Lasso Penalty - PCA applied** with the prediction error of 0.0540 and %95 level of accuracy. The tuning parameter of lambda is "lambda.min = 0.00187" chosen by 10-fold cross-validation and the regression coefficients are presented in the table below. It is evident from the table that further variable selection has also been accomplished by the Lasso method, as the coefficients corresponding to the second and third PC's are zero.

Intercept	Coefficients				
0.67986900	-0.04557849	0	0	5.58207043	9.26479508

The second response variable for which a linear model is built here is *total_UPDRS*. The same procedure as above is employed with the same predictor variables and a different response, and the results are summarized in the two tables below. Relatively similar to the results for *motor_UPDRS*, the prediction accuracy for *total_UPDRS* improved when the Ridge and Lasso penalties were added to the usual linear regression model, as the mean square error improved from 0.058 to 0.055 for both methods. Another observation is that the prediction improved slightly, i.e. by 0.001, when the square root transformation was applied, unlike for the previous response variable where it remained unchanged.

Linear Regression Prediction Error (total UPDRS)			
Linear Regression	Ridge Regression	Lasso Regression	Ridge Sqrt transform
0.058	0.055	0.055	0.054

Linear Regression Prediction Error (total UPDRS) - PCA applied	
Ridge Regression	Lasso Regression
0.0477	0.0477

And finally, also similar to the first desired response variable, the best model for the prediction of total UPDRS is obtained by **Linear Regression with Lasso Penalty - PCA applied** with the prediction error of 0.0477 and the accuracy level of %95. The tuning parameter of lambda is "lambda.min = 0.001178" chosen by 10-fold cross-validation and the regression coefficients are presented in the table below. Variable selection is also accomplished by the Lasso method as the coefficients corresponding to the second and third PC's are zero.

Intercept	Coefficients				
0.71780993	-0.06315695	0	0	6.65172809	10.65443339

SECTION 2: LOGISTIC REGRESSION ANALYSIS

In this section, a classification rule is build for predicting *human activity* - imited to six physical activities- based on the information recorded in a smartphone they were wearing while doing the activity. The *Human Activity Recognition Using smartphones Data Set* is used and several classification methods are employed to find the best model with the lowest misclassification rate that best fits the data.

Data Description

The data set has been built from the recordings of 30 subjects performing daily activities while carrying a waist-mounted smartphone with embedded inertial sensors. The six activities ans hence, the six classes included WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING. Using its embedded accelerometer and gyroscope, they captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. [2]

There are 561 predictor variables and six classes of human activity as named above in the response variable. While collecting the data, the dataset was randomly split into two sets, where 70% of the volunteers were selected for generating the training data and 30% the test data. In the present report, a subsection of the original data has been used, with 300 observations in the training set and 300 observations in the test set.

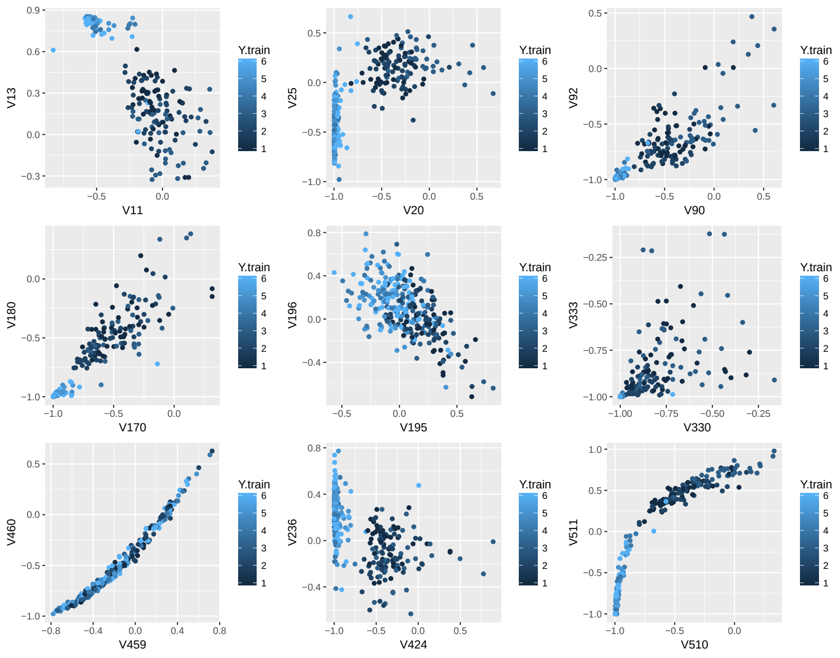

Figure 4: The amount of variability in the data explained by the PC’s

While there is multicollinearity among some of the predictor variables as can be seen in 4, the majority of the predictors do not seem to be very correlated according to 5, in which the correlation between a larger section of the data has been depicted.

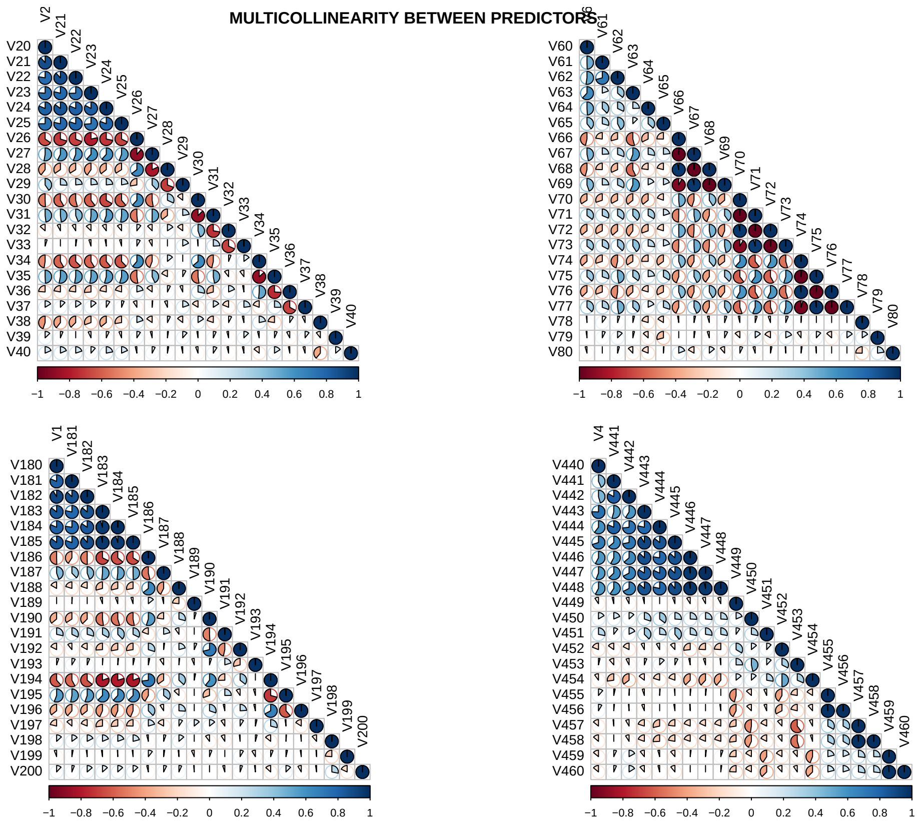

Figure 5: The amount of variability in the data explained by the PC's

Prediction Models

The goal is to build a classification model with highest accuracy level to predict one of the six pre-determined human activities based on the collected information. As before, the models' performances are tested on the test set from the data and their misclassification rates are computed and presented in following tables for comparison.

The first column shows the misclassification error resulted from *Usual Logistic Regression* that is equal to 0.09. That means the model is able to offer classification with %91 accuracy. The error becomes larger when *Ridge* and *Lasso Penalty* are applied, which could be due to the weak multicollinearity of the predictors.

Misclassification Error Comparison						
Logistic Regression	Ridge Regression	Lasso Regression	LDA	QDA	RDA	Tree
0.09	0.15	0.15	0.21	0.21	0.06	0.21

A number of further classification techniques have been employed in this study to achieve the best accuracy that will be briefly introduced here.

Three well-known discriminant analysis techniques include *Linear Discriminant Analysis (LDA*, *Quadratic Discriminant Analysis (QDA)* and *Regularized Discriminant Analysis (RDA*. The resulting models are built on the basis of *the discriminant functions* and *the Bayes Classification Rule*, that is also known as the "optimal classification rule". The three methods have been employed in this study and misclassification errors obtained are 0.21, 0.21, and 0.06, for LDA, QDA and RDA, respectively. It is apparent that the RDA error is significantly smaller and seems to offer the best classification accuracy so far obtained.

Misclassification Error Comparison			
SVM (Linear)	SVM (Polynomial)	SVM (Radial)	SVM (Sigmoid)
0.09	0.08	0.08	0.1
cost = 0.05	C = 1, d =2, γ = 0.0005, γ_0 = 10	C = 5, γ = 0.0005	C = 5, γ = 0.0005, γ_0 = 0

Support Vector Machine (SVM is another linear classification method that is designed to make optimal decision boundaries. The method uses the concept of *margin*, defined as the shortest distance from all the observation points to the linear decision boundary, and attempts to identify the classification rule that maximizes the margin, so that the probability of faulty classification due to random fluctuation of the data is minimized. SVM is computed for four types of *kernel functions* as named in the table above and the misclassification errors for the four models include 0.09 for Linear SVM, 0.08 for Polynomial SVM, 0.08 for Radial SVM and 0.1 for Sigmoid SVM. The values to the tuning parameters for each model could as well be found in the table.

Lastly, a non-linear classification method have also been employed here that is the *Classification Tree*. The tree uses a criteria such as the misclassification rate to select one variable and then start splitting the values and assigning class labels. The misclassification rate for the tree is 0.21 which in comparison cannot compete with the other rates computed before it.

Therefore, the best model for the prediction of the human activity using the smatrphone data is obtained by **Regularized Discriminant Analysis (RDA)** with the prediction error of 0.06 and %96 rate of accuracy. The tuning parameters for the model have been selected by 10-fold cross-validation that include the optimum *alpha* equal to 0.99 and the optimal *delta* equal to 0. The model summary tables are provided below.

$nonzero										
	delta = 0	0.333	0.667	1	1.333	1.667	2	2.333	2.667	3
alpha = 0	561	234	51	1	0	0	0	0	0	0
0.11	561	174	28	5	1	0	0	0	0	0
0.22	561	162	25	5	1	0	0	0	0	0
0.330	561	162	26	6	5	1	0	0	0	0
0.44	561	179	28	7	5	1	0	0	0	0
0.55	561	195	34	12	6	5	1	1	1	0
0.66	561	220	58	19	8	6	6	5	1	1
0.77	561	288	98	39	17	10	6	6	6	5
0.88	561	431	218	100	49	31	19	12	9	7

Conclusion

Three models were built for three response variables, the first two being Linear regression models for two continuous response variables and the third, a classification model for a categorical response variable containing six classes. The best final models include *Linear Regression with Lasso Penalty - PCA applied* for the first two and *Regularized Discriminant Analysis Model* for the last response, with %95, %95 and %96 accuracy levels, respectively. To sum up, it could be stated that in this study, both regularity methods such as Ridge and Lasso penalties and dimension reduction methods such as PCA have been proven to make noteworthy improvements in the accuracy levels of the prediction models, and with the selection of appropriate tuning parameters in the models, best possible accuracy could be achieved.

REFERENCES

[1] A Tsanas, MA Little, PE McSharry, LO Ramig (2009) 'Accurate telemonitoring of ParkinsonâĂŹs disease progression by non-invasive speech tests', IEEE Transactions on Biomedical Engineering [online] available at:
http://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.[online] available at:
http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

APPENDIX

In the appendix section, two bodies of codes can be found that belong to the two sections of the study as was discussed.

```
#######################################
#' Project 2
#' @author Bita Nezamdoust
#######################################

install.packages("scales")
install.packages("corrplot")
install.packages("gridExtra")
install.packages("glmnet")
#################TASK 1###################
#Read Rdata file
dir = "Work/Study_2018/Stat 8670: Computational Methods in
Statistics/Project_2/Part1:LinearReg/project_2_data_1.RData"
get(load(dir))
ls()
str(test)
#'data.frame':  4875 obs. of  22 variables:
names(test)
# [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11" "V12" "V13" "V14"
# [15] "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22"



#install.packages("scales")
library(scales)
#' want to rescale the data so that they're all between 0 and 1 (normalized)
#' and the interpretibility of the prediction errors is easier

X.train = rescale(as.matrix(train[,7:22]))
X.test = rescale(as.matrix(test[,7:22]))
Y.train = rescale(train[,5])
Y.test = rescale(test[ ,5])


#Multicollinearity plot
#install.packages("corrplot")
library(corrplot)
M<-cor(X.train)
head(round(M,2))

par(mfrow = c(1, 2))
corrplot(M, method="pie", type = "lower", tl.col = "black")
corrplot(M, method="number", type = "lower", number.cex = .7, tl.col = "black")
#title("Multicollinearity between Predictors", outer = TRUE, line = -3, cex = 5)
?"corrplot"

#' Interpretation: Seems that the predictor variables are from moderately to very
#' strongly correlated. That mean strong multicollinearity exists among the predictor
#'  variables which will lead to high prediction error,
#' unless we control for it.
#' Also see below:

cor(train$V13, train$V15)
# [1] 0.9781401
cor(train$V16, train$V19)
# [1] -0.7805525
cor(train$V8, train$V11)
# [1] 0.8326512

#Correlation plot
#install.packages("gridExtra")
library(gridExtra)
library(ggplot2)
frame.for.plot = data.frame(X.train)
p1 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V8, y = X.train$V11, color = Y.train))
p2 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V13, y = X.train$V15, color = Y.train))
p3 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V16, y = X.train$V19, color = Y.train))
p4 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V13, y = X.train$V17, color = Y.train))
p5 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V16, y = X.train$V19, color = Y.train))
p6 = ggplot(data = frame.for.plot) +
  geom_point(mapping = aes(x = X.train$V20, y = X.train$V22, color = Y.train))

grid.arrange(p1+ labs(x = "X8", y = "X11"),p2+ labs(x = "X13", y = "X15")
             ,p3+ labs(x = "X16", y = "X19"),p4+ labs(x = "X13", y = "X17")
             ,p5+ labs(x = "X16", y = "X19"),p6+ labs(x = "X20", y = "X22"),
             ncol = 3)
?ggplot
```

```
#Investigate correlation between X's and Y
for(i in 1:ncol(X.train)){
  cor = cor(X.train[,i], Y.train)
  print(paste("i = ",  i, ", cor = ", cor))
}

# [1] "i =  1 , cor =  0.0774291944103845"
# [1] "i =  2 , cor =  0.026756233353953"
# [1] "i =  3 , cor =  0.0645911488215885"
# [1] "i =  4 , cor =  0.0673660077273427"
# [1] "i =  5 , cor =  0.0646006572068302"
# [1] "i =  6 , cor =  0.0776083313207561"
# [1] "i =  7 , cor =  0.0849986557442271"
# [1] "i =  8 , cor =  0.0596058212539527"
# [1] "i =  9 , cor =  0.0664688151058887"
# [1] "i =  10 , cor =  0.119396952349143"
# [1] "i =  11 , cor =  0.0596055076857064"
# [1] "i =  12 , cor =  0.071206958395959"
# [1] "i =  13 , cor =  -0.12773370949347"
# [1] "i =  14 , cor =  0.115509719289025"
# [1] "i =  15 , cor =  -0.156666713393043"
# [1] "i =  16 , cor =  0.1271207640929

#They are very weekly correlated. However, their combination can still effectively
# predict Y. Transformation methods are also applied to help the correlation.

par(mfrow = c(3, 3))
plot(X.train$V7, Y.train, col = "blue")
plot(X.train$V9, Y.train, col = "blue")
plot(X.train$V10, Y.train, col = "blue")
plot(X.train$V11, Y.train, col = "blue")
plot(X.train$V12, Y.train, col = "blue")
plot(X.train$V13, Y.train, col = "blue")
plot(X.train$V15, Y.train, col = "blue")
plot(X.train$V19, Y.train, col = "blue")
plot(X.train$V22, Y.train, col = "blue")



#Transformation
X.train.log = log10(X.train)
X.train.sq = sqrt(X.train)

X.test.log = log10(X.test)
X.test.sq = sqrt(X.test)

for(i in 1:ncol(X.train)){
  cor = cor(X.train.sq[,i], Y.train)
  print(paste("i = ",  i, ", cor = ", cor))
}

for(i in 1:ncol(X.train)){
  cor = cor(X.train.log[,i], Y.train)
  print(paste("i = ",  i, ", cor = ", cor))
}
## Correlation stayed quite the same.

####################Usual Linear Regression model:#####################
fit.usual = lm(Y.train ~ X.train)
summary(fit.usual)
fit.usual$coefficients

coef.usual = coef(fit.usual)
Y.pred = coef.usual[1] + X.test%*%coef.usual[-1]

error.usual = mean((Y.test - Y.pred)^2)
error.usual
#[1] 0.0688315

#################Linear regression with Ridge penalty#################
library(glmnet)
fit.ridge = cv.glmnet(X.train, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, X.test , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.06281705

fit.ridge$lambda.min
# [1] 0.007351743
range(fit.ridge$lambda)
```

```
# [1]  0.004206937 38.332046104

library(glmnet)
fit.ridge = cv.glmnet(X.train, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, X.test , s = 1)
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
#[1] 0.05527972

#Ridge fit with sqrt transformation
library(glmnet)
fit.ridge = cv.glmnet(X.train.sq, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, X.test.sq, s = 1)
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05500825

#################Linear regression with Lasso penalty#################
library(glmnet)
fit.lasso = cv.glmnet(X.train, Y.train)
Y.pred = predict(fit.lasso, X.test , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.0625924


fit.lasso$lambda.min
# [1] 0.002352019
range(fit.ridge$lambda)
# [1] 0.004206937 38.332046104

#refit Lasso with lambda = 0.04
library(glmnet)
fit.lasso = cv.glmnet(X.train, Y.train)
Y.pred = predict(fit.lasso, X.test , s = 0.04)
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05633234

#Lasso fit with sqrt transformation
library(glmnet)
fit.lasso = cv.glmnet(X.train.sq, Y.train)
Y.pred = predict(fit.lasso, X.test.sq , s = 0.04)
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05633234

######################## PCA ########################
A = cov(X.train)
eigen(A)$values
which.max(eigen(A)$values) #The first one has the biggest eigenvalue
#hence is the PC with most amount of information

par(mfrow = c(1,2))
plot(eigen(A)$values, type = "h", lwd = 5, col = "purple",
     xlab = "PC's", ylab = "Eigenvalues")
plot(eigen(A)$values[-1], type = "h", lwd = 5, col = "purple",
     xlab = "PC's (minus the 1st)", ylab = "Eigenvalues")
title("The Significant PC's'", outer = TRUE, line = -2, cex = 3)
#PC1 to PC5 seem significant

C = eigen(A)$vectors
new.train = cbind(X.train, X.train%*%C[,c(1:5)])

A = cov(X.test)
eigen(A)$values
which.max(eigen(A)$values) #The first one has the biggest eigenvalue
#hence is the PC with most amount of information

C = eigen(A)$vectors
new.test = cbind(X.test, X.test%*%C[,c(1:5)])

dim(new.train)
#[1] 1000   20 #4 PC's added

new.xtrain = new.train[,17:21]
dim(new.xtrain)
# [1] 1000   5
new.xtest = new.test[,17:21]
```

```
dim(new.xtest)
#[1] 4875   5

#Ridge
library(glmnet)
fit.ridge = cv.glmnet(new.xtrain, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, new.xtest , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
#[1] 0.05460363

#Lasso
fit.lasso = cv.glmnet(new.xtrain, Y.train)
Y.pred = predict(fit.lasso, new.xtest , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05400543

###BEST FIT

fit.lasso$lambda.min
# [1] 0.00187675

coef(fit.lasso)
# 6 x 1 sparse Matrix of class "dgCMatrix"
# 1
# (Intercept)  0.67986900
# -0.04557849
# .
# .
# 5.58207043
# 9.26479508


##################Repeat above for the second Y variable#############

library(scales)
#' want to rescale the data so that they're all between 0 and 1 (normalized)
#' and the interpretibility of the prediction errors is easier

X.train = rescale(as.matrix(train[,7:22]))
X.test = rescale(as.matrix(test[,7:22]))
Y.train = rescale(train[,6])
Y.test = rescale(test[ ,6])


#Investigate correlation between X's and Y
for(i in 1:ncol(X.train)){
  cor = cor(X.train[,i], Y.train)
  print(paste("i = ",  i, ", cor = ", cor))
}
#
# [1] "i =  1 , cor =  0.0739802239448816"
# [1] "i =  2 , cor =  0.0534700980063472"
# [1] "i =  3 , cor =  0.0659329631046685"
# [1] "i =  4 , cor =  0.0620614598319494"
# [1] "i =  5 , cor =  0.0659538125182766"
# [1] "i =  6 , cor =  0.069413301364315"
# [1] "i =  7 , cor =  0.0754844983969396"
# [1] "i =  8 , cor =  0.0574491295062612"
# [1] "i =  9 , cor =  0.0586362496768223"
# [1] "i =  10 , cor =  0.102272961176441"
# [1] "i =  11 , cor =  0.0574490662629386"
# [1] "i =  12 , cor =  0.0624389209831174"
# [1] "i =  13 , cor =  -0.133570072042911"
# [1] "i =  14 , cor =  0.131124242383567"
# [1] "i =  15 , cor =  -0.161134776873336"
# [1] "i =  16 , cor =  0.131711832230074"

#They are very weekly correlated. However, their combination can still effectively
# predict Y. Transformation methods are also applied to help the correlation.

#Transformation
X.train.log = log10(X.train)
X.train.sq = sqrt(X.train)
X.test.log = log10(X.test)
X.test.sq = sqrt(X.test)


####################Usual Linear Regression model:#####################
fit.usual = lm(Y.train ~ X.train)
```

```
summary(fit.usual)
fit.usual$coefficients

coef.usual = coef(fit.usual)
Y.pred = coef.usual[1] + X.test%*%coef.usual[-1]

error.usual = mean((Y.test - Y.pred)^2)
error.usual
#[1] 0.0577305

#################Linear regression with Ridge penalty#################
library(glmnet)
fit.ridge = cv.glmnet(X.train, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, X.test , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05524565


#Fit with sqrt transformation
library(glmnet)
fit.ridge = cv.glmnet(X.train.sq, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, X.test.sq, s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05443571

#################Linear regression with Lasso penalty#################
library(glmnet)
fit.lasso = cv.glmnet(X.train, Y.train)
Y.pred = predict(fit.lasso, X.test , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.05506565


######################### PCA ########################

# A = cov(X.train)
# eigen(A)$values
#
# par(mfrow = c(1,2))
# plot(eigen(A)$values, type = "h", lwd = 5, col = "purple",
#      xlab = "PC's", ylab = "Eigenvalues")
# plot(eigen(A)$values[-1], type = "h", lwd = 5, col = "purple",
#      xlab = "PC's (minus the 1st)", ylab = "Eigenvalues")
# title("The Significant PC's'", outer = TRUE, line = -2, cex = 3)
# #PC1 to PC5 seem significant
#
# C = eigen(A)$vectors
# new.train = cbind(X.train, X.train%*%C[,c(1:5)])
#
# A = cov(X.test)
# eigen(A)$values
# C = eigen(A)$vectors
# new.test = cbind(X.test, X.test%*%C[,c(1:5)])
#
# new.xtrain = new.train[,17:21]
# new.xtest = new.test[,17:21]
#


# Using the new X variables (PC's) computed above as new.xtrain and new.xtest:

library(glmnet)
fit.ridge = cv.glmnet(new.xtrain, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, new.xtest , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
#[1] 0.0476928
# coef(fit.ridge)
# 6 x 1 sparse Matrix of class "dgCMatrix"
# 1
# (Intercept)  0.59749903
# -0.04745375
# 0.46997583
# 0.03654323
# 3.59981162
# 4.75306913


fit.lasso = cv.glmnet(new.xtrain, Y.train)
Y.pred = predict(fit.lasso, new.xtest , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge
# [1] 0.04768863

fit.lasso$lambda.min
# [1] 0.001178654

coef(fit.lasso)
# 6 x 1 sparse Matrix of class "dgCMatrix"
# 1
# (Intercept)  0.71780993
# -0.06315695
# .
# .
# 6.65172809
# 10.65443339

###BEST FIT

fit.lasso$lambda.min
# [1] [1] 0.002040315

coef(fit.lasso)
# 6 x 1 sparse Matrix of class "dgCMatrix"
# 1
# (Intercept)  0.64773879
# -0.04584834
# .
# .
# 6.15853225
# 8.36215970

#Variable selection is done

########################PCA 2 (using PCA function)#######################
train.pca <- prcomp(X.train, center = TRUE, scale. = TRUE)
print(train.pca)
plot(train.pca, type = "l")
test.pca <- prcomp(X.test, center = TRUE, scale. = TRUE)

new.xtrain = train.pca$x[,1:6]
dim(new.xtrain)
# [1] 1000   5
new.xtest = test.pca$x[,1:6]
dim(new.xtest)

library(glmnet)
fit.ridge = cv.glmnet(new.xtrain, Y.train, alpha = 0)
coef(fit.ridge)
range(fit.ridge$lambda)
Y.pred = predict(fit.ridge, new.xtest , s = "lambda.min")
MSE.ridge = mean((Y.pred - Y.test)^2)
MSE.ridge

# [1] 0.054
```

```r
#######################################
#' Project 2
#' @author Bita Nezamdoust
#######################################

install.packages("corrplot")
install.packages("gridExtra")
install.packages("glmnet")
install.packages("rda")
install.packages("e1071")
#################TASK 2###################
#Read Rdata file
get(load("Work/Study_2018/Stat 8670: Computational Methods in
Statistics/Project_2/Paert2:LogisticReg/project_2_data_2.RData"))
ls()
summary(X.train)
summary(Y.train)
table(Y.train)
# 1  2  3  4  5  6
# 45 47 38 49 50 71

cor(X.train[,1], X.train[,5])

#Correlation Analysis
# install.packages("corrplot")

#head(round(M,2))

library(corrplot)
par(mfrow = c(2, 2))
M<-cor(X.train[,c(20:40)])
corrplot(M, method="pie", type = "lower", tl.col = "black")
M<-cor(X.train[,c(60:80)])
corrplot(M, method="pie", type = "lower", tl.col = "black")
M<-cor(X.train[,c(180:200)])
corrplot(M, method="pie", type = "lower", tl.col = "black")
M<-cor(X.train[,c(440:460)])
corrplot(M, method="pie", type = "lower", tl.col = "black")
#corrplot(M, method="number", type = "lower", number.cex = .7, tl.col = "black")
title("MULTICOLLINEARITY BETWEEN PREDICTORS", outer = TRUE, line = -1, cex = 5)



library(ggplot2)
# install.packages("gridExtra")
library(gridExtra)
frame.for.plot = data.frame(X.train)
p1 = ggplot(data = frame.for.plot) +
geom_point(mapping = aes(x = V11, y = V13, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p2 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V20, y = V25, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p3 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V90, y = V92, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p4 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V170, y = V180, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p5 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V195, y = V196, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p6 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V330, y = V333, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p7 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V459, y = V460, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p8 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V424, y = V236, color = Y.train))
scale_color_manual(values = c("red", "blue"))
p9 = ggplot(data = frame.for.plot) +
 geom_point(mapping = aes(x = V510, y = V511, color = Y.train))
scale_color_manual(values = c("red", "blue"))
grid.arrange(p1,p2,p3,p4,p5,p6,p7,p8,p9, ncol = 3)




#Fitting models
########Usual logistic Regression without Penalty######
library(glmnet)
```

```
usual.fit = glmnet(X.train, Y.train, family = "multinomial")
summary(usual.fit)
coef(usual.fit, s = 0)

Y.pred = predict(usual.fit, newx = X.test, s=0, type = "class")
error.usual = sum(Y.pred!=Y.test)/length(Y.test)

error.usual
#[1] 0.09

#########Logistic regression with Ridge penalty#########
#To shrink the parameter estimates to deal with the effect of multicolinearity
#"multinomial" for Y having more than 2 classes
ridge.fit = cv.glmnet(X.train, Y.train, family = "multinomial", alpha = 0)
Y.pred = predict(ridge.fit,newx = X.test, type = "class", s = "lambda.min")
error.ridge = sum(Y.pred!=Y.test)/length(Y.test)
error.ridge
#[1] 0.1533333
coef(ridge.fit)

#Plot
fit.all = glmnet(X.train, Y.train, family = "multinomial", alpha = 0)
plot(fit.all, xvar = "lambda")

#########Logistic regression with Lasso penalty#########
lasso.fit = cv.glmnet(X.train, Y.train, family = "multinomial")
Y.pred = predict(ridge.fit,newx = X.test, type = "class", s = "lambda.min")
error.lasso = sum(Y.pred!=Y.test)/length(Y.test)
error.lasso
# [1] 0.1533333

##########LDA#########
library(MASS)
lda.fit = lda(X.train, Y.train)
Y.pred = predict(lda.fit, X.test)$class
error.lda = sum(Y.pred!=Y.test)/length(Y.test)
error.lda
# [1] 0.2166667
summary(lda.fit)

#########QDA##########
library(MASS)
qda.fit = qda(X.train, Y.train)
Y.pred = predict(qda.fit, X.test)$class
error.qda = sum(Y.pred!=Y.test)/length(Y.test)
error.qda

# [1] 0.2166667

##########RDA##########
#install.packages("rda")
library("rda")
X.train.rda = t(X.train)
Y.train.rda = as.numeric(Y.train)
X.test.rda = t(X.test)
Y.test.rda = as.numeric(Y.test)

rda.fit = rda(X.train.rda, Y.train.rda)
cv.rda.fit = rda.cv(rda.fit, x = X.train.rda, y = Y.train.rda)
alpha = (cv.rda.fit$alpha)
delta = (cv.rda.fit$delta)
opt.alpha = alpha[which.min(apply(cv.rda.fit$cv.err, 1, min))]
opt.delta = delta[which.min(apply(cv.rda.fit$cv.err, 2, min))]
Y.pred = predict(rda.fit, x = X.train.rda, y = Y.train.rda, xnew = X.test.rda,
                 alpha = opt.alpha, delta = opt.delta)
error.rda = sum(Y.pred!=Y.test.rda)/length(Y.test.rda)
error.rda
# 0.0666667

##BEST FIT
predict(rda.fit, x = X.train.rda, y = Y.train.rda, xnew = X.test.rda,
        type = "nonzero")
str(rda.fit)
summary(rda.fit)
opt.alpha
# [1] 0.99
opt.delta
# [1] 0
rda.fit
# $nonzero
# delta
# alpha    0 0.333 0.667   1 1.333 1.667   2 2.333 2.667   3
# 0    561   234    51   1     0     0   0     0     0   0
# 0.11 561   174    28   5     1     0   0     0     0   0
# 0.22 561   162    25   5     1     0   0     0     0   0
# 0.33 561   162    26   6     5     1   0     0     0   0
# 0.44 561   179    28   7     5     1   1     0     0   0
# 0.55 561   195    34  12     6     5   1     1     1   0
# 0.66 561   220    58  19     8     6   6     5     1   1
# 0.77 561   288    98  39    17    10   6     6     6   5
# 0.88 561   431   218 100    49    31  19    12     9   7
# 0.99 561   561   557 543   527   502 466   429   391 356
#
# $errors
# delta
# alpha   0 0.333 0.667   1 1.333 1.667   2 2.333 2.667   3
# 0    41   117   141 229   229   229 229   229   229 229
# 0.11 35   115   138 183   229   229 229   229   229 229
# 0.22 32   107   102 179   229   229 229   229   229 229
# 0.33 30   101    85 179   182   229 229   229   229 229
# 0.44 27    84    82 179   179   187 229   229   229 229
# 0.55 26    73    80 179   179   179 182   229   229 229
# 0.66 19    50    86  96   179   179 179   179   182 229
# 0.77 16    31    67  90   101   179 179   179   179 179
# 0.88  5    16    28  63    84    91  95   133   176 179
# 0.99  0     0     1   2     5     9  16    19    29  37


#More model fitting using Support Vector Machine
################ SVM ##################
#Linear:
# install.packages("e1071")
library(e1071)
fit = svm(X.train, Y.train, cost = 0.5, kernel = "linear", type = "C")
pred.class = predict(fit, X.test)
sum(Y.test!=pred.class)/length(Y.test)
#[1] 0.09333333
fit = svm(X.train, Y.train, cost = 0.05, kernel = "linear", type = "C")
pred.class = predict(fit, X.test)
sum(Y.test!=pred.class)/length(Y.test)
#[1] [1] 0.09333333
fit = svm(X.train, Y.train, cost = 1, kernel = "linear", type = "C")
pred.class = predict(fit, X.test)
sum(Y.test!=pred.class)/length(Y.test)
#[1] [1] 0.09333333
fit = svm(X.train, Y.train, cost = 10, kernel = "linear", type = "C")
pred.class = predict(fit, X.test)
sum(Y.test!=pred.class)/length(Y.test)
#[1] [1] 0.09333333


#Polynomial
library(e1071)
degree = 2
for(C in c(0.1, 1, 5)){
  for(gamma in c(0.0005, 5)){
    for(gamma0 in c(0, 0.1, 10)){
      fit = svm(X.train, Y.train, cost = C, degree = degree, gamma = gamma,
                coef0 = gamma0, kernel = "polynomial", type = "C")
      pred.class = predict(fit, X.test)
      print(c("C, gamma, gamma0, error = ", C, gamma, gamma0,
              sum(Y.test!=pred.class)/length(Y.test)))
    }
  }
}
# [1] "C, gamma, gamma0, error = " "1"
# [3] "5e-04"                      "0"
# [5] "0.76"
# [1] "C, gamma, gamma0, error = " "1"
# [3] "5e-04"                      "0.1"
# [5] "0.426666666666667"
# [1] "C, gamma, gamma0, error = " "1"
# [3] "5e-04"                      "10"
# [5] "0.0866666666666667"
# [1] "C, gamma, gamma0, error = " "1"
# [3] "5"                          "0"
# [5] "0.14"
# ...

fit$coefs #????

#Sigmoid
library(e1071)
for(C in c(1, 5, 100)){
  for(gamma in c(0.0005, 0.2)){
    for(gamma0 in c(0, 0.1, 10)){
      fit = svm(X.train, Y.train, cost = C, degree = degree, gamma = gamma,
                coef0 = gamma0, kernel = "sigmoid", type = "C")
      pred.class = predict(fit, X.test)
      print(c("C, gamma, gamma0, error = ", C, gamma, gamma0,
              sum(Y.test!=pred.class)/length(Y.test)))
    }
  }
}

# [1] "C, gamma, gamma0, error = " "5"
# [3] "5e-04"                      "0"
# [5] "0.106666666666667"
# [1] "C, gamma, gamma0, error = " "5"
# [3] "5e-04"                      "0.1"
# [5] "0.103333333333333"
# [1] "C, gamma, gamma0, error = " "5"
# [3] "5e-04"                      "10"
# [5] "0.793333333333333"
# [1] "C, gamma, gamma0, error = " "5"
# [3] "0.2"                        "0"
# [5] "0.566666666666667"

#Radial
library(e1071)
for(C in c(1, 5, 8)){
  for(gamma in c(0.01, 0.0005)){
    fit = svm(X.train, Y.train, cost = C, gamma = gamma, kernel = "radial", type = "C")
    pred.class = predict(fit, X.test)
    print(c("C, gamma, error = ", C, gamma,
            sum(Y.test!=pred.class)/length(Y.test)))
  }
}


# [1] "C, gamma, error = " "5"                  "0.01"
# [4] "0.18"
# [1] "C, gamma, error = " "5"                  "5e-04"
# [4] "0.0866666666666667"
# [1] "C, gamma, error = " "8"                  "0.01"
# [4] "0.18"
# [1] "C, gamma, error = " "8"                  "5e-04"
# [4] "0.0966666666666667"

##############Classification Tree################
library(rpart)
A = data.frame(X.train, Y.train.factor = as.factor(Y.train))
tree.fit = rpart(Y.train.factor ~., data = A)
Y.pred = predict(tree.fit, data.frame(X.test), type="class")
sum(Y.pred != Y.test)/length(Y.pred)
# [1] 0.21


for(maxdepth in c(3, 6, 9))
{
  for(minobs in c(1, 5, 10, 20))
  {
    for(Cp in c(0, 0.001, 0.01))
    {
      A = data.frame(X.train, Y.train.factor = as.factor(Y.train))
      mycontrol = rpart.control(minsplit = minobs, cp = Cp, maxdepth =maxdepth)
      fit = rpart(Y.train.factor ~., data = A, control = mycontrol)
      a = predict(fit, data.frame(X.test), type = "class")
      print(c("maxdepth =", maxdepth, "minobs =", minobs, "Cp =", Cp, "error =",
            sum(a != Y.test)/length(a)))
    }
  }
}

# [1] "maxdepth ="        "6"                 "minobs ="          "1"
# [5] "Cp ="              "0.01"              "error ="           "0.213333333333333"
# [1] "maxdepth ="        "6"                 "minobs ="          "5"
# [5] "Cp ="              "0"                 "error ="           "0.223333333333333"
# [1] "maxdepth ="        "6"                 "minobs ="          "5"
# [5] "Cp ="              "0.001"             "error ="           "0.223333333333333"
# [1] "maxdepth ="        "6"                 "minobs ="          "5"
# [5] "Cp ="              "0.01"              "error ="           "0.213333333333333"
# [1] "maxdepth ="        "6"                 "minobs ="          "10"
# [5] "Cp ="              "0"                 "error ="           "0.233333333333333"
```