Assignment_1_8259_binfeng2

Bin Feng

Following codes are written for Coding Assignment 1 of STAT 542. To start with, two libraries "ggplot2" and "class" are included. Seed is setted to ensure reproducibility.

```r
#include necessary libraries
library(ggplot2)
library(class)

#set seed to ensure reproducibility
set.seed(8259)
```

Then, creating 10 two-dimensional centers for each class. Two classes are involved, labeled as 0 and 1, respectively.

```r
#create 10 centers for each class
csize = 10;       # number of centers
p = 2;      #dimension size
s_center = 1;      # sd for generating the centers within each class
s = sqrt(1/5);   #sd for generating x
#m0 include 10 center coordinates for class 1, following a normal distribution with
#mean=(0, 0) and sd=1; m1 also include 10 center coordinates for class 2,
#following a normal distribution with mean=(2, 2) and sd=1;
m0 = matrix(rnorm(csize*p), csize, p)*s_center + cbind(rep(0,csize), rep(0,csize));
m1 = matrix(rnorm(csize*p), csize, p)*s_center + cbind(rep(2,csize), rep(2,csize));
```

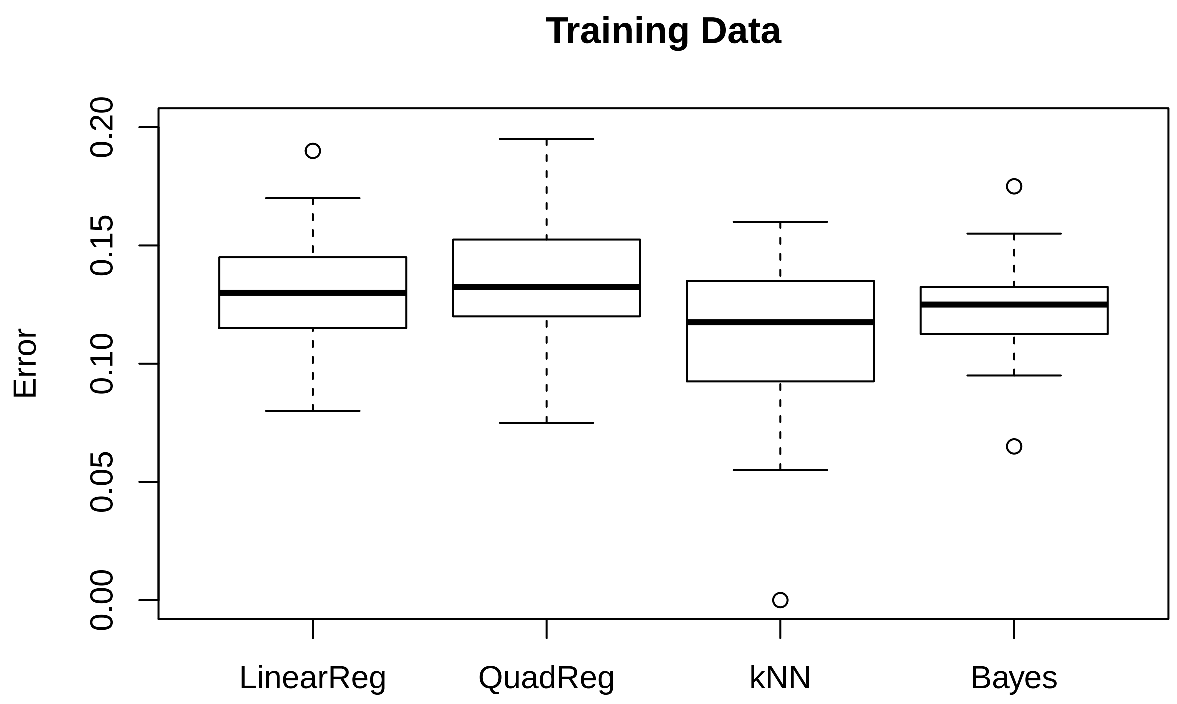
Four regression methods are developed below: linear regression, quadratic regression, kNN classification and the Bayes rule. For all four methods, relevant functions are written first. Then, in the loop for 20 repetitions, functions are called accordingly.

```r
#Linear regression function with cut-off value 0.5
LinearReg = function(traindata, testdata, Ytrain, Ytest){
  LR_Model = lm(as.numeric(Ytrain) - 1 ~ traindata);
  Ytrain_pred_LR = as.numeric(LR_Model$fitted > 0.5);

  Ytest_pred_LR = LR_Model$coef[1] + LR_Model$coef[2] * testdata[,1] +
    LR_Model$coef[3] * testdata[,2];
  Ytest_pred_LR = as.numeric(Ytest_pred_LR > 0.5);

  train.err.LR = sum(Ytrain !=  Ytrain_pred_LR) / (2*n);
  test.err.LR = sum(Ytest !=  Ytest_pred_LR) / (2*N);
  error = list(train.err.LR, test.err.LR);
}

#Quadratic regression function with cut-off value 0.5
QuadReg = function(traindata, testdata, Ytrain, Ytest){
  QR_Model = lm(as.numeric(Ytrain) - 1 ~ traindata[,1] + traindata[,2] +
                  I(traindata[,1] * traindata[,2]) + I(traindata[,1]^2)+I(traindata[,2]^2));
  Ytrain_pred_QR = as.numeric(QR_Model$fitted > 0.5);

  Ytest_pred_QR = QR_Model$coef[1] + QR_Model$coef[2] * testdata[,1] +
    QR_Model$coef[3] * testdata[,2] + QR_Model$coef[4] * (testdata[,1] * testdata[,2]) +
    QR_Model$coef[5] * testdata[,1]^2 + QR_Model$coef[6] * testdata[,2]^2;
  Ytest_pred_QR = as.numeric(Ytest_pred_QR > 0.5);

  # table(Ytrain, Ytrain_pred_QR);
  train.err.QR = sum(Ytrain !=  Ytrain_pred_QR) / (2*n);
  # table(Ytest, Ytest_pred_QR);
  test.err.QR = sum(Ytest !=  Ytest_pred_QR) / (2*N);
  error = list(train.err.QR, test.err.QR);
}

#help function to find the optimum k using 10-fold cross-validation on training data
#In cases of having multiple equally good k-values from CV, the larger one is picked.
find_k = function(traindata, Ytrain){
  myk = seq(1, 180, 5)
  l = length(myk);
  m = 200;

  fold.id = rep(1:10, each = 20);
  fold.id = fold.id[sample(1:m)];

  Y.pred = rep(0, m)
  cv.err.knn = rep(0,10);
  k.err.knn = rep(0,l);

  for(j in 1:l){
    for(i in 1:10){
      test.id = (1:m)[fold.id == i]
      Y.pred = knn(traindata[-test.id, ],
                              traindata[test.id, ],
                              Ytrain[-test.id], k=myk[j]);
      cv.err.knn[i] = sum(Ytrain[test.id] != Y.pred)/(m/10);
    }
    k.err.knn[j] = mean(cv.err.knn);
  }
  max(myk[which(k.err.knn == min(k.err.knn))]);
}

#kNN classification with the optimum k
KNNReg = function(traindata, testdata, Ytrain, Ytest){
  k = find_k(traindata, Ytrain);
  Ytrain.pred = knn(traindata, traindata, Ytrain, k);
  train.err.knn = sum(Ytrain != Ytrain.pred)/(2*n);
  Ytest.pred = knn(traindata, testdata, Ytrain,k);
  test.err.knn = sum(Ytest != Ytest.pred)/(2*N);

  error = list(train.err.knn, test.err.knn, k);
}

#helper function for calculating Bayes error
mixnorm = function(x){
  ## return the density ratio for a point x, where each
  ## density is a mixture of normal with 10 components
  sum(exp(-apply((t(m0)-x)^2, 2, sum)/(2*s^2)))/sum(exp(-apply((t(m1)-x)^2, 2, sum)/(2*s^2)))
}
```

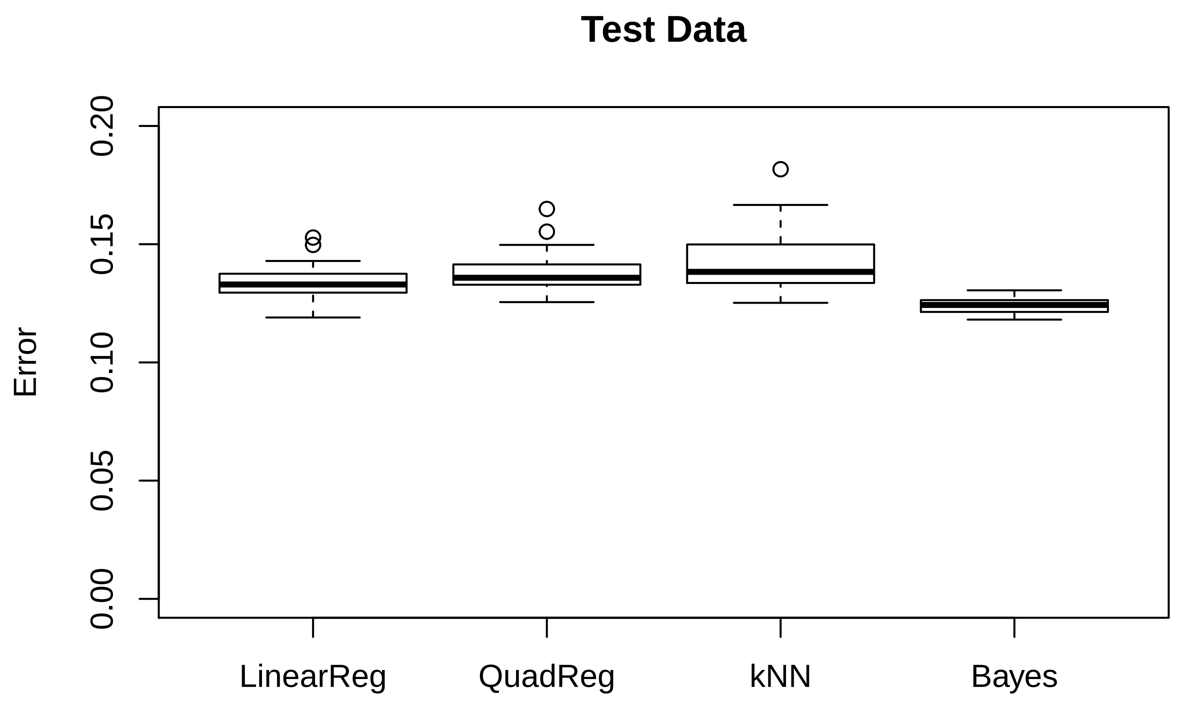
```r
#main Bayes rule funciton
Bayes = function(traindata, testdata, Ytrain, Ytest){
  Ytrain_pred_Bayes = apply(traindata, 1, mixnorm);
  Ytrain_pred_Bayes = as.numeric(Ytrain_pred_Bayes < 1);
  # table(Ytrain, Ytrain_pred_Bayes);
  train.err.Bayes = sum(Ytrain !=  Ytrain_pred_Bayes) / (2*n);

  Ytest_pred_Bayes = apply(testdata, 1, mixnorm)
  Ytest_pred_Bayes = as.numeric(Ytest_pred_Bayes < 1);
  # table(Ytest, Ytest_pred_Bayes);
  test.err.Bayes = sum(Ytest !=  Ytest_pred_Bayes) / (2*N);
  error = list(train.err.Bayes, test.err.Bayes);
}

T = 20;
no.method = 4;
Test.err = matrix(0, T, no.method);
colnames(Test.err) = c("LinearReg", "QuadReg", "kNN", "Bayes");
Train.err = Test.err;
k.vals = rep(0, T);

#main loop
for(t in 1:T){
  #Generating training data
  n = 100;  # n data for each class
  # Randomly allocate the n samples for class 0  to the 10 clusters
  id0 = sample(1:csize, n, replace = TRUE);
  # Randomly allocate the n samples for class 1 to the 10 clusters
  id1 = sample(1:csize, n, replace = TRUE);
  traindata = matrix(rnorm(2*n*p), 2*n, p)*s + rbind(m0[id0,], m1[id1,]);
  Ytrain = factor(c(rep(0,n), rep(1,n)));

  #Generating test data
  N = 5000;
  # Randomly allocate the N samples for class 0  to the 10 clusters
  id0 = sample(1:csize, N, replace=TRUE);
  # Randomly allocate the n samples for class 1 to the 10 clusters
  id1 = sample(1:csize, N, replace=TRUE);
  testdata = matrix(rnorm(2*N*p), 2*N, p)*s + rbind(m0[id0,], m1[id1,]);
  Ytest = factor(c(rep(0,N), rep(1,N)));

  LR_error = LinearReg(traindata, testdata, Ytrain, Ytest);
  QR_error = QuadReg(traindata, testdata, Ytrain, Ytest);
  knn_error = KNNReg(traindata, testdata, Ytrain, Ytest);
  bayes_error = Bayes(traindata, testdata, Ytrain, Ytest);

  Train.err[t,1] = as.numeric(LR_error[1]); Test.err[t,1] = as.numeric(LR_error[2]);
  Train.err[t,2] = as.numeric(QR_error[1]); Test.err[t,2] = as.numeric(QR_error[2]);
  Train.err[t,3] = as.numeric(knn_error[1]); Test.err[t,3] = as.numeric(knn_error[2]);
  Train.err[t,4] = as.numeric(bayes_error[1]); Test.err[t,4] = as.numeric(bayes_error[2]);
  k.vals[t] = as.numeric(knn_error[3]);

}
```

After the for loop, results on training errors and test errors are summarized below using boxplot. The mean and standard error for selected k values are also calculated.

```r
#plot and calculation
boxplot(Train.err, main="Training Data", ylab="Error", ylim = c(0, 0.2));
```



```r
boxplot(Test.err, main="Test Data", ylab="Error", ylim = c(0, 0.2));
```



```r
k = cbind(mean(k.vals), sd(k.vals));
colnames(k) <- c("mean", "standard error");
#displace mean and standard error for k
```

```r
k;
```

```
##       mean standard error
## [1,] 54.25       56.45853
```