Naive Bayes & K-Nearest Neighbor

Dmitrii Dunin, ITU ID 94739

International Technological University

Question 1

```
> rm(list=ls())
> setwd("F:/Workspace/R/Homework5")
> par(mar=rep(2, 4))
> wine_data <- read.table("winequality-red.csv", header=TRUE, sep=";")
> wine_train = wine_data[1:1400,]
> wine_test = wine_data[1401:1599,]
```

1) In homework 3 ridge regression was performed on the wine quality data set. Now use k-nearest neighbors to classify this data. Use cross validation to choose the best value for k. Round the results from the ridge regression to the nearest integer to form the classification with ridge regression. Using the best values for k (nearest neighbor) and lambda (ridge regression), compare and contrast the results of these two classification techniques.

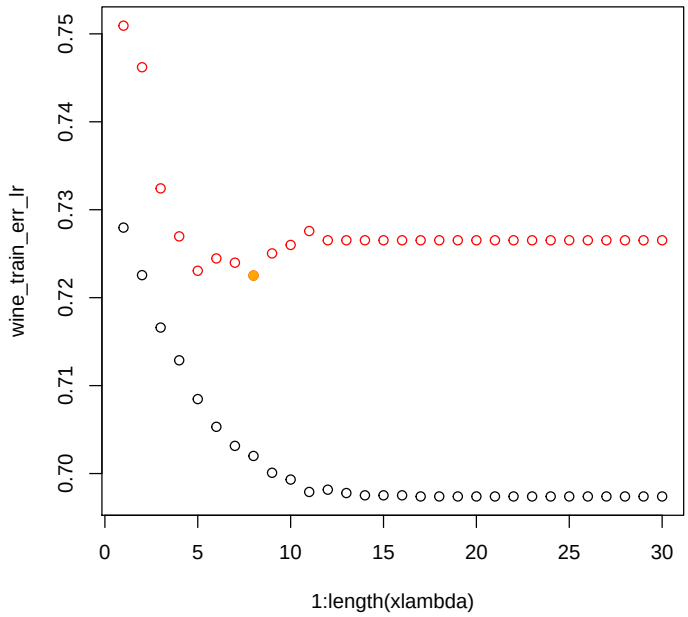
Answer 1

```
> wine_train_cv = wine_train
> rmse = function(X, Y)
+ {
+   return(sqrt(sum((X - Y) ^ 2) / length(Y)))
+ }
> library(MASS)
> xlambda = rep(0, times = 30)
> for (i in seq(from = 0, to = 29))
+ {
+   exp <- (+3 - 4 * (i / 20))
+   xlambda[i + 1] <- 10 ^ exp
+ }
> cross_valid <-
+   function(df_train_cv, k, args, method, err_type = "rmse")
+   {
+     error_train <- 0
+     error_cv <- 0
+     num_sample <- nrow(df_train_cv)
+     df_train = df_train_cv[1:(num_sample * ((k - 1) / k)), ]
+     df_cv = df_train_cv[1:(num_sample * (1 / k)), ]
+     pick <- k #pick kth set
```

```
+
+     for (j in 1:k) {
+       i_tmp <- 1
+       for (i in 1:k) {
+         if (i == pick) {
+           df_cv <- df_train_cv[((i - 1) * num_sample / k + 1):(num_sample * (i / k)), ]
+         } else {
+           df_train[((i_tmp - 1) * num_sample / k + 1):(num_sample * (i_tmp / k)),] <-
+             df_train_cv[((i - 1) * num_sample / k + 1):(num_sample * (i / k)), ]
+           i_tmp <- i_tmp + 1
+         }
+
+       }
+
+       pick <- pick - 1
+
+       y_df_train <- df_train[, ncol(df_train)]
+       x_df_train <- df_train[, -ncol(df_train)]
+       yx_df_train <- cbind(x_df_train, y_df_train)
+
+
+       y_df_cv <- df_cv[, ncol(df_cv)]
+       x_df_cv <- df_cv[, -ncol(df_cv)]
+       yx_df_cv <- cbind(x_df_cv, y_df_cv)
+
+       if (method == "linear ridge") {
+         fit <- lm.ridge(y_df_train ~ ., yx_df_train, lambda = xlambda[args])
+         A <- as.array(fit$coef[1:(ncol(df_train) - 1)] / fit$scales)
+         X <- as.matrix(x_df_train)
+         for (i in seq(from = 1, to = ncol(x_df_train))) {
+           X[, i] <- X[, i] - fit$xm[i]
+         }
+         yh <- X %*% A + fit$ym
+         error_train <- error_train + rmse(round(yh), y_df_train) / k * 0.1 / 0.1
+         X <- as.matrix(x_df_cv)
+         for (i in seq(from = 1, to = ncol(x_df_cv))) {
+           X[, i] <- X[, i] - fit$xm[i]
+         }
+         yh <- X %*% A + fit$ym
+         error_cv <- error_cv + rmse(round(yh), y_df_cv) / k * 0.1 / 0.1
+       }
+
+       if (method == "k nearest neighbors") {
+         y_df_train <- as.factor(y_df_train)
+
+         x_df_train <- scale(x_df_train)
```

```
+         KNN <- knn(x_df_train, x_df_train, y_df_train, k = xnn[args])
+
+         if (err_type == "rmse") {
+           error_train <-
+             error_train + rmse(as.numeric(as.character(y_df_train)),
+                                as.numeric(as.character(KNN))) / k * 0.1 / 0.1
+         }
+
+         if (err_type == "class") {
+           error_train <-
+             error_train + (1 - sum(abs(y_df_train == KNN)) / length(KNN)) / k * 0.1 /
+             0.1
+         }
+
+         x_df_cv <- scale(x_df_cv)
+         KNN <- knn(x_df_train, x_df_cv, y_df_train, k = xnn[args])
+         if (err_type == "rmse") {
+           error_cv <-
+             error_cv + rmse(y_df_cv, as.numeric(as.character(KNN))) / k * 0.1 / 0.1
+         }
+
+         if (err_type == "class") {
+           error_cv <-
+             error_cv + (1 - sum(abs(y_df_cv == KNN)) / length(KNN)) / k * 0.1 / 0.1
+         }
+       }
+     }
+     return(c(error_train, error_cv))
+   }
> k <- 5
> wine_train_err <- NULL
> wine_cv_err <- NULL
> for (ilambda in 1:length(xlambda)) {
+   wine_err <-
+     cross_valid(wine_train_cv, k, ilambda, method = "linear ridge")
+   wine_train_err[ilambda] <- wine_err[1]
+   wine_cv_err[ilambda] <- wine_err[2]
+ }
> wine_train_err_lr <- wine_train_err
> wine_cv_err_lr <- wine_cv_err
> min_lambda_id <- min(which(min(wine_cv_err_lr) == wine_cv_err_lr))
> min_wine_lambda <- xlambda[min_lambda_id]
> sprintf("%dth lambda %f is optimal.", min_lambda_id, min_wine_lambda)

[1] "8th lambda 39.810717 is optimal."

> plot(1:length(xlambda), wine_train_err_lr, ylim = c(
```

```
+   min(wine_train_err_lr, wine_cv_err_lr),
+   max(wine_train_err_lr, wine_cv_err_lr)
+ ))
> points(1:length(xlambda), wine_cv_err_lr, col = 'red')
> points(min_lambda_id,
+        wine_cv_err_lr[min_lambda_id],
+        pch = 19,
+        col = "orange")
```

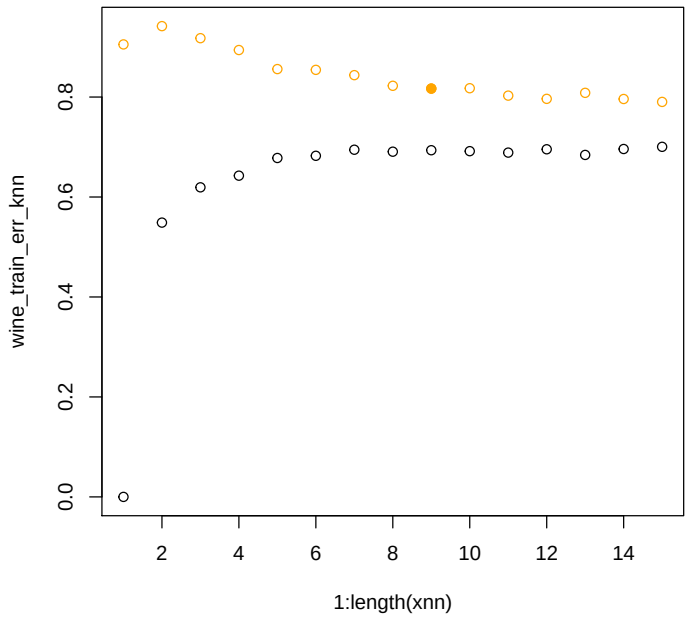



```
> library(class)
> require(class)
> wine_train_err <- NULL
> wine_cv_err <- NULL
> xnn <- c(1:15)
> for (inn in 1:length(xnn)) {
+   wine_err <-
+     cross_valid(wine_train_cv, k, inn, method = "k nearest neighbors")
+   wine_train_err[inn] <- wine_err[1]
+   wine_cv_err[inn] <- wine_err[2]
+ }
> wine_train_err_knn <- wine_train_err
> wine_cv_err_knn <- wine_cv_err
```

```
> margin = 0.04
> min_k_id <-
+   min(which(min(wine_cv_err_knn) + margin > wine_cv_err_knn))
> min_k_id <-
+   ifelse(xnn[min_k_id] %% 2, min_k_id, min_k_id + 1) # K must be odd
> min_wine_k <- xnn[min_k_id]
> sprintf("k = %d is optimal.", min_wine_k)
```

```
[1] "k = 9 is optimal."
```

```
> plot(1:length(xnn), wine_train_err_knn, ylim = c(
+   min(wine_train_err_knn, wine_cv_err_knn),
+   max(wine_train_err_knn, wine_cv_err_knn)
+ ))
> points(1:length(xnn), wine_cv_err_knn, col = 'orange')
> points(min_k_id, wine_cv_err_knn[min_k_id], pch = 19, col = "orange")
```




```
> #Comparing 2 algorithms
>
> y_wine_train_cv <- wine_train_cv[, ncol(wine_train_cv)]
> x_wine_train_cv <- wine_train_cv[, -ncol(wine_train_cv)]
> yx_wine_train_cv <- cbind(x_wine_train_cv, y_wine_train_cv)
> y_wine_test <- wine_test [, ncol(wine_test)]
```

```
> x_wine_test <- wine_test [, -ncol(wine_test)]
> yx_wine_test <- cbind(x_wine_test, y_wine_test)
> fit <-
+   lm.ridge(y_wine_train_cv ~ ., yx_wine_train_cv, lambda = min_wine_lambda)
> A <- as.array(fit$coef[1:(ncol(wine_train_cv) - 1)] / fit$scales)
> X <- as.matrix(x_wine_train_cv)
> for (i in seq(from = 1, to = ncol(x_wine_train_cv))) {
+   X[, i] <- X[, i] - fit$xm[i]
+ }
> yh <- X %*% A + fit$ym
> error_wine_train_lr <- rmse(round(yh), y_wine_train_cv) * 0.1 / 0.1
> error_wine_train_lr

[1] 0.7010197

> X <- as.matrix(x_wine_test)
> for (i in seq(from = 1, to = ncol(x_wine_test))) {
+   X[, i] <- X[, i] - fit$xm[i]
+ }
> yh <- X %*% A + fit$ym
> error_wine_test_lr <- rmse(round(yh), y_wine_test) * 0.1 / 0.1
> error_wine_test_lr

[1] 0.7263871

> y_wine_train_cv <- as.factor(y_wine_train_cv)
> x_wine_train_cv <- scale(x_wine_train_cv)
> KNN <-
+   knn(x_wine_train_cv, x_wine_train_cv, y_wine_train_cv, k = min_wine_k)
> error_wine_train_knn <-
+   rmse(as.numeric(as.character(y_wine_train_cv)), as.numeric(as.character(KNN))) *
+   0.1 / 0.1
> error_wine_train_knn

[1] 0.6813851

> x_wine_test <- scale(x_wine_test)
> KNN <-
+   knn(x_wine_train_cv, x_wine_test, y_wine_train_cv, k = min_wine_k)
> error_wine_test_knn <-
+   rmse(y_wine_test, as.numeric(as.character(KNN))) * 0.1 / 0.1
> error_wine_test_knn

[1] 0.7988685

> tb_wine_lr_vs_knn <-
+   data.frame(
```

```
+     "Train err" = c(error_wine_train_lr, error_wine_train_knn),
+     "Test err" = c(error_wine_test_lr, error_wine_test_knn)
+   )
> row.names(tb_wine_lr_vs_knn) <-
+   c("Linear Ridge", "K-Nearest Neighbors")
> print(tb_wine_lr_vs_knn)

                   Train.err  Test.err
Linear Ridge        0.7010197 0.7263871
K-Nearest Neighbors 0.6813851 0.7988685
```

Question 2

```
> rm(list=ls())
```

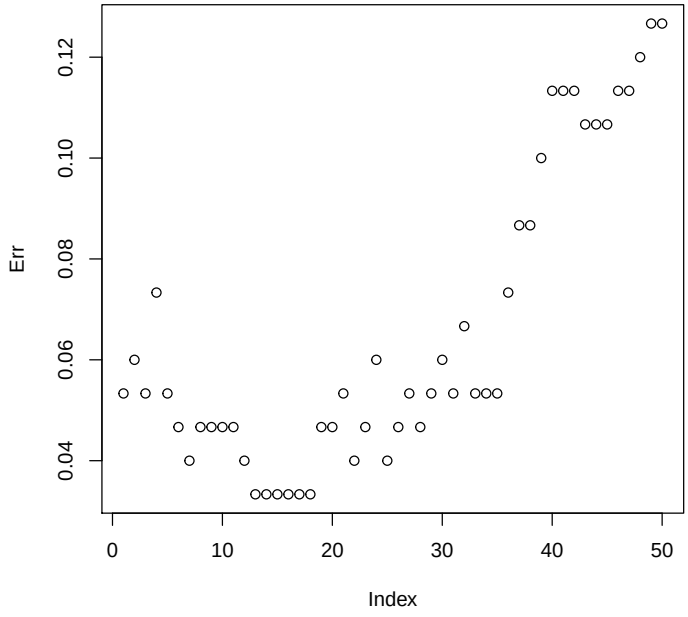
2) Use k-nearest neighbors to classify the Iris data set. Compare the knearest neighbor results with the results obtained in class using the Naive Bayes Classifier.

Answer 2

```
> library(class)
> library(e1071)
> iris = as.data.frame(iris)
> str(iris)

'data.frame':        150 obs. of  5 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 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

> train <- iris[, 1:4]
> labels <- iris[, 5]
> ###Scale the data
> train2 <- train
> for (i in seq(from = 1, to = ncol(train))) {
+   v = var(train[, i])
+   m = mean(train[, i])
+   train2[, i] <- (train[, i] - m) / sqrt(v)
+ }
> ####Perform cross validation on the new data
> out <- knn.cv(train2, labels, k = 3)
> 1 - sum(abs(labels == out)) / length(out)
```

```
[1] 0.05333333
```

```
> ## [1] 0.05333333
> Err <- rep(0, 50)
> for (kk in seq(from = 1, to = 50)) {
+   out <- knn.cv(train2, labels, k = kk)
+   Error <- 1 - sum(abs(labels == out)) / length(out)
+   Err[kk] <- Error
+ }

> plot(Err)
```



Question 3

3) Classify the wine quality data using Naive Bayes. Compare the results with the two methods described in problem 1 of this homework set. Think about why one of the methods used works better than the others.

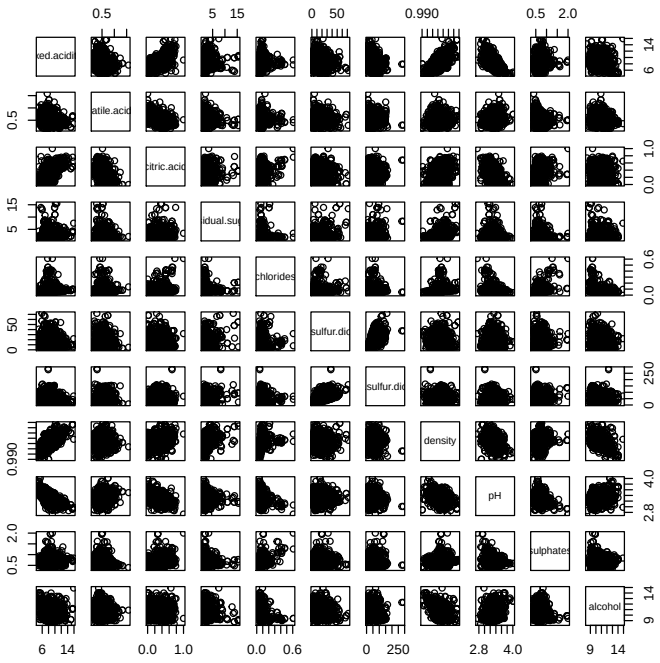
```
> rm(list = ls())
> library(klaR)
> wine_data <- read.table("winequality-red.csv", header = TRUE, sep = ";")
> str(wine_data)
```

```
'data.frame':        1599 obs. of  12 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

Answer 3

```
> wine_data$quality <- as.factor(wine_data$quality)
> mod <- naiveBayes(quality~.,data = wine_data)
> qualityHat <- predict(mod, wine_data[,1:11])
> Err <- 1 - sum(qualityHat == wine_data$quality)/length(wine_data$quality)
```

```
> pairs(wine_data[,1:11])
```



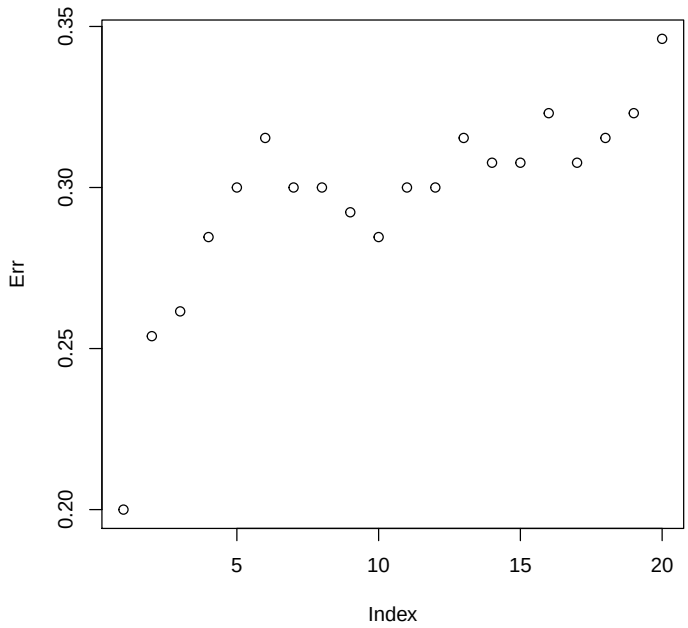
Question 4

4) Classify the sonar data using Naive Bayes. Compare the results with the methods used in class and with the last homework set. Give reasons for any discrepancies between the results for these methods. (Either in class or in homework, the following methods have been used on this data set: Trees, Linear Regression, Ridge Regression, an Ensemble Method, and now Naive Bayes.)

```
> rm(list = ls())
> library(class)
> library(e1071)
> library(klaR)
> library(MASS)
> library(rpart)
> sonar.train <- read.csv("sonar_train.csv", header = FALSE)
> sonar.train$V61 <- as.factor(sonar.train$V61)
> m <- NaiveBayes(V61 ~ ., data = sonar.train)
> out <- predict(m)
```

Answer 4

```
> Err <- 1 - sum(out$class == sonar.train$V61) / length(sonar.train$V61)
> Err

[1] 0.2384615

> train.labels <- sonar.train$V61
> train <- sonar.train[,-61]
> Err <- rep(0, 20)
> for (kk in seq(from = 1, to = 20)) {
+   out <- knn.cv(train, train.labels, k = kk)
+   Error <- 1 - sum(abs(train.labels == out)) / length(out)
+   Err[kk] <- Error
+ }
> Err

 [1] 0.2000000 0.2538462 0.2615385 0.2846154 0.3000000 0.3153846 0.3000000
 [8] 0.3000000 0.2923077 0.2846154 0.3000000 0.3000000 0.3153846 0.3076923
[15] 0.3076923 0.3230769 0.3076923 0.3153846 0.3230769 0.3461538

> plot(Err)
```




```
> bestk = which.min(Err)
> bestk

[1] 1
```

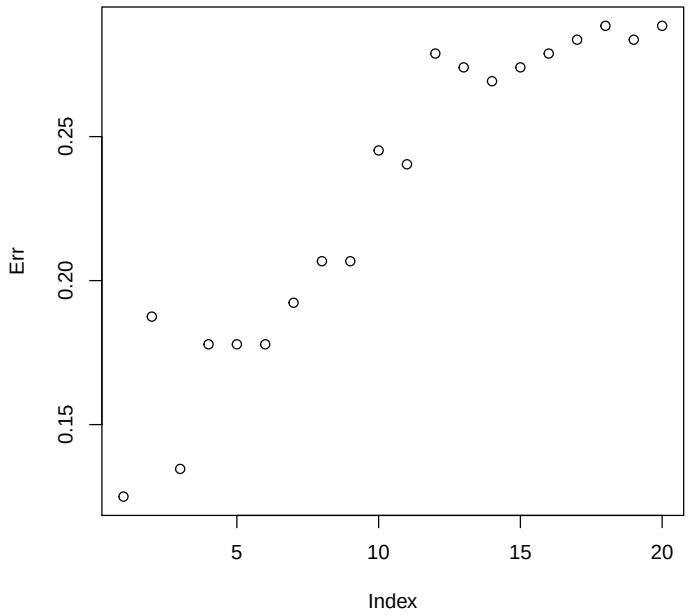
Question 5

5) Run the code in the file KfirstNearestNeighbor.R Does KNN create a better model if the data is first scaled and normalized? What should be chosen as the best value for k and why? Now use KNN with cross validation on the mixtureSimData.data. What is the best value for k for this data?

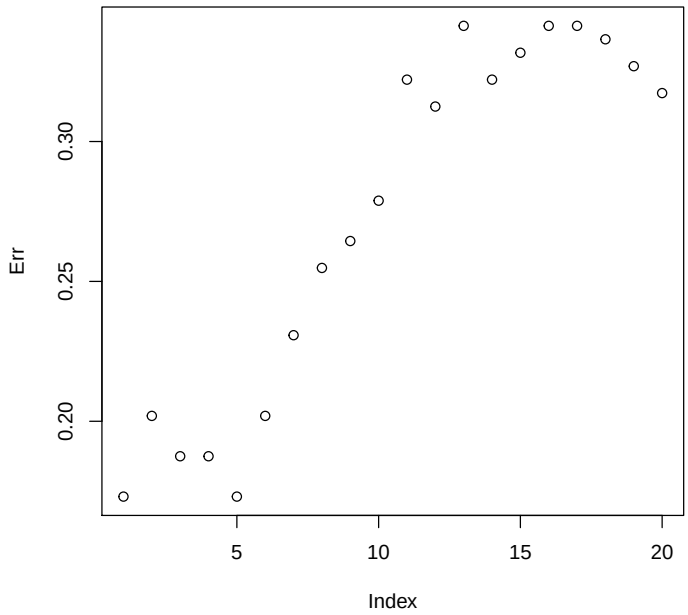
```
> rm(list = ls())
> oldpar <- par(no.readonly = TRUE)
> par(mar = rep(1, 4))
> library(class)
> library(e1071)
> STrain <- read.table("sonar_train.csv", sep = ",", header = FALSE)
> STest <- read.table("sonar_test.csv", sep = ",", header = FALSE)
```

Answer 5

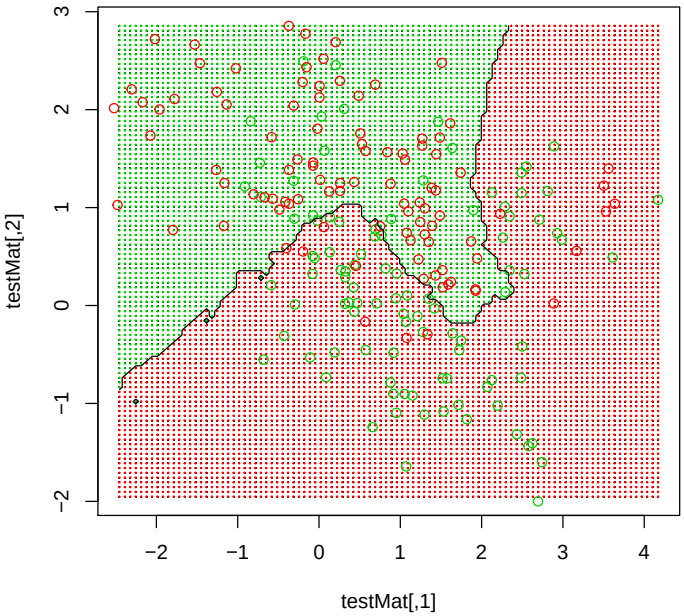
```
> Sonar <- rbind(STrain, STest)
> train <- Sonar[, 1:60]
> labels <- Sonar[, 61]
> # Test error with knn
> out <- knn.cv(train, labels, k = 1)
> 1 - sum(abs(labels == out)) / length(out)

[1] 0.1730769

> train2 <- train
> for (i in seq(from = 1, to = ncol(train))) {
+   v = var(train[, i])
+   m = mean(train[, i])
+   train2[, i] <- (train[, i] - m) / sqrt(v)
+ }
> out <- knn.cv(train2, labels, k = 1)
> 1 - sum(abs(labels == out)) / length(out)

[1] 0.125

> # Cross - validation
>
> Err <- rep(0, 20)
> for (kk in seq(from = 1, to = 20)) {
+   out <- knn.cv(train2, labels, k = kk)
+   Error <- 1 - sum(abs(labels == out)) / length(out)
+   Err[kk] <- Error
+ }
> Err

 [1] 0.1250000 0.1875000 0.1346154 0.1778846 0.1778846 0.1778846 0.1923077
 [8] 0.2067308 0.2067308 0.2451923 0.2403846 0.2788462 0.2740385 0.2692308
[15] 0.2740385 0.2788462 0.2836538 0.2884615 0.2836538 0.2884615

> plot(Err)
```




```
> ## k = 1 gives the best result ----
>
> Err <- rep(0,20)
> for(kk in seq(from=1,to=20)){
+   out <- knn.cv(train,labels,k=kk)
+   Error <- 1-sum(abs(labels == out))/length(out)
+   Err[kk] <- Error
+ }
> Err

 [1] 0.1730769 0.2019231 0.1875000 0.1875000 0.1730769 0.2019231 0.2307692
 [8] 0.2548077 0.2644231 0.2788462 0.3221154 0.3125000 0.3413462 0.3221154
[15] 0.3317308 0.3413462 0.3413462 0.3365385 0.3269231 0.3173077

> plot(Err)
```

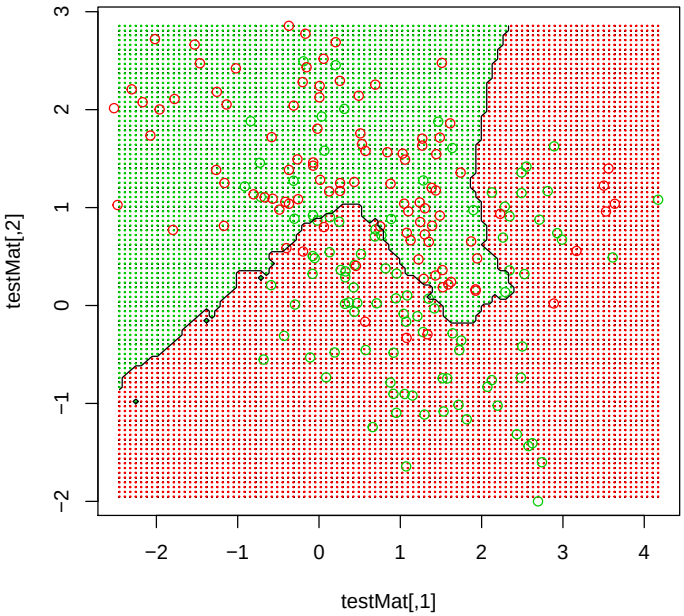



```
> mixSim <- read.table(file="mixtureSimData.data")
> mixSimMat <- matrix(0.0,200,2)
> mixSimMat[,1] <- mixSim[1:200,1]
> mixSimMat[,2] <- mixSim[201:400,1]
> Y <- rep(1.0,200)
> Y[101:200] <- 2.0

> plot(mixSimMat)
> points(mixSimMat[101:200,1:2], col = 2) # color = Red
> points(mixSimMat[1:100,1:2], col = 3)   # color = Green
> linMod <- lm(Y~mixSimMat)
> coef <- linMod$coefficients
> a <- (1.5-coef[1])/coef[3]
> b <- -coef[2]/coef[3]
> abline(a,b)
```



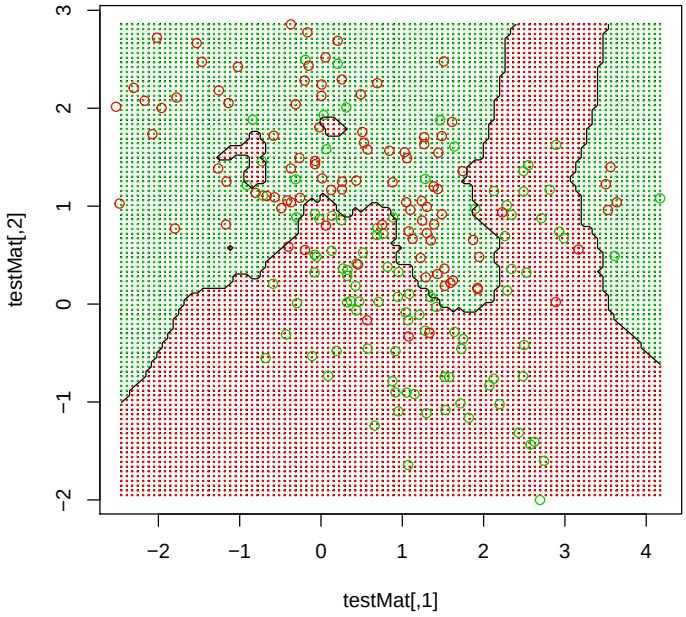

```
> maxX1 <- max(mixSimMat[,1])
> minX1 <- min(mixSimMat[,1])
> maxX2 <- max(mixSimMat[,2])
> minX2 <- min(mixSimMat[,2])
> testMat <- matrix(0.0,10000,2) #matrix(data,number of rows, number of columns)
> for(i in 1:100){
+   for(j in 1:100){
+     x1 <- minX1 + i*(maxX1 - minX1)/100
+     x2 <- minX2 + j*(maxX2 - minX2)/100
+     index <- (i-1)*100 + j
+     testMat[index,1] <- x1
+     testMat[index,2] <- x2
+   }
+ }
> XX <- c((1:100)*(maxX1 - minX1))
> XX <- XX/100
> XX <- XX + minX1
> YY <- c((1:100)*(maxX2 - minX2))
> YY <- YY/100
> YY <- YY + minX2
> i = 7
> j = 2
```

```
> index <- (i-1)*100 + j
> testMat[index,]

[1] -2.05241 -1.90274

> XX[i]

[1] -2.05241

> YY[j]

[1] -1.90274

> require(class)
> KNN <- knn(mixSimMat, testMat, Y, 15)
> ZZ <- matrix(0.0,100,100)
> for(i in 1:100){
+   for(j in 1:100){
+     index <- (i-1)*100 + j
+     ZZ[i,j] <- KNN[index]
+   }
+ }
> i = 7;j = 2;index <- (i-1)*100 + j
> KNN[index]

[1] 1
Levels: 1 2

> ZZ[i,j]

[1] 1

> I1 <- which(KNN == 1)
> # first plot the grid as .  (test data)
> # next plot the data as o
> plot(testMat, pch=".")
> points(testMat[I1,], col=2, cex = 0.2,pch=20)   # plot a small red .
> points(testMat[-I1,],col = 3, cex = 0.2,pch=20) # plot a small green .
> points(mixSimMat[1:100,], col = 3)
> points(mixSimMat[101:200,], col = 2)
> contour(XX,YY,ZZ,levels = 1.5, drawlabels = FALSE, add = TRUE)
```



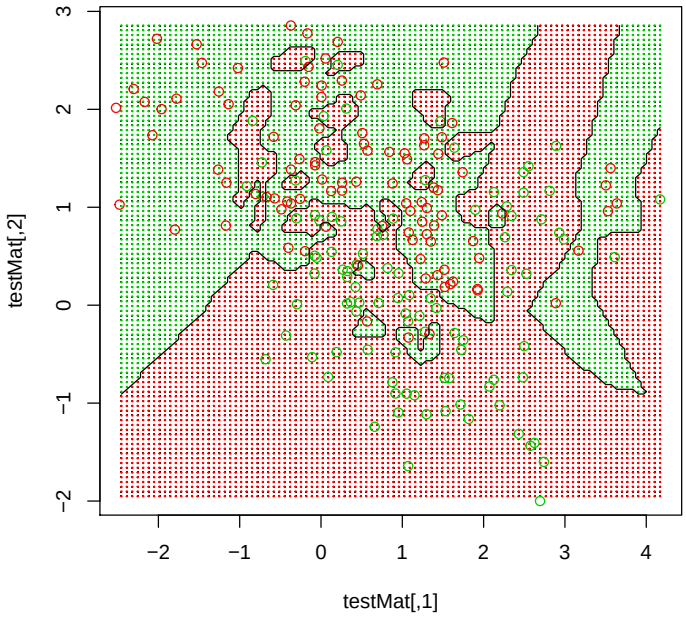

```
> KNN <- knn(mixSimMat, testMat, Y, 5)
> ZZ <- matrix(0.0, 100, 100)
> for (i in 1:100) {
+   for (j in 1:100) {
+     index <- (i - 1) * 100 + j
+     ZZ[i, j] <- KNN[index]
+   }
+ }
> I1 <- which(KNN == 1)
> plot(testMat, pch = ".")
> points(testMat[I1, ],
+        col = 2,
+        cex = 0.2,
+        pch = 20)
> points(testMat[-I1, ],
+        col = 3,
+        cex = 0.2,
+        pch = 20)
> points(mixSimMat[1:100, ], col = 3)
> points(mixSimMat[101:200, ], col = 2)
> contour(XX,
+         YY,
```

```
+          ZZ,
+          levels = 1.5,
+          drawlabels = FALSE,
+          add = TRUE)
```




```
> KNN <- knn(mixSimMat, testMat, Y, 1)
> ZZ <- matrix(0.0, 100, 100)
> for (i in 1:100) {
+   for (j in 1:100) {
+     index <- (i - 1) * 100 + j
+     ZZ[i, j] <- KNN[index]
+   }
+ }
> I1 <- which(KNN == 1)
> plot(testMat, pch = ".")
> points(testMat[I1, ],
+        col = 2,
+        cex = 0.2,
+        pch = 20)
> points(testMat[-I1, ],
+        col = 3,
+        cex = 0.2,
+        pch = 20)
```

```
> points(mixSimMat[1:100, ], col = 3)
> points(mixSimMat[101:200, ], col = 2)
> contour(XX,
+         YY,
+         ZZ,
+         levels = 1.5,
+         drawlabels = FALSE,
+         add = TRUE)
```




```
> Err <- rep(0,50)
> for(kk in seq(from=1,to=50)){
+   out <- knn.cv(mixSimMat, Y, k=kk)
+   Error <- 1-sum(abs(Y == out))/length(out)
+   Err[kk] <- Error
+ }
> Err
 [1] 0.240 0.270 0.185 0.190 0.165 0.195 0.175 0.170 0.195 0.195 0.190 0.170
[13] 0.165 0.165 0.190 0.190 0.190 0.190 0.180 0.195 0.190 0.185 0.175 0.190
[25] 0.195 0.185 0.185 0.185 0.185 0.195 0.195 0.205 0.220 0.205 0.215 0.220
[37] 0.235 0.240 0.235 0.230 0.240 0.245 0.230 0.225 0.230 0.220 0.215 0.220
[49] 0.220 0.225

> min(Err)
```

```
[1] 0.165

> which(Err == min(Err))

[1]  5 13 14

> plot(Err)
```