Math574M_Hw3

Saheed Adisa, Ganiyu

2023-09-28

```
# Loading necessary libraries
library(MASS)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(ggplot2)  # Load ggplot2 for plotting
library(class)
```

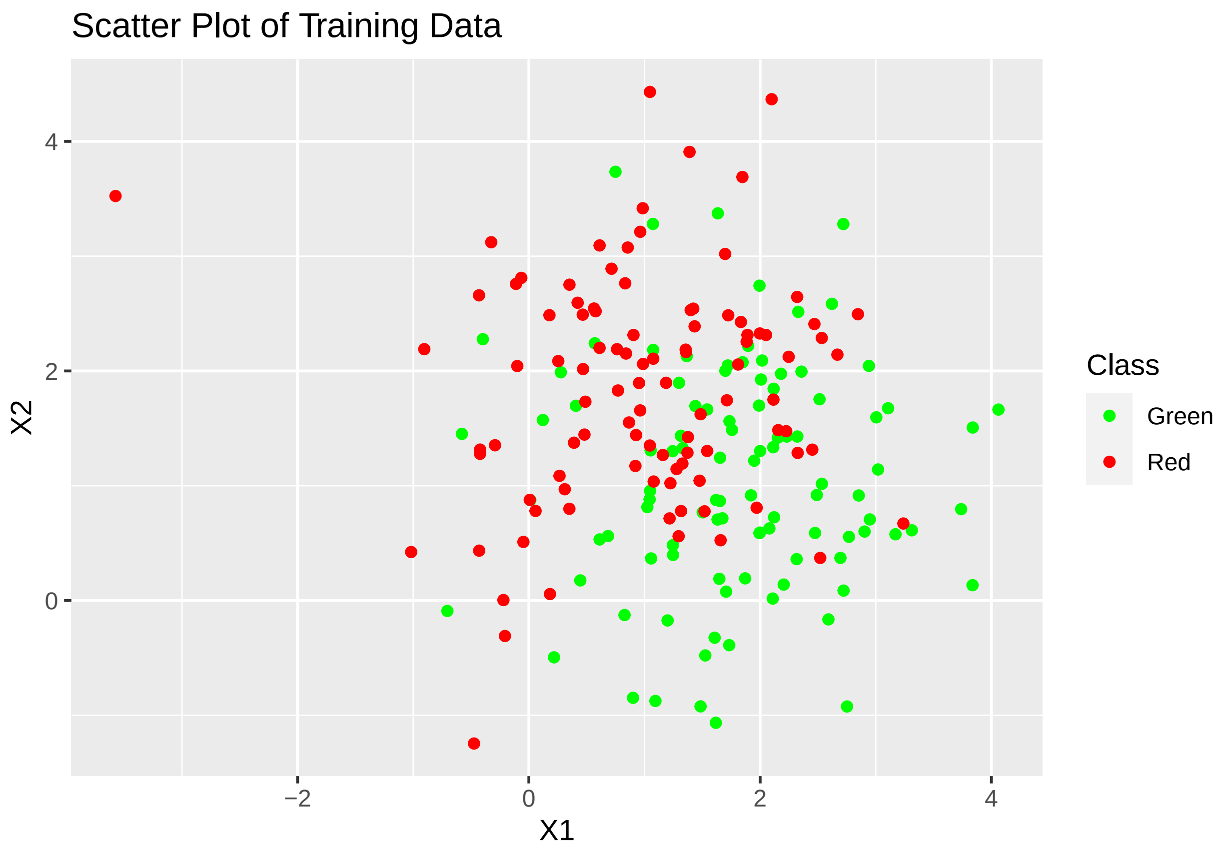
From homework#2

Question 6. (Two-Class Classification Problem: Scenario 1)

Generate 100 observations from a bivariate Gaussian distribution $N(\mu_1, \Sigma_1)$ with $\mu_1 = (2,1)^T$ and $\Sigma_1 = \mathbf{I}$ (the identity matrix), and label them as Green. Generate 100 observations from a bivariate Gaussian distribution $N(\mu_2, \Sigma_2)$ with $\mu_2 = (1,2)^T$ and $\Sigma_2 = \mathbf{I}$, and label them as Red.
(a) Write R code to generate the training data. Set the seed with set.seed(2023) before calling the random number generation function.
(b) Draw the scatter plot of the training data, using different labels/colors for two classes.

```
set.seed(2023)

# setting mean and covariance
n1 = 100
mean1 <- c(2,1)
mean2 <- c(1,2)
cov1 = matrix(c(1,0,0,1),nrow=2)
cov2 = matrix(c(1,0,0,1),nrow=2)
green_train = mvrnorm(100,mean1,cov1)      # generating a random sample of 50 data points from a multiva
class1_labels <- rep("Green", n1)
red_train = mvrnorm(n1,mean2,cov2)
class2_labels <- rep("Red", n1)

# Combining both green and red data, then add class labels
training <- rbind(data.frame(green_train, Class = class1_labels),
                       data.frame(red_train, Class = class2_labels))
training$Class <- as.factor(training$Class)

# b) Plot the scatter plot of training data
```

```r
ggplot(training, aes(x = X1, y = X2, color = Class)) +
  geom_point() +
  scale_color_manual(values = c("Green" = "green", "Red" = "red")) +
  labs(title = "Scatter Plot of Training Data", x = "X1", y = "X2")
```



(c) Generate a test set, with 500 observations from each class, using set.seed(2024). Save the data set for future use.

```r
set.seed(2024)

# Generate test data
n2 <- 500   # Number of observations per class
green_test <- mvrnorm(n2, mean1, cov1)
class1_test_labels <- rep("Green", n2)
red_test <- mvrnorm(n2, mean2, cov2)
class2_test_labels <- rep("Red", n2)

# Combine data and labels for the test set
testing <- rbind(data.frame(green_test, Class = class1_test_labels),
                    data.frame(red_test, Class = class2_test_labels))
testing$Class <- as.factor(testing$Class)
```

Homework#3 begins

5. (Linear Method for Classification: Scenario 1) For this problem, use the training and test data sets you generated in HW2 - Question 6.

(a) Fit the LDA for the training data, and report the training and testing errors.

```
# Fit  LDA model, and make predictions on training data
lda_model <- lda(Class ~ X1 + X2, data = training)
training_pred_lda <- predict(lda_model, training)$class
lda_training_error <- mean(training_pred_lda != training$Class)      # Calculate training error

# Make predictions on test data
testing_pred_lda <- predict(lda_model, testing)$class
lda_testing_error <- mean(testing_pred_lda != testing$Class)     # Calculate testing error

# printing results
cat("LDA Training Error:", lda_training_error, "\n")
```

```
## LDA Training Error: 0.28
```

```
cat("LDA Testing Error:", lda_testing_error, "\n")
```

```
## LDA Testing Error: 0.238
```

(b) Fit the logistic regression for the training data, and report the training and testing errors.

```
# Fit logistic regression model and make predictions on training data
logistic_model <- glm(Class ~ X1 + X2, data = training, family = binomial)
logistic_training_probabilities <- predict(logistic_model, type = "response", newdata = training)
logistic_training_pred <- ifelse(logistic_training_probabilities > 0.5, "Red", "Green")
logistic_training_error <- mean(logistic_training_pred != training$Class) # Calculate training error

# Make predictions on test data
logistic_testing_probabilities <- predict(logistic_model, type = "response", newdata = testing)
logistic_testing_pred <- ifelse(logistic_testing_probabilities > 0.5, "Red", "Green")
logistic_testing_error <- mean(logistic_testing_pred != testing$Class)     # Calculate testing error

# printing results
cat("Logistic Regression Training Error:", logistic_training_error, "\n")
```

```
## Logistic Regression Training Error: 0.265
```

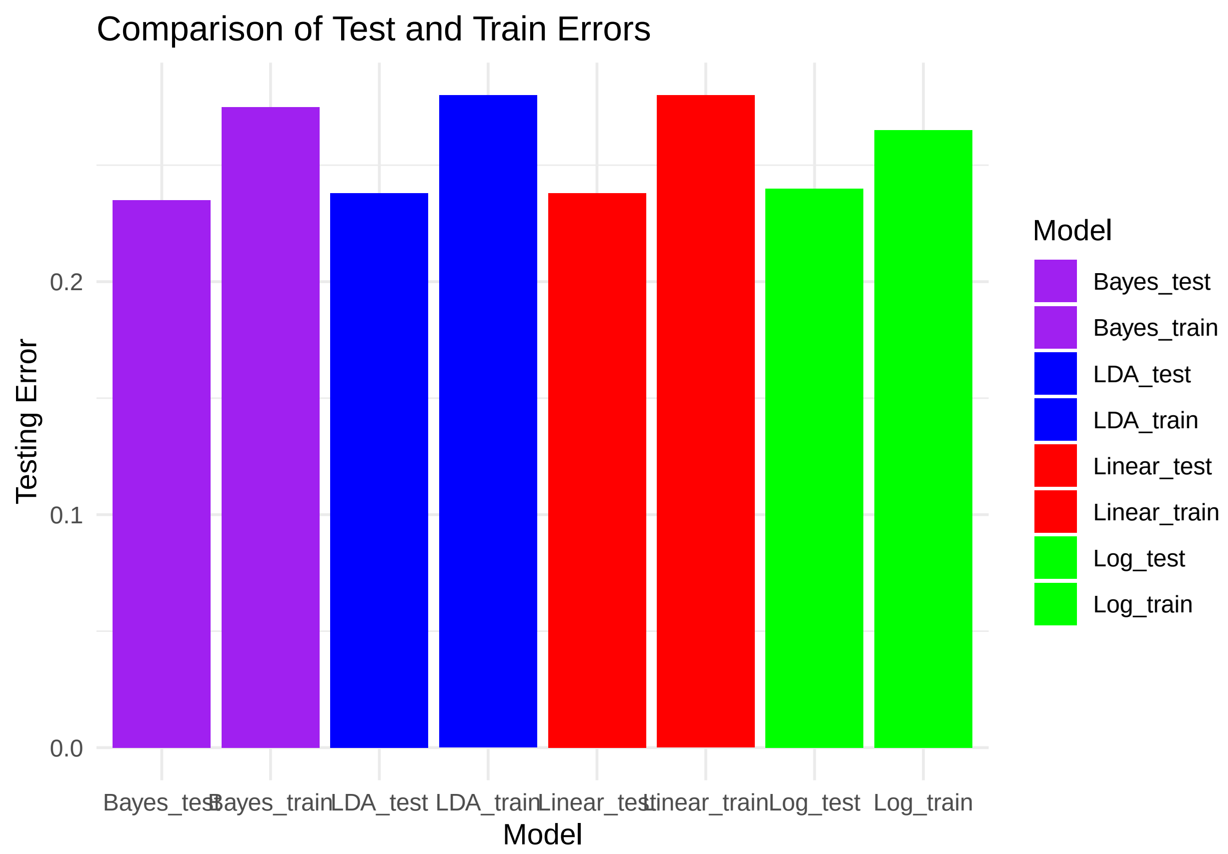
```
cat("Logistic Regression Testing Error:", logistic_testing_error, "\n")
```

```
## Logistic Regression Testing Error: 0.24
```

(c) Compare the prediction performance of the linear regression model, LDA, logistic regression, and the Bayes rule. Provide your comments.

```
# The result from homework#2
bayes_training_error <- 0.275
bayes_testing_error <- 0.235
```

```r
linear_training_error <-0.28
linear_testing_error <-0.238

# Create a vector of testing errors
test_errors <- c(lda_training_error,  logistic_training_error, linear_training_error, bayes_training_er
                 lda_testing_error, logistic_testing_error, linear_testing_error, bayes_testing_error )
model_names <- c("LDA_train", "Log_train", "Linear_train", "Bayes_train",
                 "LDA_test", "Log_test", "Linear_test", "Bayes_test")      # Names for the models
error_data <- data.frame(Model = model_names, Error = test_errors)     # make it a data frame

# Create a bar chart
ggplot(data = error_data, aes(x = Model, y = Error, fill = Model)) +
  geom_bar(stat = "identity") +
  labs(title = "Comparison of Test and Train Errors", x = "Model", y = "Testing Error") +
  scale_fill_manual(values = c("LDA_train" = "blue", "Log_train" = "green", "Linear_train" = "red", "Ba
                               "LDA_test" = "blue", "Log_test" = "green", "Linear_test" = "red", "Bayes
  theme_minimal()
```



Comment: All the four model have almost the same training and testing error. But, Bayes rule has the smallest for testing error while logistic regression has minimum error for training error.

6. (Two-Class Classification Problem: Scenerio 2) (Textbook page 17).

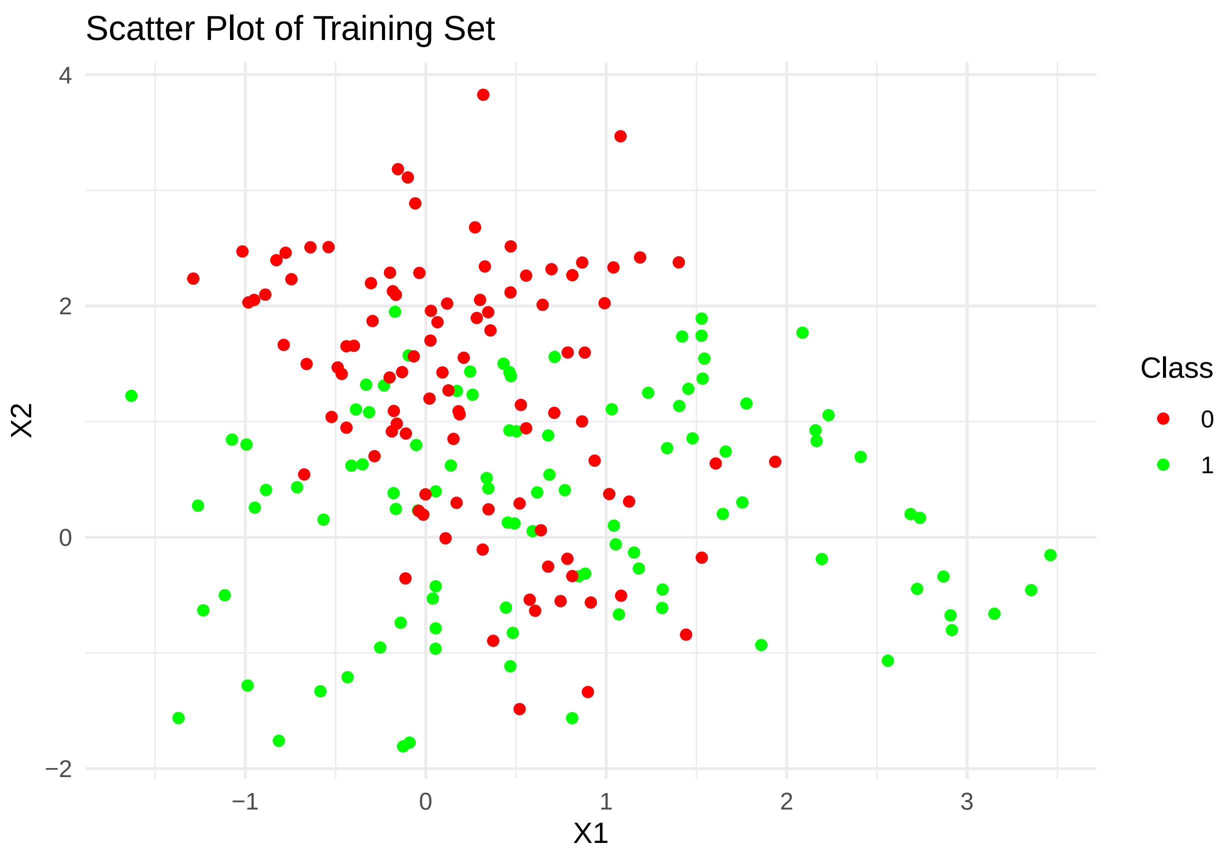
Generate a training set of $n = 200$ from a mixture data as follows.
step 1: Generate 10 points $\mu_k, k = 1, \ldots, 10$ from a bivariate Gaussian distribution $N\left((1,0)^T, \mathbf{I}\right)$. They will be used as means (centers) to generate the Green class for both training and test data.

step 2: Generate 10 points $\nu_k, k = 1, \ldots, 10$ from a bivariate Gaussian distribution $N\left((0,1)^T, \mathbf{I}\right)$. They will be used as means (centers) to generate the Red class.
step 3: For the Green class, generate 100 observations as follows: for each observation, randomly pick a μ_k with probability 1/10, and then generate a point from $N(\mu_k, \mathbf{I}/5)$.
step 4: For the Red class, generate 100 observations as follows: for each observation, randomly pick a ν_k with probability 1/10, and then generate a point from $N(\nu_k, \mathbf{I}/5)$.
(a) Use the following code to generate the training set:

```
# generate ten centers, which are treated as fixed parameters
Sig <- matrix(c(1,0,0,1),nrow=2)
seed_center <- 16
set.seed(seed_center)
center_green <- mvrnorm(n=10,c(1,0),Sig)
center_red <- mvrnorm(n=10,c(0,1),Sig)

# define a function "gendata2" first
gendata2 <-function(n,mu1,mu2,Sig1,Sig2,myseed)
{
set.seed(myseed)
mean1 <- mu1[sample(1:10,n,replace=T),]
mean2 <- mu2[sample(1:10,n,replace=T),]
green <- matrix(0,ncol=2,nrow=n)
red <- matrix(0,ncol=2,nrow=n)
for(i in 1:n){
green[i,] <- mvrnorm(1,mean1[i,],Sig1)
red[i,] <- mvrnorm(1,mean2[i,],Sig2)
}
x <- rbind(green,red)
return(x)
}

# generate the training set
seed_train <- 2000
ntrain <- 100
train2 <- gendata2(ntrain,center_green,center_red,Sig/5,Sig/5,seed_train)
ytrain <- c(rep(1,ntrain),rep(0,ntrain))
```

(b) Draw the scatter plot of the training set, using different labels/colors for two classes.

```
# Create a data frame for the training set
training2 <- data.frame(X1 = train2[, 1], X2 = train2[, 2], Class = factor(ytrain))

# Create a scatter plot with different colors for each class
ggplot(data = training2, aes(x = X1, y = X2, color = Class)) +
  geom_point() +
  labs(title = "Scatter Plot of Training Set",
       x = "X1", y = "X2") +
  scale_color_manual(values = c("1" = "green", "0" = "red")) +
  theme_minimal()
```



(c) Generate a test set, with 500 observations from each class, using set.seed(2014). The same center parameters are used in the training and test sets. Save the test set for future use.

```
# Set the seed for reproducibility
set.seed(2014)

# Generate the test set
seed_test <- 2014
ntest <- 500
test2 <- gendata2(ntest, center_green, center_red, Sig / 5, Sig / 5, seed_test)
ytest <- c(rep(1, ntest), rep(0, ntest))
testing2 <- data.frame(X1 = test2[, 1], X2 = test2[, 2], Class = factor(ytest) )
```

7. (Linear Method for Classification: Scenario 2) For this problem, use the training and test data sets you generated above in the previous question.

(a) Train the linear regression model, using the function " lm(y ∼ x) ‘,’ with the training set. Report the training and test errors.

```
# Fiiting the model for trainind data
# Create a new numeric column based on the "Class" column
training2$ClassNumeric <- ifelse(training2$Class == "1", 1, 0)
testing2$ClassNumeric <- ifelse(testing2$Class == "1", 1, 0)
linear_model2 <- lm(ClassNumeric ~ X1 + X2, data = training2)
```

```r
# Predict class labels for the training and testing data
training2_pred_linear <- ifelse(predict(linear_model2, newdata = training2, type = "response") >= 0.5,
testing2_pred_linear <- ifelse(predict(linear_model2, newdata = testing2, type = "response") >= 0.5, "1

# computing training and test errors for the linear classifier
linear_training2_error <- mean(training2_pred_linear != training2$Class)
linear_testing2_error <- mean(testing2_pred_linear != testing2$Class)

# training and test errors printing
cat("Training2 Error:", linear_training2_error, "\n")
```

```
## Training2 Error: 0.31
```

```r
cat("Test2 Error:", linear_testing2_error, "\n")
```

```
## Test2 Error: 0.295
```

(b) Fit the LDA for the training data, and report the training and testing errors.

```r
# Fit  LDA model, and make predictions on training data
lda_model2 <- lda(Class ~ X1 + X2, data = training2)
training2_pred_lda <- predict(lda_model2, training2)$class
lda_training2_error <- mean(training2_pred_lda != training2$Class)      # Calculate training error

# Make predictions on test data
testing2_pred_lda <- predict(lda_model2, testing2)$class
lda_testing2_error <- mean(testing2_pred_lda != testing2$Class)     # Calculate testing error

# printing results
cat("LDA Training Error:", lda_training2_error, "\n")
```

```
## LDA Training Error: 0.31
```

```r
cat("LDA Testing Error:", lda_testing2_error, "\n")
```

```
## LDA Testing Error: 0.295
```

(c) Fit the logistic regression for the training data, and report the training and testing errors.

```r
# Fit logistic regression model and make predictions on training data
logistic_model2 <- glm(Class ~ X1 + X2, data = training2, family = binomial)
logistic_training2_probabilities <- predict(logistic_model2, type = "response", newdata = training2)
logistic_training2_pred <- ifelse(logistic_training2_probabilities > 0.5, "1", "0")
logistic_training2_error <- mean(logistic_training2_pred != training2$Class) # Calculate training error

# Make predictions on test data
logistic_testing2_probabilities <- predict(logistic_model2, type = "response", newdata = testing2)
logistic_testing2_pred <- ifelse(logistic_testing2_probabilities > 0.5, "1", "0")
logistic_testing2_error <- mean(logistic_testing2_pred != testing2$Class)     # Calculate testing error

# printing results
cat("Logistic Regression Training Error:", logistic_training2_error, "\n")
```

```
## Logistic Regression Training Error: 0.305
```

```r
cat("Logistic Regression Testing Error:", logistic_testing2_error, "\n")
```

```
## Logistic Regression Testing Error: 0.292
```

(d) Compare (a)(b)(c) in terms of their errors and provide comments.

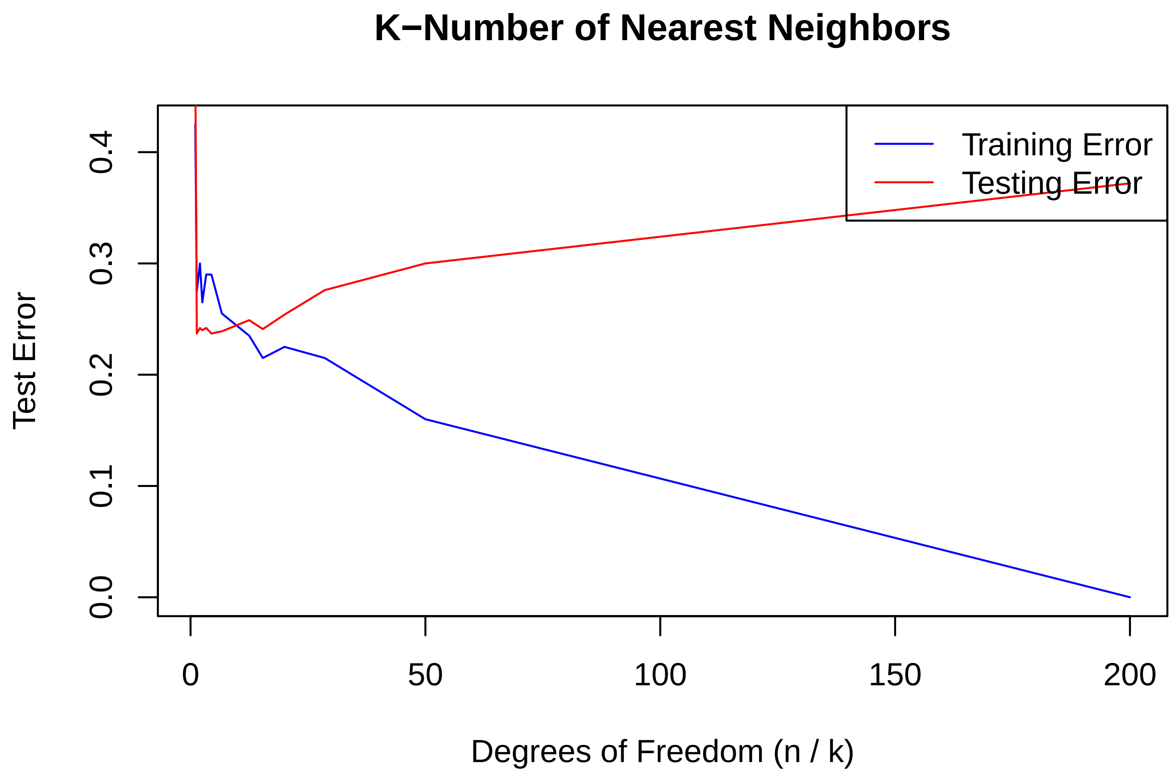
```r
# Create a vector of testing errors
test2_errors <- c(lda_training2_error,  logistic_training2_error, linear_training2_error,
                  lda_testing2_error, logistic_testing2_error, linear_testing2_error )
model2_names <- c("LDA_train", "Log_train", "Linear_train",
                  "LDA_test", "Log_test", "Linear_test")      # Names for the models
error_data2 <- data.frame(Model = model2_names, Error = test2_errors)      # make it a data frame

# Create a bar chart
ggplot(data = error_data2, aes(x = Model, y = Error, fill = Model)) +
  geom_bar(stat = "identity") +
  labs(title = "Comparison of Test and Train Errors", x = "Model", y = "Testing Error") +
  scale_fill_manual(values = c("LDA_train" = "blue", "Log_train" = "green", "Linear_train" = "red",
                               "LDA_test" = "blue", "Log_test" = "green", "Linear_test" = "red")) +
  theme_minimal()
```

Comment: We observed that all the three models have the same testing and training error rate, only the Logistic regression perform slightly better. The error rate are high due to non-linearity of the dataset boundary, which means we need to use another model that can handel non_linearity boundary instead of those three above considered for better result.

8. (k-Nearest Neighbor for Classification)

You may use the functions knn or knn1 in the R library "class" for this problem. Submit your codes along with your results.
(a) Fit k-nearest neighbor classifier with a range of values k for the training data in Scenario 1, $k = \{1, 4, 7, 10, 13, 16, 30, 45, 60, 80, 100, 150, 200\}$. Report both training and testing errors for each k-NN classifier. Plot two curves: the training error vs the degree of freedom n/k, and the testing error vs n/k, in one same figure (Similar to Figure 2.4 in the textbook).

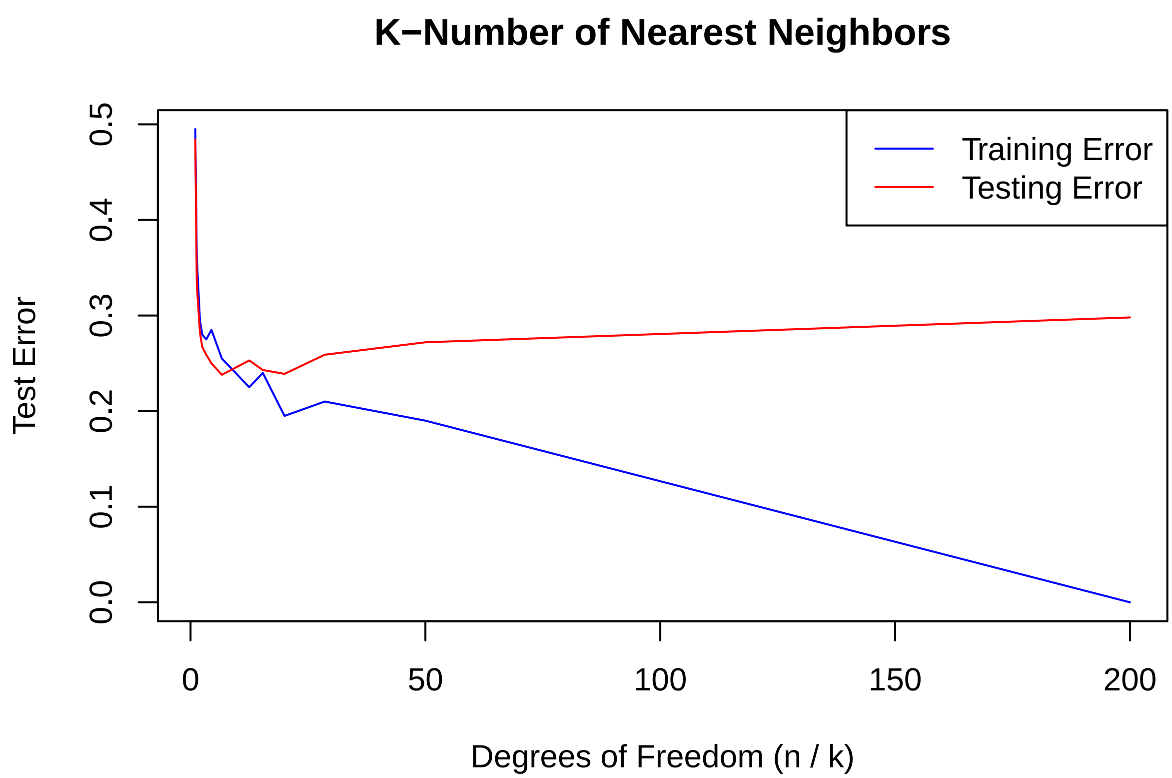
```r
### For Scenario 1
k_values <- c(1, 4, 7, 10, 13, 16, 30, 45, 60, 80, 100, 150, 200)

# Initialize vectors to store training and testing errors
knn_training_errors <- numeric(length(k_values))
knn_testing_errors <- numeric(length(k_values))

# Calculate errors for each k
for (i in 1:length(k_values)) {
  k <- k_values[i]

  # Fit k-NN classifier
  knn_training_pred <- knn(training[, c("X1", "X2")], training[, c("X1", "X2")],
                  training$Class, k)
  knn_testing_pred <- knn(training[, c("X1", "X2")], testing[, c("X1", "X2")],
                  training$Class, k)

  # Calculate training and testing error
  knn_training_errors[i] <- mean(knn_training_pred != training$Class)
  knn_testing_errors[i] <- mean(knn_testing_pred != testing$Class)
}

# Calculate degrees of freedom n/k
degrees_of_freedom <- nrow(training) / k_values

# Create a plot for training and testing errors
plot(degrees_of_freedom, knn_training_errors, type = "l", col = "blue", xlab = "Degrees of Freedom (n /
     ylab = "Test Error", main = "K-Number of Nearest Neighbors")
lines(degrees_of_freedom, knn_testing_errors, type = "l", col = "red")
legend("topright", legend = c("Training Error", "Testing Error"), col = c("blue", "red"), lty = 1)
```



(b) Repeat (a) for Scenario 2.

```r
### For Scenario 2
k_values <- c(1, 4, 7, 10, 13, 16, 30, 45, 60, 80, 100, 150, 200)

# Initialize vectors to store training and testing errors
knn_training2_errors <- numeric(length(k_values))
knn_testing2_errors <- numeric(length(k_values))

# Calculate errors for each k
for (i in 1:length(k_values)) {
  k <- k_values[i]

  # Fit k-NN classifier
  knn_training2_pred <- knn(training2[, c("X1", "X2")], training2[, c("X1", "X2")],
                   training2$Class, k)
  knn_testing2_pred <- knn(training2[, c("X1", "X2")], testing2[, c("X1", "X2")],
                   training2$Class, k)

  # Calculate training and testing error
  knn_training2_errors[i] <- mean(knn_training2_pred != training2$Class)
  knn_testing2_errors[i] <- mean(knn_testing2_pred != testing2$Class)
}
```

```r
# Calculate degrees of freedom n/k
degrees_of_freedom2 <- nrow(training2) / k_values

# Create a plot for training and testing errors
plot(degrees_of_freedom2, knn_training2_errors, type = "l", col = "blue", xlab = "Degrees of Freedom (n
     ylab = "Test Error", main = "K-Number of Nearest Neighbors")
lines(degrees_of_freedom2, knn_testing2_errors, type = "l", col = "red")
legend("topright", legend = c("Training Error", "Testing Error"), col = c("blue", "red"), lty = 1)
```



(c) Based on the plots obtained in (a) and (b), describe the different patterns between the training error and test error curves under each scenario. How should you choose the best k and recommend your best k for each scenario.

Observation: In both scenario, we notice that testing error produces U-shape on the graph, it starts decreasing initially and then stabilizes as the degree of freedom (n/k) decreases (i.e. as k increases) while the training error tends to increase. Moreover, for small values of k, the model overfits the training data, resulting in a low training error but a high testing error, while ss k increases, the model becomes more robust and generalizes better to unseen data, reducing the testing error.

For the best k for each scenario, we need to consider to balancing the trade-off between bias and variance, which involves selecting a k value that corresponds to the point where the testing error reaches its minimum or stabilizes. We can also use techniques like cross-validation to find the optimal k value more systematically. So from the plot, we can see that the best k for scenario1 is 45 (degree of fredom = 4.44) while it is 30 for scenario2.