Statistical Learning Assignment 2

12112627 李乐平

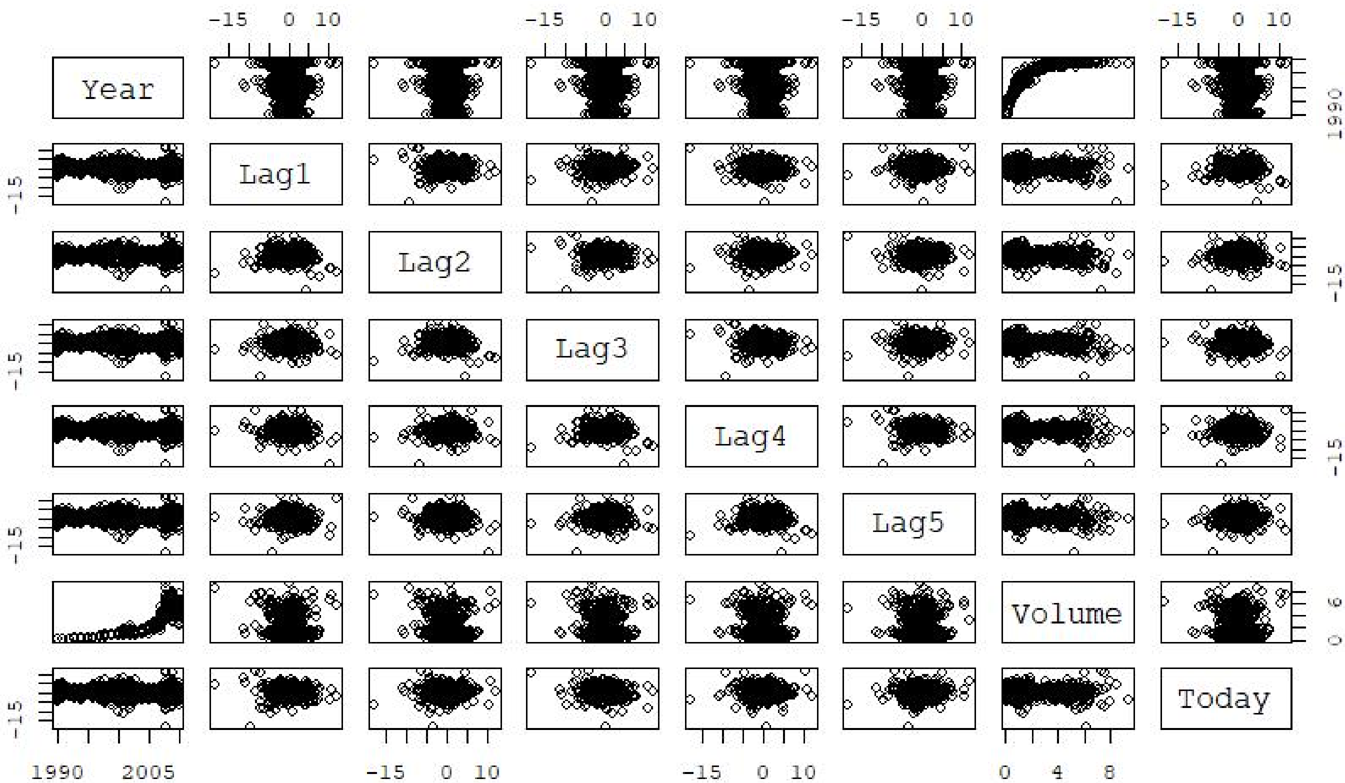
```
# install.packages("ISLR")

library(dplyr)
library(ISLR)
library(ggplot2)
library(MASS)
library(class)
library(e1071)
library(caret)
library(Metrics)
library(pROC)
library(ROCR)
```

Q4.7.10

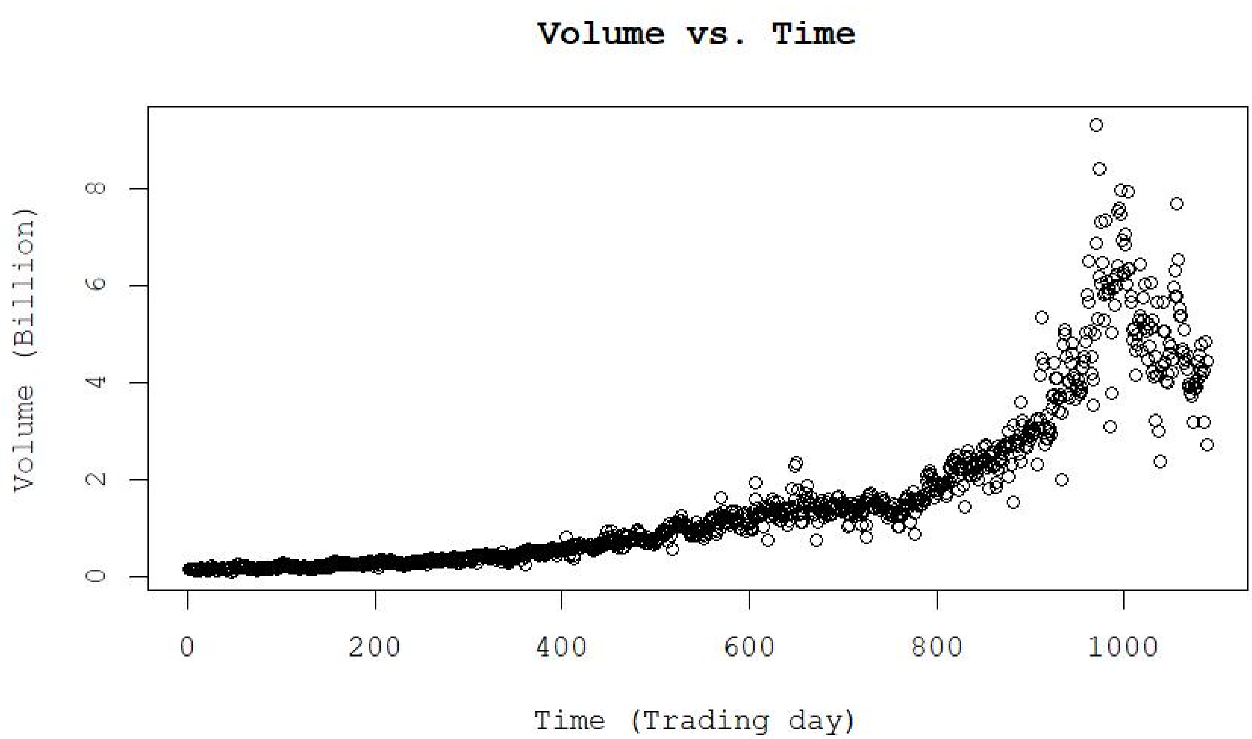
This question should be answered using the Weekly data set, which is part of the ISLR package. This data is similar in nature to the Smarket data from this chapter's lab, except that it contains 1, 089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

(a) Produce some numerical and graphical summaries of the Weekly data. Do there appear to be any patterns?

```
data(Weekly)
glimpse(Weekly)

## Rows: 1,089
## Columns: 9
## $ Year      <dbl> 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, …
## $ Lag1      <dbl> 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0.807, 0…
## $ Lag2      <dbl> 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0…
## $ Lag3      <dbl> -3.936, 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -…
## $ Lag4      <dbl> -0.229, -3.936, 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, …
## $ Lag5      <dbl> -3.484, -0.229, -3.936, 1.572, 0.816, -0.270, -2.576, 3.514,…
## $ Volume    <dbl> 0.1549760, 0.1485740, 0.1598375, 0.1616300, 0.1537280, 0.154…
## $ Today     <dbl> -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0.807, 0.041, 1…
## $ Direction <fct> Down, Down, Up, Up, Up, Down, Up, Up, Up, Down, Down, Up, Up…

par(family = "mono")
pairs(Weekly[, -9])
```




```r
par(family = "mono")
plot(Weekly$Volume, main = "Volume vs. Time", xlab = "Time (Trading day)", ylab = "Volume (Bi
llion)")
```



(b) Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

The predictor Lag2 appears to be statistically significant.

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial(), data = Weekly)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

(c) Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

The confusion matrix shows the performance of a logistic regression model. In this case, it tells us that the model correctly predicted 54 instances of "Down" and 557 instances of "Up." However, it made 430 false negatives (predicted "Up" when it was "Down") and 48 false positives (predicted "Down" when it was "Up").

```r
p = predict(log_res, type = "response")
predicted_direction = ifelse(p > 0.5, "Up", "Down")
confusion_matrix = table(Actual = Weekly$Direction, Predicted = predicted_direction)
confusion_matrix
```

```
##       Predicted
## Actual Down  Up
##   Down   54 430
##   Up     48 557
```

```r
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.5610652
```

(d) Now fit the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).

```r
training_data = Weekly[Weekly$Year <= 2008,]
test_data = Weekly[Weekly$Year > 2008,]
```

```r
model = glm(Direction ~ Lag2, data = training_data, family = binomial)
predicted_probs = predict(model, newdata = test_data, type = "response")
predicted_direction = ifelse(predicted_probs > 0.5, "Up", "Down")

confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
confusion_matrix
```

```
##       Predicted
## Actual Down Up
##   Down    9 34
##   Up      5 56
```

```r
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.625
```

(e) Repeat (d) using LDA.

```r
lda_model = lda(Direction ~ Lag2, data = training_data)
predicted_probs = predict(lda_model, newdata = test_data, type = "response")
predicted_direction = predict(lda_model, newdata = test_data)$class

confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
confusion_matrix
```

```
##       Predicted
## Actual Down Up
##   Down    9 34
##   Up      5 56
```

```r
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.625
```

(f) Repeat (d) using QDA.

```r
qda_model = qda(Direction ~ Lag2, data = training_data)
predicted_probs = predict(qda_model, newdata = test_data, type = "response")
predicted_direction = predict(qda_model, newdata = test_data)$class

confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
confusion_matrix
```

```
##       Predicted
## Actual Down Up
##   Down    0 43
##   Up      0 61
```

```r
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.5865385
```

(g) Repeat (d) using KNN with K = 1.

```
knn_model = knn(train = cbind(training_data$Lag2), test = cbind(test_data$Lag2), cl = traini
ng_data$Direction, k = 1)
confusion_matrix = table(Actual = test_data$Direction, Predicted = knn_model)
confusion_matrix
```

```
##       Predicted
## Actual Down Up
##   Down   21 22
##   Up     30 31
```

```
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.5
```

(h) Which of these methods appears to provide the best results on this data?

By simply comparing accuracies, LDA performs the best.

(i) Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.

7 different combinations are tested as follows, and QDA with Multiple Variable Interactions performs the best, with accuracy of 64.4%.

```
# KNN with K = 3
knn_model = knn(train = cbind(training_data$Lag2), test = cbind(test_data$Lag2), cl = traini
ng_data$Direction, k = 3)
confusion_matrix = table(Actual = test_data$Direction, Predicted = knn_model)
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("KNN with K = 3:\n")
```

```
## KNN with K = 3:
```

```
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.5480769
```

```
print(confusion_matrix)
```

```
##       Predicted
## Actual Down Up
##   Down   15 28
##   Up     19 42
```

```
cat("\n")
```

```
# KNN with K = 5
knn_model = knn(train = cbind(training_data$Lag2), test = cbind(test_data$Lag2), cl = traini
ng_data$Direction, k = 5)
confusion_matrix = table(Actual = test_data$Direction, Predicted = knn_model)
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("KNN with K = 5:\n")
```

```
## KNN with K = 5:
```

```
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")
```

```
## Overall Fraction of Correct Predictions: 0.5288462

print(confusion_matrix)

##       Predicted
## Actual Down Up
##   Down   15 28
##   Up     21 40

cat("\n")

# KNN with K = 10
knn_model = knn(train = cbind(training_data$Lag2), test = cbind(test_data$Lag2), cl = traini
ng_data$Direction, k = 10)
confusion_matrix = table(Actual = test_data$Direction, Predicted = knn_model)
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("KNN with K = 10:\n")

## KNN with K = 10:

cat("Overall Fraction of Correct Predictions:", accuracy, "\n")

## Overall Fraction of Correct Predictions: 0.6057692

print(confusion_matrix)

##       Predicted
## Actual Down Up
##   Down   19 24
##   Up     17 44

cat("\n")

# LDA with Multiple Variables
lda_model = lda(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = training_data)
predicted_probs = predict(lda_model, newdata = test_data, type = "response")
predicted_direction = predict(lda_model, newdata = test_data)$class
confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
cat("LDA with Multiple Variables\n")

## LDA with Multiple Variables

accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")

## Overall Fraction of Correct Predictions: 0.5480769

print(confusion_matrix)

##       Predicted
## Actual Down Up
##   Down    9 34
##   Up     13 48

cat("\n")

# LDA with Multiple Variable Interactions
lda_model = lda(Direction ~ (Lag1 + Lag2 + Lag3 + Lag4 + Lag5)^2, data = training_data)
predicted_probs = predict(lda_model, newdata = test_data, type = "response")
predicted_direction = predict(lda_model, newdata = test_data)$class
```

```r
confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
cat("LDA with Multiple Variable Interactions\n")

## LDA with Multiple Variable Interactions

accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")

## Overall Fraction of Correct Predictions: 0.5288462

print(confusion_matrix)

##       Predicted
## Actual Down Up
##   Down   10 33
##   Up     16 45

cat("\n")

# QDA with Multiple Variables
qda_model = qda(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = training_data)
predicted_probs = predict(qda_model, newdata = test_data, type = "response")
predicted_direction = predict(qda_model, newdata = test_data)$class
confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
cat("QDA with Multiple Variables\n")

## QDA with Multiple Variables

accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")

## Overall Fraction of Correct Predictions: 0.4615385

print(confusion_matrix)

##       Predicted
## Actual Down Up
##   Down   10 33
##   Up     23 38

cat("\n")

# QDA with Multiple Variable Interactions
qda_model = qda(Direction ~ (Lag1 + Lag2 + Lag3 + Lag4 + Lag5)^2, data = training_data)
predicted_probs = predict(qda_model, newdata = test_data, type = "response")
predicted_direction = predict(qda_model, newdata = test_data)$class
confusion_matrix = table(Actual = test_data$Direction, Predicted = predicted_direction)
cat("QDA with Multiple Variable Interactions\n")

## QDA with Multiple Variable Interactions

accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Overall Fraction of Correct Predictions:", accuracy, "\n")

## Overall Fraction of Correct Predictions: 0.6442308

print(confusion_matrix)

##       Predicted
## Actual Down Up
```

```
##   Down   19 24
##   Up     13 48

cat("\n")
```

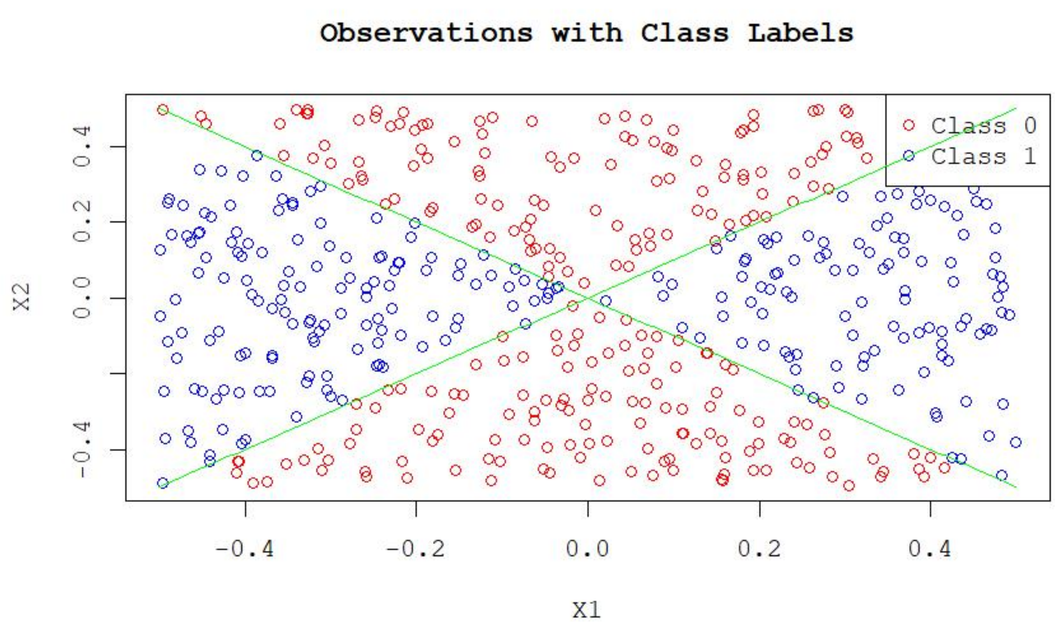
Q9.7.5 We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(a) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows:

```
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y=1*(x1^2 - x2^2 > 0)

set.seed(114514)

n = 500
x1 = runif(n) - 0.5
x2 = runif(n) - 0.5

y = 1 * (x1^2 - x2^2 > 0)

data = data.frame(x1, x2, y)
```

(b) Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the y-axis.

```
par(family = "mono")
plot(x1, x2, col = ifelse(y == 1, "blue", "red"), xlab = "X1", ylab = "X2", main = "Observati
ons with Class Labels")
legend("topright", legend = c("Class 0", "Class 1"), col = c("red", "blue"), pch = 1)

curve(1*x, from = -0.5, to = 0.5, add = TRUE, col = "green")
curve(-1*x, from = -0.5, to = 0.5, add = TRUE, col = "green")
```



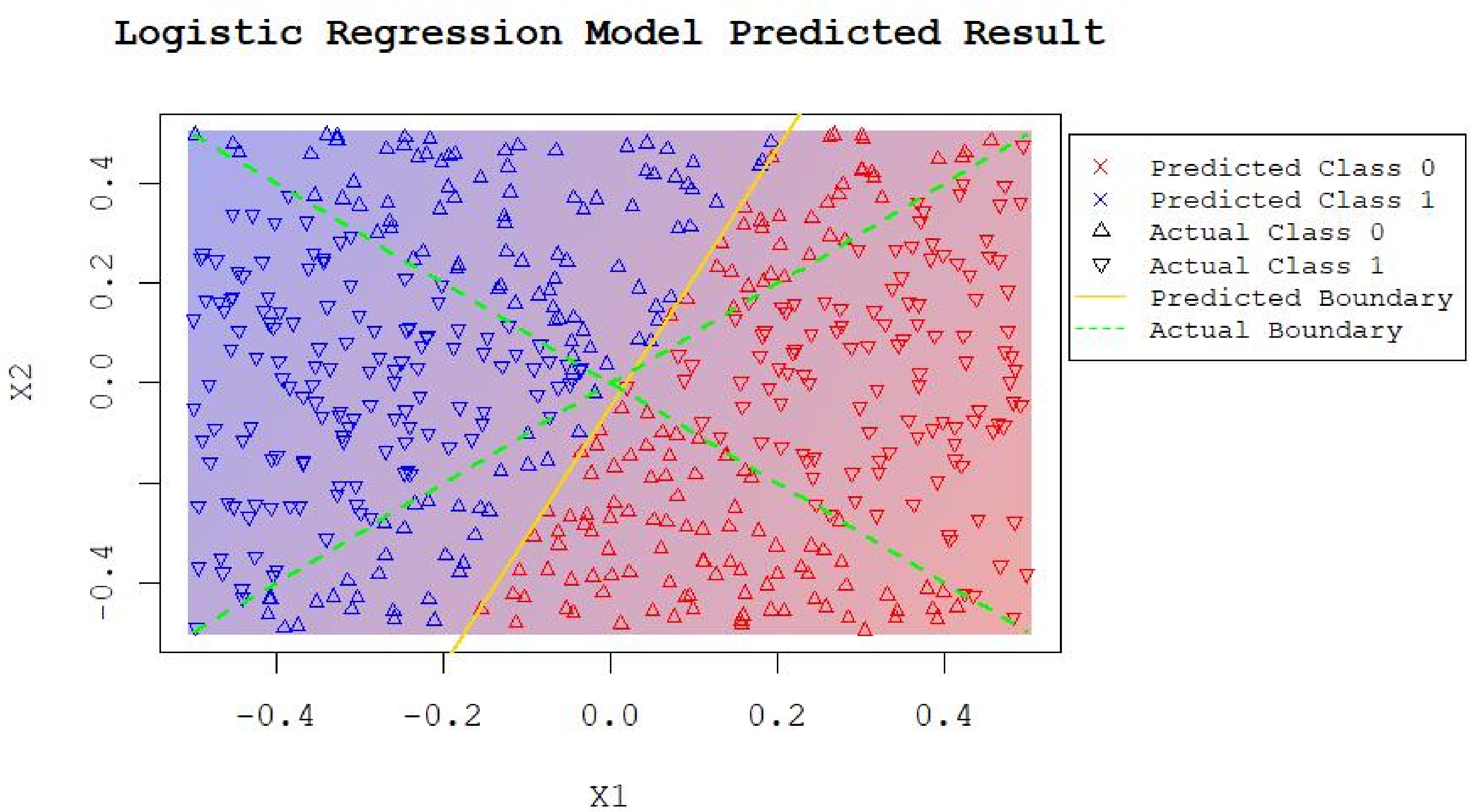
(c) Fit a logistic regression model to the data, using X1 and X2 as predictors.

```
model = glm(y ~ x1 + x2, data = data, family = binomial)

summary(model)

##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial, data = data)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.005622   0.089578   0.063    0.950
## x1          -0.316573   0.313418  -1.010    0.312
## x2           0.122073   0.315136   0.387    0.698
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 693.14  on 499  degrees of freedom
## Residual deviance: 691.98  on 497  degrees of freedom
## AIC: 697.98
##
## Number of Fisher Scoring iterations: 3
```

(d) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```
data$predicted_class = predict(model, newdata = data, type = "response") >= 0.5

x1_seq = seq(-0.5, 0.5, length.out = 100)
x2_seq = seq(-0.5, 0.5, length.out = 100)
boundary = matrix(0, nrow = 100, ncol = 100)
for (i in 1:100) {
  for (j in 1:100) {
    boundary[i, j] <- predict(model, newdata = data.frame(x1 = x1_seq[i], x2 = x2_seq[j], typ
e = "response"))
  }
}
par(oma = c(0, 0, 0, 8))
par(family = "mono")
plot(0, type = "n", xlim = range(x1_seq), ylim = range(x2_seq), xlab = "X1", ylab = "X2", mai
n = "Logistic Regression Model Predicted Result")

custom_palette = colorRampPalette(c("#EEAAAA", "#AAAAEE"))
image(x1_seq, x2_seq, boundary, col = custom_palette(50), add = TRUE)

points(data$x1, data$x2, col = ifelse(data$predicted_class == 1, "blue", "red"),
       pch = y * 4 + 2, cex = 0.7)
abline(coef(model)[1] / -coef(model)[3], coef(model)[2] / -coef(model)[3], col = "gold", lwd
 = 2)
curve(1*x, from = -0.5, to = 0.5, add = TRUE, col = "green",lty = 2, lwd = 2)
curve(-1*x, from = -0.5, to = 0.5, add = TRUE, col = "green", lty = 2, lwd = 2)
par(xpd = NA)

legend(x = 0.55, y = 0.5,
       legend = c("Predicted Class 0", "Predicted Class 1",
                  "Actual Class 0", "Actual Class 1", "Predicted Boundary", "Actual Boundary"),
```

```
       col = c("red", "blue", "black", "black", "gold", "green"),
       pch = c(4, 4, 2, 6, NA, NA),
       lwd = c(NA, NA, NA, NA, 1, 1),
       lty = c(1, 1, 1, 1, 1, 2),
       cex = 0.8
       )
```



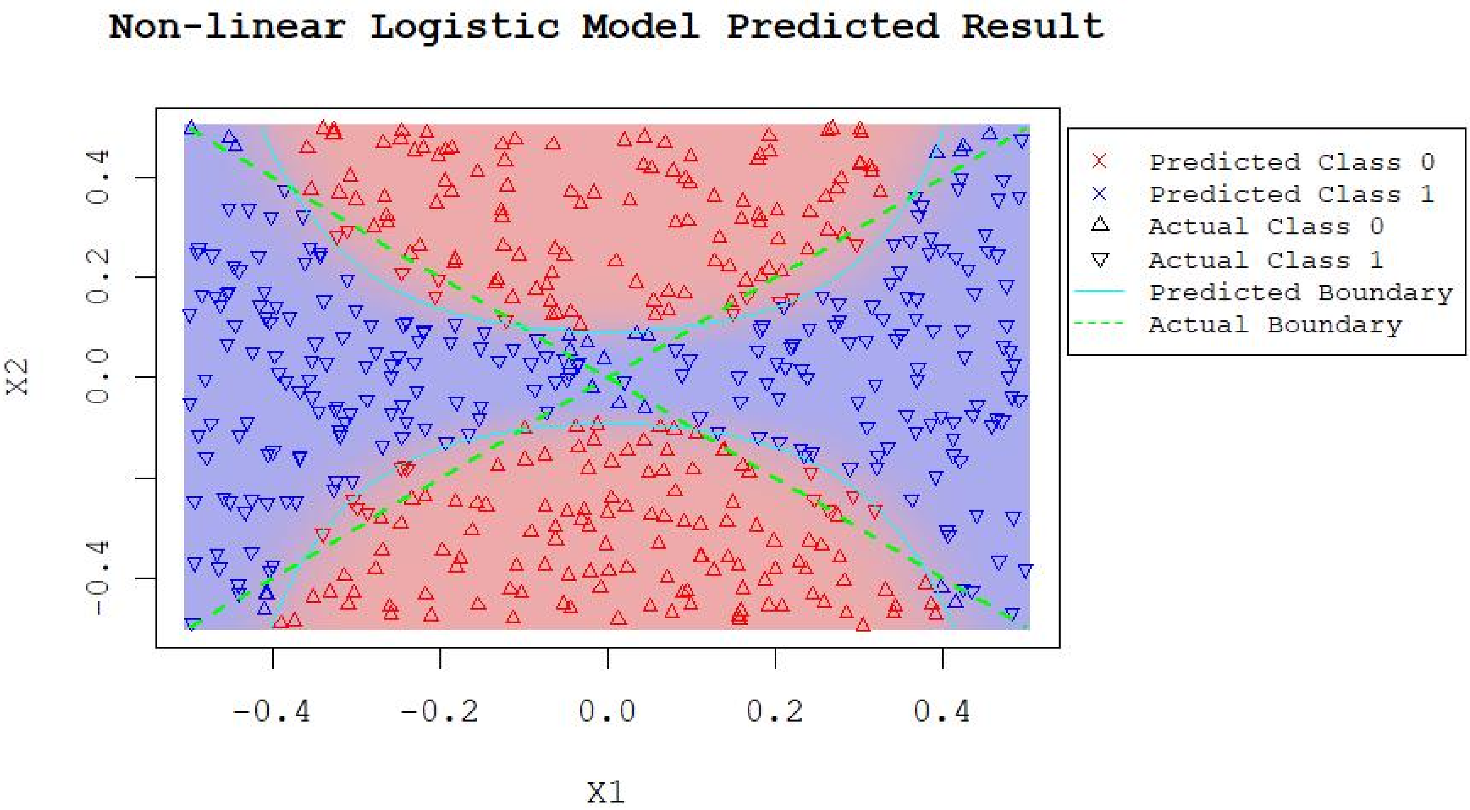
(e) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X_1^2 , X1×X2, log(X2), and so forth).

```
data$X1_squared <- data$x1^2
data$X1_times_X2 <- data$x1 * data$x2
data$log_X2_squared <- log(data$x2^2)

model_nonlinear <- glm(y ~ X1_squared + X1_times_X2 + log_X2_squared, data = data, family = b
inomial)

data$predicted_class = predict(model_nonlinear, newdata = data, type = "response") >= 0.5

summary(model_nonlinear)

##
## Call:
## glm(formula = y ~ X1_squared + X1_times_X2 + log_X2_squared,
##     family = binomial, data = data)
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -11.2188     1.1471  -9.780   <2e-16 ***
```

```
## X1_squared      47.7221     4.9685   9.605   <2e-16 ***
## X1_times_X2      0.9873     1.8845   0.524      0.6
## log_X2_squared  -2.3545     0.2555  -9.214   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 693.14  on 499  degrees of freedom
## Residual deviance: 208.35  on 496  degrees of freedom
## AIC: 216.35
##
## Number of Fisher Scoring iterations: 7
```

(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

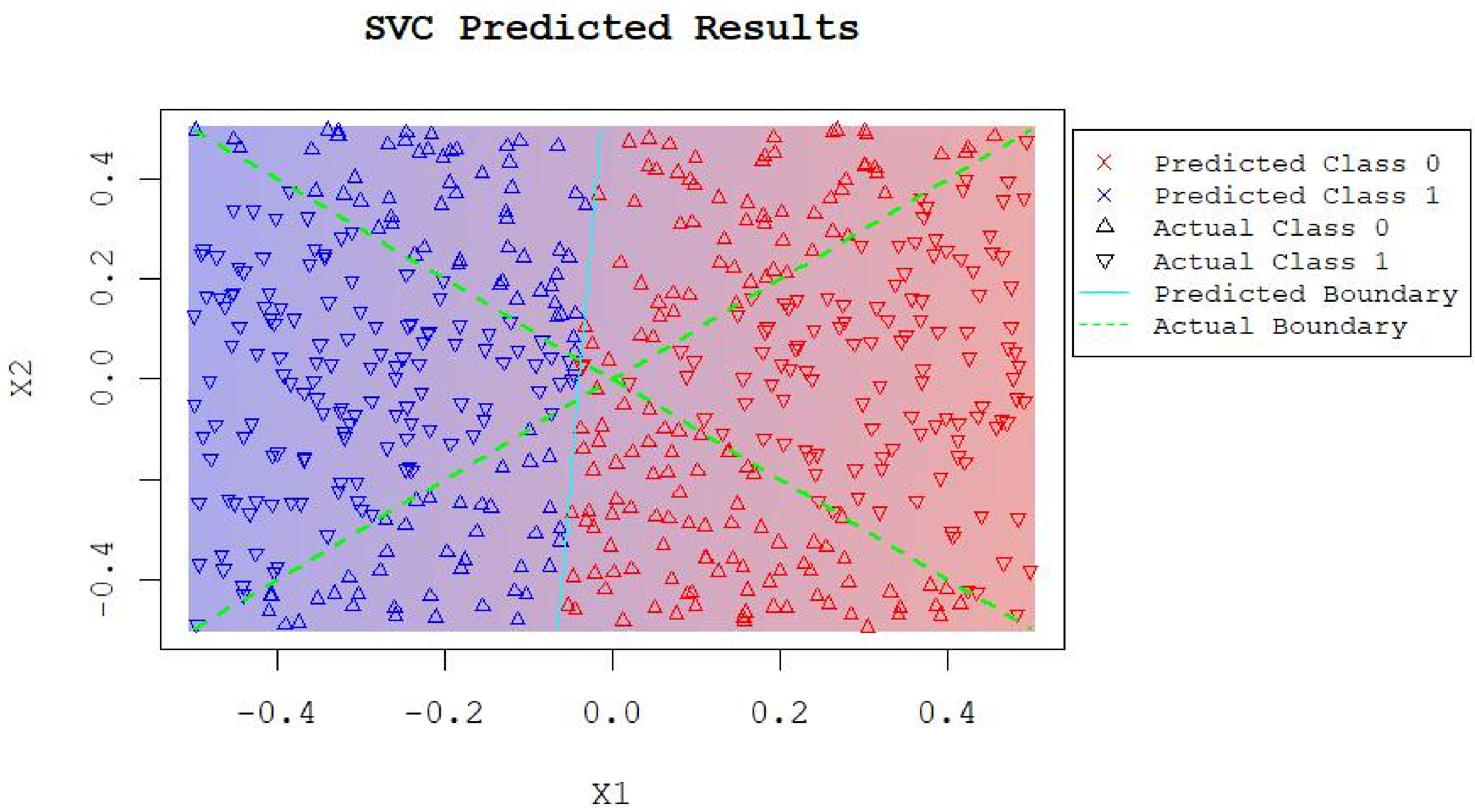
```
x1_seq = seq(-0.5, 0.5, length.out = 100)
x2_seq = seq(-0.5, 0.5, length.out = 100)
boundary = matrix(0, nrow = 100, ncol = 100)
for (i in 1:100) {
  for (j in 1:100) {
    boundary[i, j] <- predict(model_nonlinear, newdata = data.frame(X1_squared = x1_seq[i]^2,
 X1_times_X2 = x1_seq[i] * x2_seq[j], log_X2_squared = log(x2_seq[j]^2)), type = "response")
  }
}
par(oma = c(0, 0, 0, 8))
par(family = "mono")
plot(0, type = "n", xlim = range(x1_seq), ylim = range(x2_seq), xlab = "X1", ylab = "X2", mai
n = "Non-linear Logistic Model Predicted Result")

custom_palette = colorRampPalette(c("#EEAAAA", "#AAAAEE"))
image(x1_seq, x2_seq, boundary, col = custom_palette(50), add = TRUE)

points(data$x1, data$x2, col = ifelse(data$predicted_class == 1, "blue", "red"),
       pch = y * 4 + 2, cex = 0.7)

curve(1*x, from = -0.5, to = 0.5, add = TRUE, col = "green",lty = 2, lwd = 2)
curve(-1*x, from = -0.5, to = 0.5, add = TRUE, col = "green", lty = 2, lwd = 2)
contour(x1_seq, x2_seq, boundary, levels = 0.5, drawlabels = FALSE, col = "cyan", add = TRUE)
par(xpd = NA)

legend(x = 0.55, y = 0.5,
       legend = c("Predicted Class 0", "Predicted Class 1",
                  "Actual Class 0", "Actual Class 1", "Predicted Boundary", "Actual Boundary"),

       col = c("red", "blue", "black", "black", "cyan", "green"),
       pch = c(4, 4, 2, 6, NA, NA),
       lwd = c(NA, NA, NA, NA, 1, 1),
       lty = c(1, 1, 1, 1, 1, 2),
       cex = 0.8
       )
```



(g) Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
svm_model <- svm(y ~ ., data = data.frame(data[, c("x1", "x2")]), kernel = "linear", cost = 5,
 scale = FALSE)

data$predicted_svm <- predict(svm_model, data.frame(data[, c("x1", "x2")]), type = "response
") >= 0.5

x1_seq = seq(-0.5, 0.5, length.out = 100)
x2_seq = seq(-0.5, 0.5, length.out = 100)
boundary = matrix(0, nrow = 100, ncol = 100)
for (i in 1:100) {
  for (j in 1:100) {
    boundary[i, j] <- predict(svm_model, newdata = data.frame(x1 = x1_seq[i], x2 = x2_seq[j]),
 type = "response")
  }
}
par(oma = c(0, 0, 0, 8))
par(family = "mono")
plot(0, type = "n", xlim = range(x1_seq), ylim = range(x2_seq), xlab = "X1", ylab = "X2", mai
n = "SVC Predicted Results")

custom_palette = colorRampPalette(c("#EEAAAA", "#AAAAEE"))
image(x1_seq, x2_seq, boundary, col = custom_palette(50), add = TRUE)

points(data$x1, data$x2, col = ifelse(data$predicted_svm == 1, "blue", "red"),
       pch = y * 4 + 2, cex = 0.7)
```

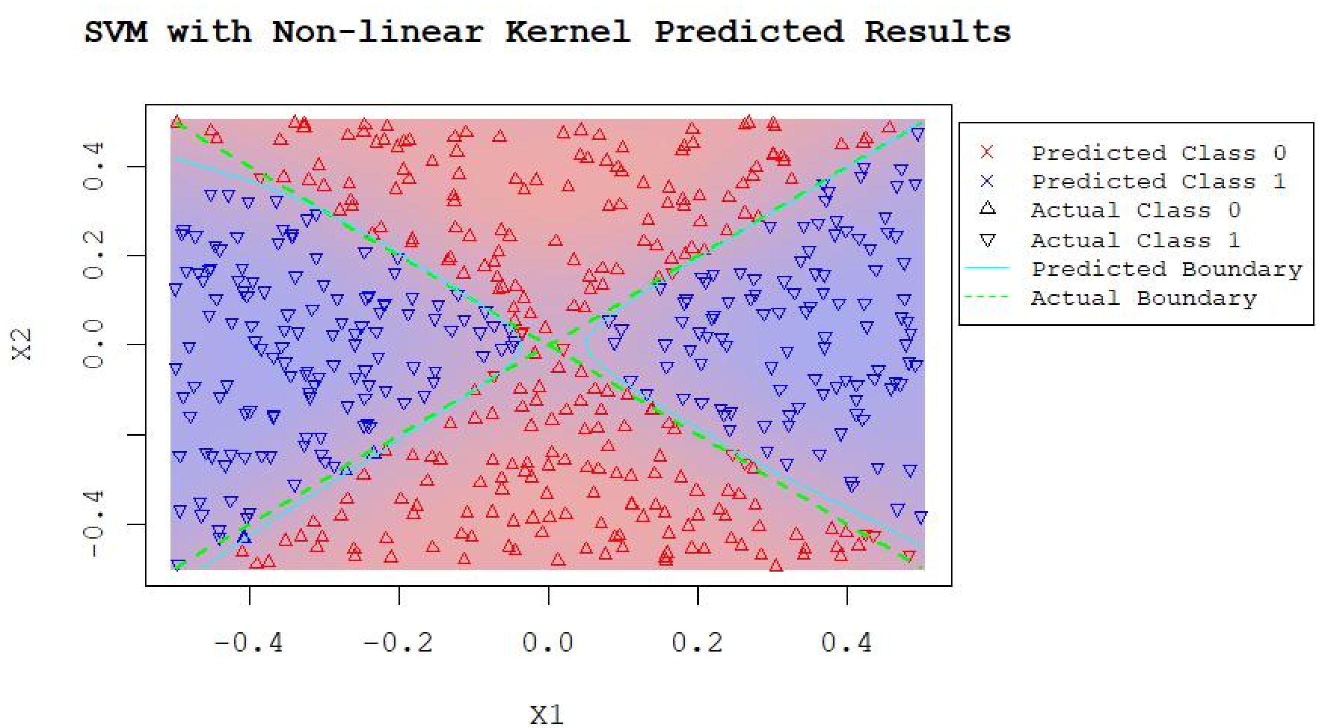
```
curve(1*x, from = -0.5, to = 0.5, add = TRUE, col = "green",lty = 2, lwd = 2)
curve(-1*x, from = -0.5, to = 0.5, add = TRUE, col = "green", lty = 2, lwd = 2)
contour(x1_seq, x2_seq, boundary, levels = 0.5, drawlabels = FALSE, col = "cyan", add = TRUE)
par(xpd = NA)

legend(x = 0.55, y = 0.5,
       legend = c("Predicted Class 0", "Predicted Class 1",
                  "Actual Class 0", "Actual Class 1", "Predicted Boundary", "Actual Boundary"),

       col = c("red", "blue", "black", "black", "cyan", "green"),
       pch = c(4, 4, 2, 6, NA, NA),
       lwd = c(NA, NA, NA, NA, 1, 1),
       lty = c(1, 1, 1, 1, 1, 2),
       cex = 0.8
       )
```



(h) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
X = data[, c("x1", "x2")]

svm_model = svm(y ~ ., data = data.frame(X, y), kernel = "radial")

predicted_classes <- predict(svm_model, data.frame(X)) >= 0.5

x1_seq = seq(-0.5, 0.5, length.out = 100)
x2_seq = seq(-0.5, 0.5, length.out = 100)
boundary = matrix(0, nrow = 100, ncol = 100)
```

```
for (i in 1:100) {
  for (j in 1:100) {
    boundary[i, j] <- predict(svm_model, newdata = data.frame(x1 = x1_seq[i], x2 = x2_seq[j]),
 type = "response")
  }
}
par(oma = c(0, 0, 0, 8))
par(family = "mono")
plot(0, type = "n", xlim = range(x1_seq), ylim = range(x2_seq), xlab = "X1", ylab = "X2", mai
n = "SVM with Non-linear Kernel Predicted Results")

custom_palette = colorRampPalette(c("#EEAAAA", "#AAAAEE"))
image(x1_seq, x2_seq, boundary, col = custom_palette(50), add = TRUE)

points(data$x1, data$x2, col = ifelse(predicted_classes == 1, "blue", "red"),
       pch = y * 4 + 2, cex = 0.7)

curve(1*x, from = -0.5, to = 0.5, add = TRUE, col = "green",lty = 2, lwd = 2)
curve(-1*x, from = -0.5, to = 0.5, add = TRUE, col = "green", lty = 2, lwd = 2)
contour(x1_seq, x2_seq, boundary, levels = 0.5, drawlabels = FALSE, col = "cyan", add = TRUE)
par(xpd = NA)

legend(x = 0.55, y = 0.5,
       legend = c("Predicted Class 0", "Predicted Class 1",
                  "Actual Class 0", "Actual Class 1", "Predicted Boundary", "Actual Boundary"),

       col = c("red", "blue", "black", "black", "cyan", "green"),
       pch = c(4, 4, 2, 6, NA, NA),
       lwd = c(NA, NA, NA, NA, 1, 1),
       lty = c(1, 1, 1, 1, 1, 2),
       cex = 0.8
       )
```



(i) Comment on your results.

All non-linear models applied in this experiment work well, while linear models cannot fit the quadratic boundary well. Modifications of import non-linear kernel is significant to the quadratic boundary fitting.

ISLR Section 9.3.3 Reproduced

```
heart_data = read.csv("Heart.csv")

heart_data = na.omit(heart_data)

heart_data$AHD[heart_data$AHD == "Yes"] = 1
heart_data$AHD[heart_data$AHD == "No"] = 0
heart_data$AHD = as.numeric(heart_data$AHD)


set.seed(114514)

train_indices = sample(1 : nrow(heart_data), 207)
train_data = heart_data[train_indices, ]
test_data = heart_data[-train_indices, ]

lda_model = lda(AHD ~ .-X, data = train_data)

svm_linear_model = svm(AHD~.-X, data = train_data, kernel = "polynomial", d = 1, cost = 1e-1,
 scale = TRUE)
svm_k1_model = svm(AHD~.-X, data = train_data, kernel = "radial", gamma = 0.1, cost = 1e-1, s
cale = TRUE)
svm_k2_model = svm(AHD~.-X, data = train_data, kernel = "radial", gamma = 0.01, cost = 1e-1,
scale = TRUE)
svm_k3_model = svm(AHD~.-X, data = train_data, kernel = "radial", gamma = 0.001, cost = 1e-1,
 scale = TRUE)

par(mfrow = c(1, 2))
par(family = "mono")
lda_prob = predict(lda_model, test_data, type = "response")
lda_pred = prediction(lda_prob$posterior[, 2], test_data$AHD)
lda_performance = performance(lda_pred, measure = "tpr", x.measure = "fpr")

svm_linear_prob = predict(svm_linear_model, test_data, type = "response")
svm_linear_prediction = prediction(svm_linear_prob, test_data$AHD)
svm_linear_performance = performance(svm_linear_prediction, measure = "tpr", x.measure = "fp
r")

svm_k1_prob = predict(svm_k1_model, test_data, type = "response")
svm_k1_prediction = prediction(svm_k1_prob, test_data$AHD)
svm_k1_performance = performance(svm_k1_prediction, measure = "tpr", x.measure = "fpr")

svm_k2_prob = predict(svm_k2_model, test_data, type = "response")
svm_k2_prediction = prediction(svm_k2_prob, test_data$AHD)
svm_k2_performance = performance(svm_k2_prediction, measure = "tpr", x.measure = "fpr")

svm_k3_prob = predict(svm_k3_model, test_data, type = "response")
svm_k3_prediction = prediction(svm_k3_prob, test_data$AHD)
svm_k3_performance = performance(svm_k3_prediction, measure = "tpr", x.measure = "fpr")
```

```
plot(lda_performance, col = "darkblue")
plot(svm_linear_performance, col = "red", add = TRUE)
legend("bottomright", legend = c("SVC", "LDA"), fill = c("red", "darkblue"))

plot(svm_linear_performance, col = "red")
plot(svm_k3_performance, col = "black", add = TRUE)
plot(svm_k2_performance, col = "#00AA00", add = TRUE)
plot(svm_k1_performance, col = "darkblue", add = TRUE)
legend("bottomright", legend = c("SVC", "SVM: γ=1e-3", "SVM: γ=1e-2", "SVM: γ=1e-1"), fill =
 c("red", "darkblue", "#00AA00", "black"))
```