Stats 101C HW 7

Charles Liu (304804942)

12/2/2020

Loading Necessary Packages

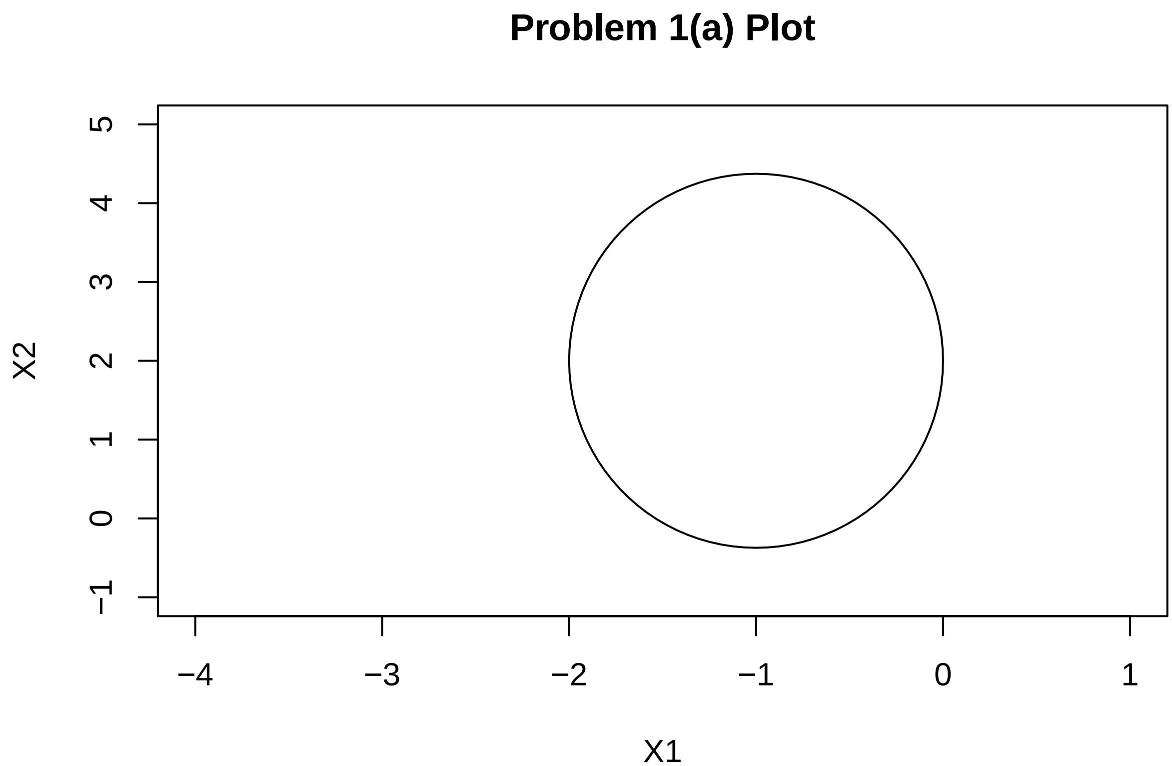
```
library(ISLR)
library(plotrix)
library(e1071)
```

Problem 1 (Exercise 9.7.2)

We have seen that in p = 2 dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

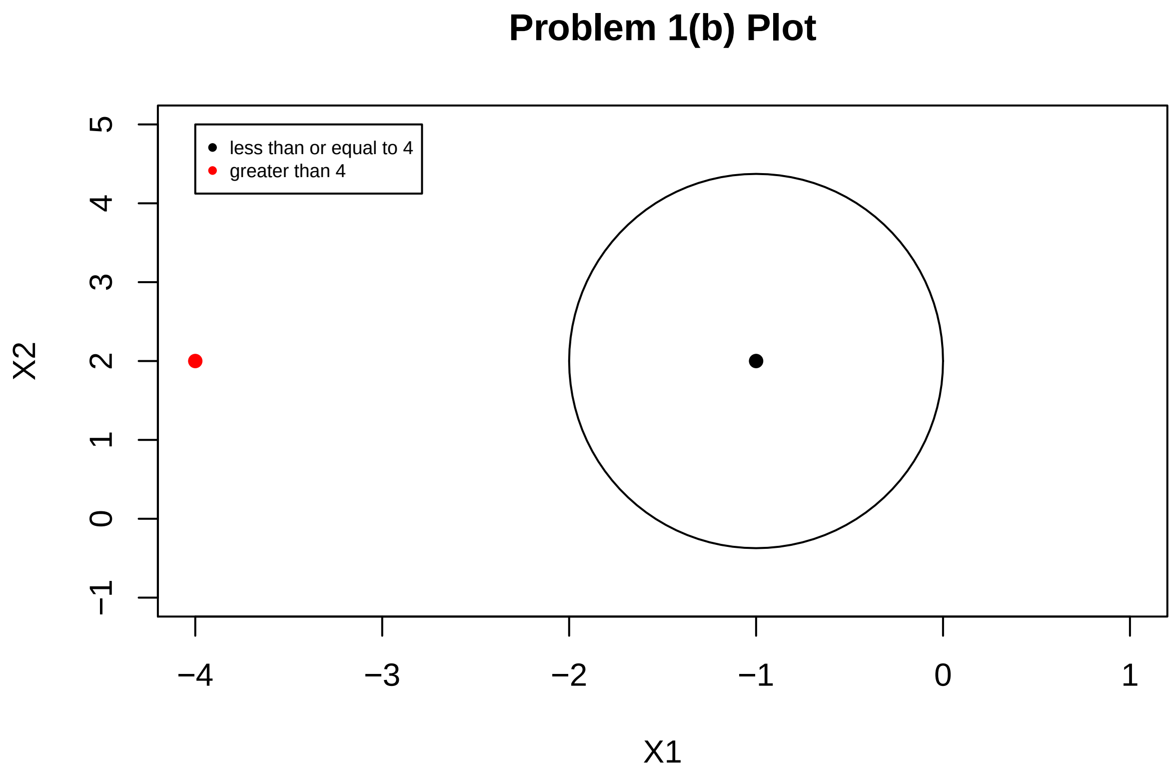
(a) Sketch the curve $(1 + X_1)^2 + (2 - X_2)^2 = 4$.

```
plot(NA, NA, xlim = c(-4, 1), ylim = c(-1, 5),
     main = "Problem 1(a) Plot", xlab = "X1", ylab = "X2")
draw.circle(x = -1, y = 2, radius = 1)
```



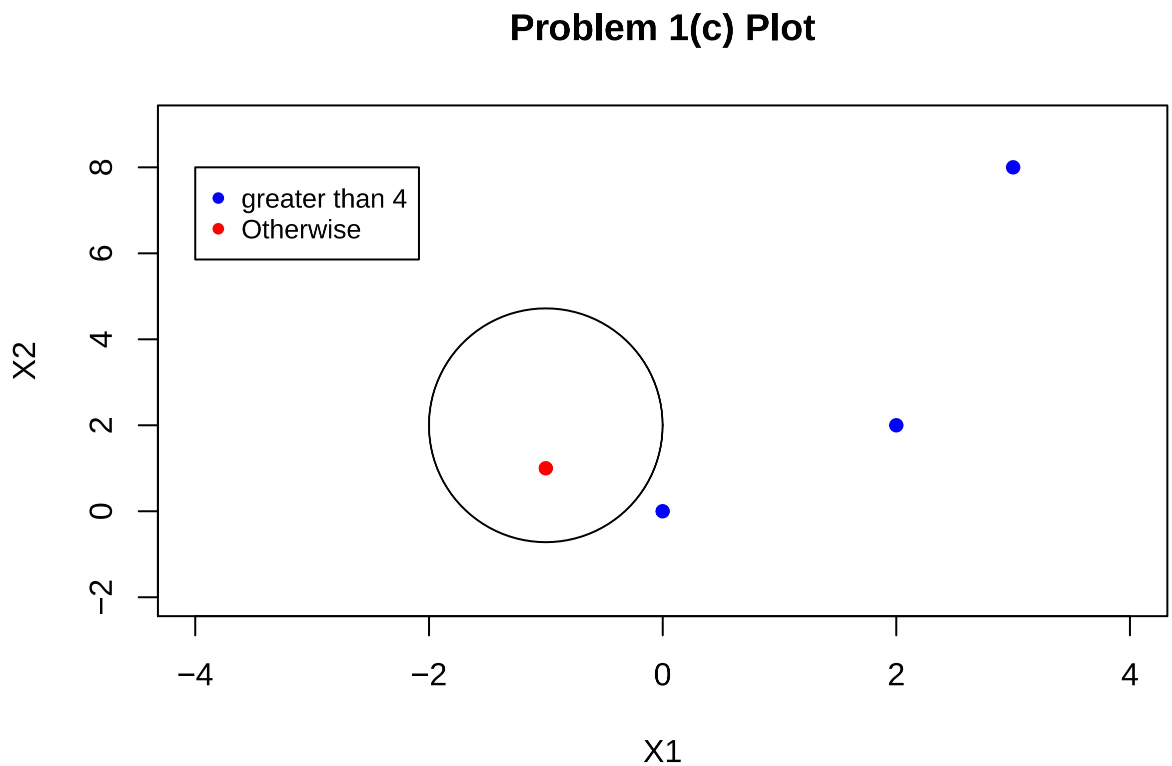
(b) On your sketch, indicate the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 > 4$, as well as the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 \leq 4$.

```
plot(NA, NA, xlim = c(-4, 1), ylim = c(-1, 5),
     main = "Problem 1(b) Plot", xlab = "X1", ylab = "X2")
draw.circle(x = -1, y = 2, radius = 1)
points(-1, 2, pch = 16)
points(-4, 2, pch = 16, col = "red")
legend(-4, 5, legend = c("less than or equal to 4", "greater than 4"),
       pch = 16, col = c("black", "red"), cex = 0.6)
```



(c) Suppose that a classifier assigns an observation to the blue class if $(1 + X_1)^2 + (2 - X_2)^2 > 4$, and to the red class otherwise. To what class is the observation (0, 0) classified? (−1, 1)? (2, 2)? (3, 8)?

```r
# (0,0) = 5 -> blue
# (-1,1) = 1 -> red
# (2,2) = 9 -> blue
# (3,8) = 52 -> blue
plot(NA, NA, xlim = c(-4, 4), ylim = c(-2, 9),
     main = "Problem 1(c) Plot", xlab = "X1", ylab = "X2")
draw.circle(x = -1, y = 2, radius = 1)
points(0, 0, pch = 16, col = "blue")
points(-1, 1, pch = 16, col = "red")
points(2, 2, pch = 16, col = "blue")
points(3, 8, pch = 16, col = "blue")
legend(-4, 8, legend = c("greater than 4", "Otherwise"),
       pch = 16, col = c("blue", "red"), cex = 0.8)
```



(d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.

ANSWER: We can simply do this by expanding on the equation from $(1 + X_1)^2 + (2 - X_2)^2 = 4$ to $X_1^2 + X_2^2 + 2X_1 - 4X_2 + 5 = 4$ and finally we get $X_1^2 + X_2^2 + 2X_1 - 4X_2 + 1 = 0$ (expanded). This shows us that the expanded equation is linear and has terms X_1, X_1^2, X_2, and X_2^2.

Problem 2 (Exercise 9.7.5)

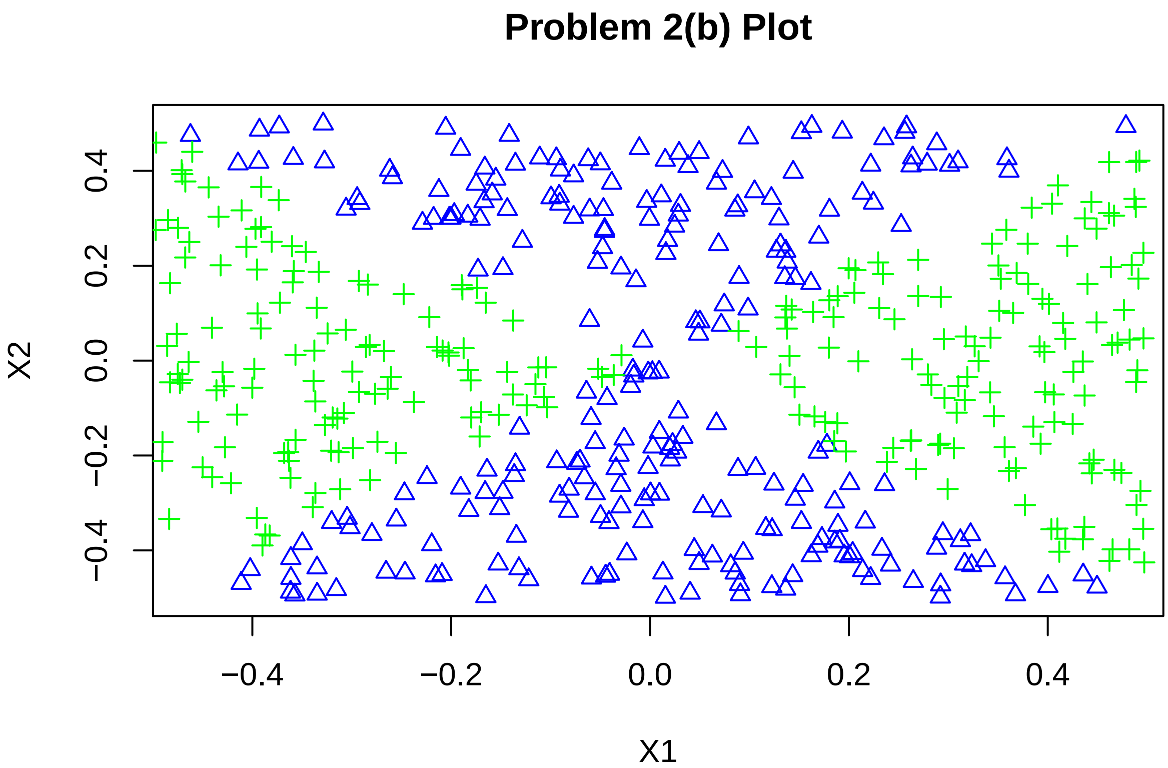
We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary.We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(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)
```

(b) Plot the observations, colored according to their class labels. Your plot should display X_1 on the x-axis, and X_2 on the y-axis.

```r
plot(x1[y == 0], x2[y == 0],
     main = "Problem 2(b) Plot", xlab = "X1", ylab = "X2",
     col = "blue", pch = 2)
points(x1[y == 1], x2[y == 1], col = "green", pch = 3)
```



(c) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

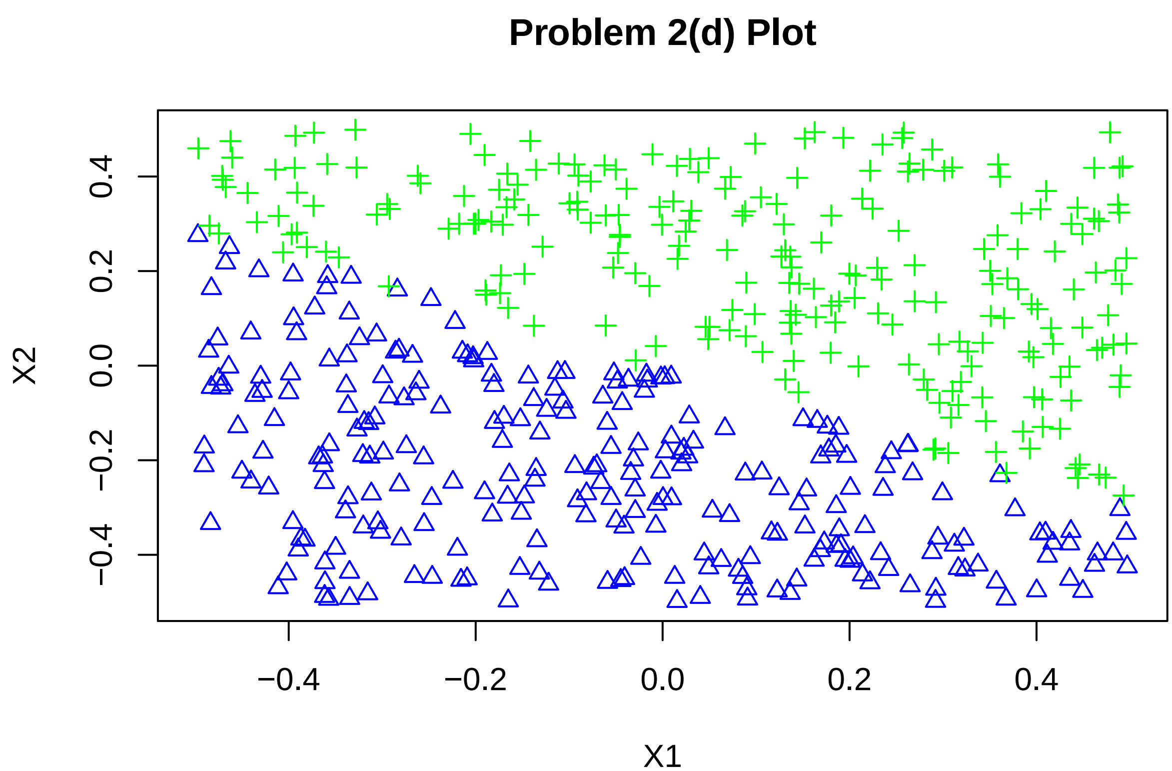
```r
lm_2c <- glm(y ~ x1 + x2, family = "binomial")
summary(lm_2c)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.32303  -1.15791  -0.00149   1.17699   1.29056
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.003541   0.089794   0.039    0.969
## x1          0.255359   0.313160   0.815    0.415
## x2          0.426042   0.313975   1.357    0.175
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 693.15  on 499  degrees of freedom
## Residual deviance: 690.75  on 497  degrees of freedom
## AIC: 696.75
##
## 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.

```r
train_df <- data.frame(x1 = x1, x2 = x2, y = y)
lm_prob <- predict(lm_2c, train_df, type = "response")
lm_predict <- ifelse(lm_prob > 0.5, 1, 0)
lm_one <- train_df[lm_predict == 1, ]
lm_zero <- train_df[lm_predict == 0, ]
plot(NA, NA, xlim = c(-0.5, 0.5), ylim = c(-0.5, 0.5),
     main = "Problem 2(d) Plot", xlab = "X1", ylab = "X2")
points(lm_zero$x1, lm_zero$x2, col = "blue", pch = 2)
points(lm_one$x1, lm_one$x2, col = "green", pch = 3)
```



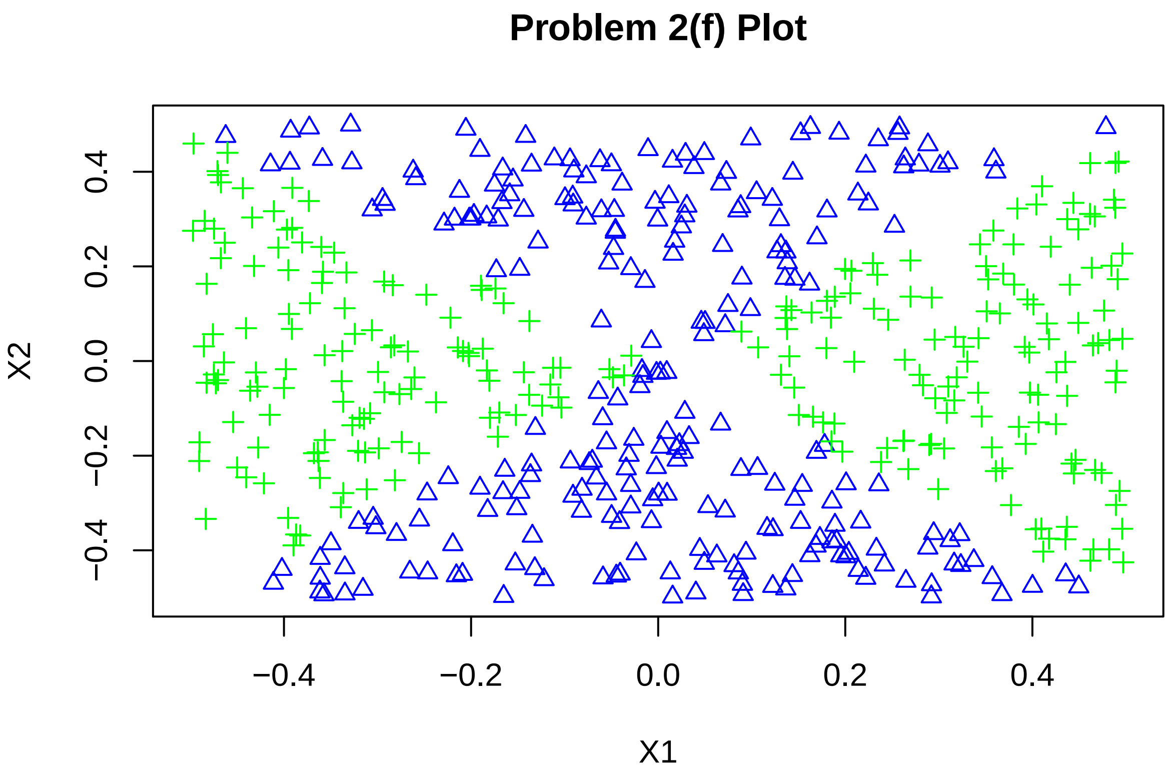
(e) Now fit a logistic regression model to the data using non-linear functions of X_1 and X_2 as predictors (e.g. X_1^2 , $X_1 * X_2$, $log(X_2)$, and so forth).

```r
lm_2e <- glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2),
             data = train_df, family = "binomial")
summary(lm_2e)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial",
##     data = train_df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -2.625e-03  -2.000e-08   0.000e+00   2.000e-08   3.016e-03
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -41.91    2659.11  -0.016    0.987
## poly(x1, 2)1    2443.19   76401.68   0.032    0.974
## poly(x1, 2)2   71512.27 1122757.43   0.064    0.949
## poly(x2, 2)1    4607.63   96068.07   0.048    0.962
## poly(x2, 2)2  -70390.26 1104395.32  -0.064    0.949
## I(x1 * x2)        43.36   31461.73   0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9315e+02  on 499  degrees of freedom
## Residual deviance: 1.9025e-05  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

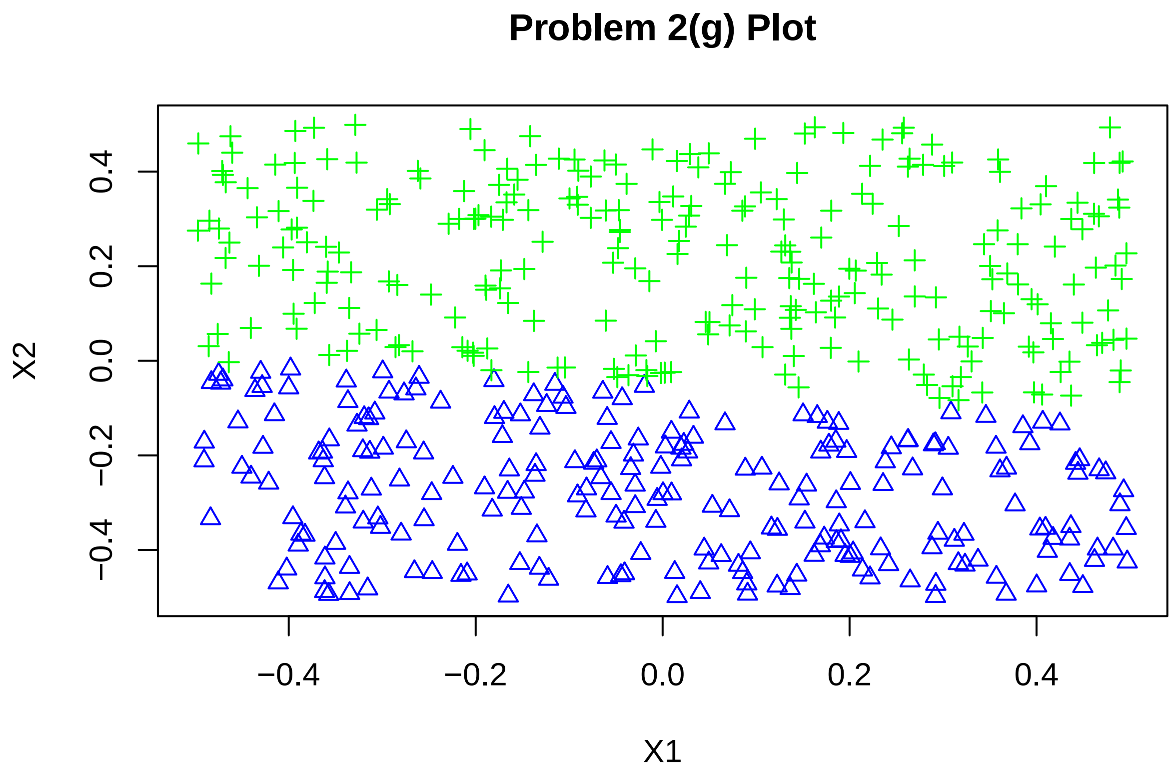
(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an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

```r
train_df <- data.frame(x1 = x1, x2 = x2, y = y)
lm_prob <- predict(lm_2e, train_df, type = "response")
lm_predict <- ifelse(lm_prob > 0.5, 1, 0)
lm_one <- train_df[lm_predict == 1, ]
lm_zero <- train_df[lm_predict == 0, ]
plot(NA, NA, xlim = c(-0.5, 0.5), ylim = c(-0.5, 0.5),
     main = "Problem 2(f) Plot", xlab = "X1", ylab = "X2")
points(lm_zero$x1, lm_zero$x2, col = "blue", pch = 2)
points(lm_one$x1, lm_one$x2, col = "green", pch = 3)
```



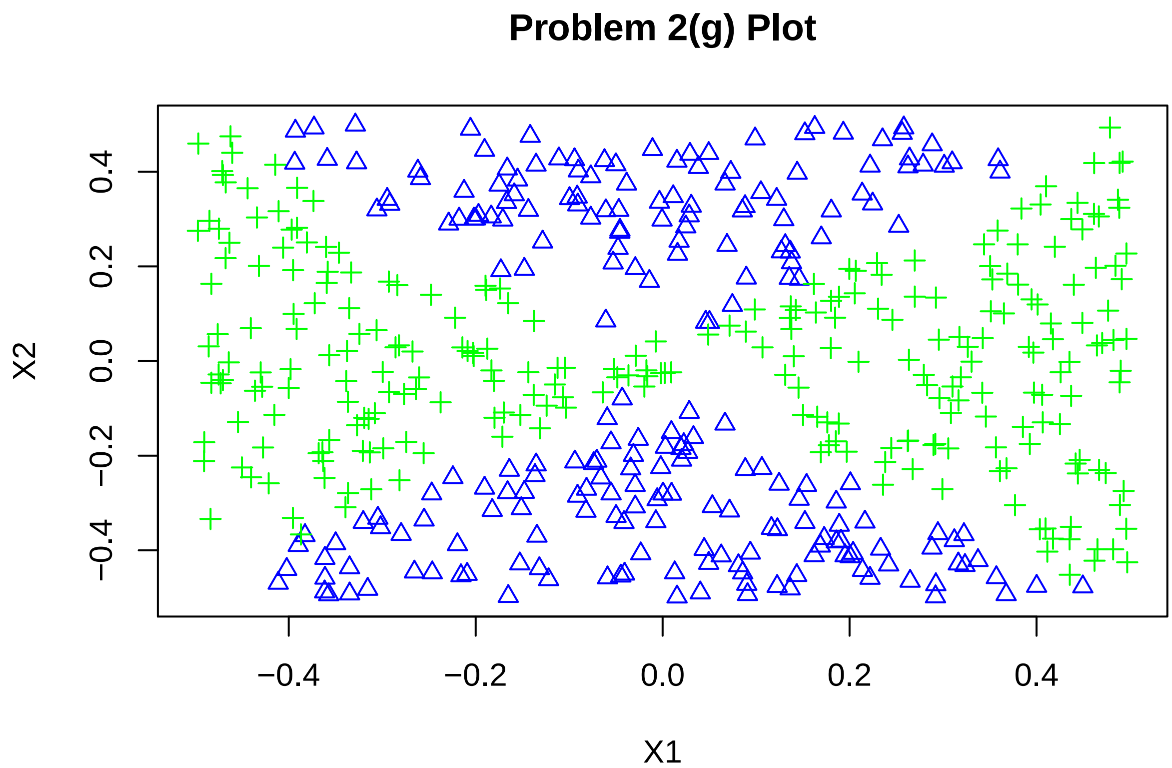
(g) Fit a support vector classifier to the data with X_1 and X_2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
svm_2g <- svm(as.factor(y) ~ x1 + x2, train_df,
              kernel = "linear", scale = FALSE,
              cost = 10, type = "C")
svm_pred <- predict(svm_2g, train_df)
svm_one <- train_df[svm_pred == 1, ]
svm_zero <- train_df[svm_pred == 0, ]
plot(NA, NA, xlim = c(-0.5, 0.5), ylim = c(-0.5, 0.5),
     main = "Problem 2(g) Plot", xlab = "X1", ylab = "X2")
points(svm_zero$x1, svm_zero$x2, col = "blue", pch = 2)
points(svm_one$x1, svm_one$x2, col = "green", pch = 3)
```



(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.

```
svm_2h <- svm(as.factor(y) ~ x1 + x2, train_df)
svm_pred <- predict(svm_2h, train_df)
svm_one <- train_df[svm_pred == 1, ]
svm_zero <- train_df[svm_pred == 0, ]
plot(NA, NA, xlim = c(-0.5, 0.5), ylim = c(-0.5, 0.5),
     main = "Problem 2(g) Plot", xlab = "X1", ylab = "X2")
points(svm_zero$x1, svm_zero$x2, col = "blue", pch = 2)
points(svm_one$x1, svm_one$x2, col = "green", pch = 3)
```



(i) Comment on your results.

COMMENTS: We can see that non-linear kernel SVM's are good at locating non-linear boundaries compared to linear kernel SVM's. The results from our logistic regression method is similar to the results of a non-linear kernel SVM's, but we have to create quadratic terms. Creating the quadratic terms could prove to be difficult and non-efficient. Thus, we are better off using non-linear kernel SVM's for this case.

Problem 3 (Exercise 9.7.8)

This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
data(OJ)
set.seed(1)
train <- sample(nrow(OJ), 800)
oj_train <- OJ[train, ]
oj_test <- OJ[-train, ]
```

(b) Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```r
oj_svm <- svm(Purchase ~ ., data = oj_train,
              kernel = "linear", scale = FALSE, cost = 10)
summary(oj_svm)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  324
##
##  ( 163 161 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

(c) What are the training and test error rates?

```r
# Create the tables and predictions
oj_train_pred <- predict(oj_svm, oj_train)
oj_train_table <- table(oj_train$Purchase, oj_train_pred)
oj_test_pred <- predict(oj_svm, oj_test)
oj_test_table <- table(oj_test$Purchase, oj_test_pred)

# Show tables
oj_train_table
```

```
##     oj_train_pred
##       CH  MM
##   CH 420  65
##   MM  71 244
```

```r
oj_test_table
```

```
##     oj_test_pred
##       CH  MM
##   CH 154  14
##   MM  29  73
```

```r
# Calculate the error rates for training and test
oj_train_error <- (65+71) / (420+65+71+244)
oj_test_error <- (14+29) / (154+14+29+73)
```

```r
# Compare the error rates
compare_matrix <- matrix(c(oj_train_error, oj_test_error))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1700000
## Test       0.1592593
```

```r
# 17% error rate for training
# 16% error rate for test
```

(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```r
optimal_cost <- tune(svm, Purchase ~ ., data = oj_train,
                     kernel = "linear", scale = FALSE,
                     ranges = list(cost = 10^seq(-2, 1, by = 0.5)))
# Optimal Cost
optimal_cost$best.parameters$cost
```

```
## [1] 0.3162278
```

ANSWER: We find the best parameter for optimal cost is 0.3162278.

(e) Compute the training and test error rates using this new value for cost.

```r
# Create the tables and predictions
oj_svm_best <- svm(Purchase ~ ., kernel = "linear", data = oj_train,
                       cost = optimal_cost$best.parameters$cost)
oj_train_pred_best <- predict(oj_svm_best, oj_train)
oj_train_table_best <- table(oj_train$Purchase, oj_train_pred_best)
oj_test_pred_best <- predict(oj_svm_best, oj_test)
oj_test_table_best <- table(oj_test$Purchase, oj_test_pred_best)

# Show tables
oj_train_table_best
```

```
##     oj_train_pred_best
##       CH  MM
##   CH 423  62
##   MM  71 244
```

```r
oj_test_table_best
```

```
##     oj_test_pred_best
##       CH  MM
##   CH 155  13
##   MM  29  73
```

```r
# Calculate the error rates for training and test
oj_train_error_best <- (62+71) / (423+62+71+244)
oj_test_error_best <- (13+29) / (155+13+29+73)

# Compare the error rates
compare_matrix <- matrix(c(oj_train_error_best, oj_test_error_best))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1662500
## Test       0.1555556
```

```r
# 16.625% error rate for training
# 15.556% error rate for test
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```r
### part (b)
set.seed(1)
oj_svm_radial <- svm(Purchase ~ ., data = oj_train, kernel = "radial")
summary(oj_svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
### part (c)
# Create the tables and predictions
oj_train_pred_radial <- predict(oj_svm_radial, oj_train)
oj_train_table_radial <- table(oj_train$Purchase, oj_train_pred_radial)
oj_test_pred_radial <- predict(oj_svm_radial, oj_test)
oj_test_table_radial <- table(oj_test$Purchase, oj_test_pred_radial)
```

```r
# Show tables
oj_train_table_radial
```

```
##     oj_train_pred_radial
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```r
oj_test_table_radial
```

```
##     oj_test_pred_radial
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```r
# Calculate the error rates for training and test
oj_train_error_radial <- (44+77) / (441+44+77+238)
oj_test_error_radial <- (33+17) / (151+17+33+69)

# Compare the error rates
compare_matrix <- matrix(c(oj_train_error_radial, oj_test_error_radial))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1512500
## Test       0.1851852
```

```r
# 15.125% error rate for training
# 18.519% error rate for test


### part (d)
optimal_cost_radial <- tune(svm, Purchase ~ ., data = oj_train,
                     kernel = "radial", scale = FALSE,
                     ranges = list(cost = 10^seq(-2, 1, by = 0.5)))
# Optimal Cost
optimal_cost_radial$best.parameters$cost
```

```
## [1] 10
```

```r
### part (e)
# Create the tables and predictions
oj_svm_best_radial <- svm(Purchase ~ ., kernel = "radial", data = oj_train,
                      cost = optimal_cost_radial$best.parameters$cost)
oj_train_pred_best_radial <- predict(oj_svm_best_radial, oj_train)
oj_train_table_best_radial <- table(oj_train$Purchase,
                                    oj_train_pred_best_radial)
oj_test_pred_best_radial <- predict(oj_svm_best_radial, oj_test)
```

```r
oj_test_table_best_radial <- table(oj_test$Purchase, oj_test_pred_best_radial)

# Show tables
oj_train_table_best_radial
```

```
##     oj_train_pred_best_radial
##       CH  MM
##   CH 442  43
##   MM  73 242
```

```r
oj_test_table_best_radial
```

```
##     oj_test_pred_best_radial
##       CH  MM
##   CH 154  14
##   MM  36  66
```

```r
# Calculate the error rates for training and test
oj_train_error_best_radial <- (43+73) / (442+43+73+242)
oj_test_error_best_radial <- (14+36) / (154+14+36+66)

# Compare the error rates
compare_matrix <- matrix(c(oj_train_error_best_radial,
                           oj_test_error_best_radial))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1450000
## Test       0.1851852
```

```r
# 14.5% error rate for training
# 18.519% error rate for test
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```r
### part (b)
set.seed(1)
oj_svm_poly <- svm(Purchase ~ ., data = oj_train,
                   kernel = "polynomial", degree = 2)
summary(oj_svm_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  447
##
##  ( 225 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
### part (c)
# Create the tables and predictions
oj_train_pred_poly <- predict(oj_svm_poly, oj_train)
oj_train_table_poly <- table(oj_train$Purchase, oj_train_pred_poly)
oj_test_pred_poly <- predict(oj_svm_poly, oj_test)
oj_test_table_poly <- table(oj_test$Purchase, oj_test_pred_poly)

# Show tables
oj_train_table_poly
```

```
##     oj_train_pred_poly
##       CH  MM
##   CH 449  36
##   MM 110 205
```

```r
oj_test_table_poly
```

```
##     oj_test_pred_poly
##       CH  MM
##   CH 153  15
##   MM  45  57
```

```r
# Calculate the error rates for training and test
oj_train_error_poly <- (110+36) / (110+36+205+449)
oj_test_error_poly <- (45+15) / (45+15+153+57)

# Compare the error rates
compare_matrix <- matrix(c(oj_train_error_poly, oj_test_error_poly))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1825000
## Test       0.2222222
```

```r
# 18.25% error rate for training
# 22.22% error rate for test


### part (d)
#optimal_cost_poly <- tune(svm, Purchase ~ ., data = oj_train,
#                      kernel = "polynomial", degree = 2, scale = FALSE,
#                      ranges = list(cost = 10^seq(-2, 1, by = 0.5)))
# Optimal Cost
#optimal_cost_poly$best.parameters$cost
# Difficult to load but it came out as cost = 10 (best)

### part (e)
# Create the tables and predictions
oj_svm_best_poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2,
                        data = oj_train,
                        cost = 10)
oj_train_pred_best_poly <- predict(oj_svm_best_poly, oj_train)
oj_train_table_best_poly <- table(oj_train$Purchase,
                                  oj_train_pred_best_poly)
oj_test_pred_best_poly <- predict(oj_svm_best_poly, oj_test)
oj_test_table_best_poly <- table(oj_test$Purchase, oj_test_pred_best_poly)

# Show tables
oj_train_table_best_poly
```

```
##     oj_train_pred_best_poly
##       CH  MM
##   CH 447  38
##   MM  82 233
```

```r
oj_test_table_best_poly
```

```
##     oj_test_pred_best_poly
##       CH  MM
##   CH 154  14
##   MM  37  65
```

```r
# Calculate the error rates for training and test
oj_train_error_best_poly <- (38+82) / (447+38+82+233)
oj_test_error_best_poly <- (14+37) / (154+14+37+65)

# Compare the error rates
compare_matrix <- matrix(c(oj_train_error_best_poly,
                           oj_test_error_best_poly))
colnames(compare_matrix) <- "Error Rates"
rownames(compare_matrix) <- c("Training", "Test")
compare_matrix
```

```
##          Error Rates
## Training   0.1500000
## Test       0.1888889
```

```
# 15% error rate for training
# 18.889% error rate for test
```

(h) Overall, which approach seems to give the best results on this data?

ANSWER: We can see that radial kernel with the optimal cost of 10 yielded the lowest error rates with $training = 14.5\%$ and $test = 18.519\%$. For this data, we should use the Radial Kernel with its optimal cost.