Stats 202 Homework 9

Carina Zhang

12/7/2018

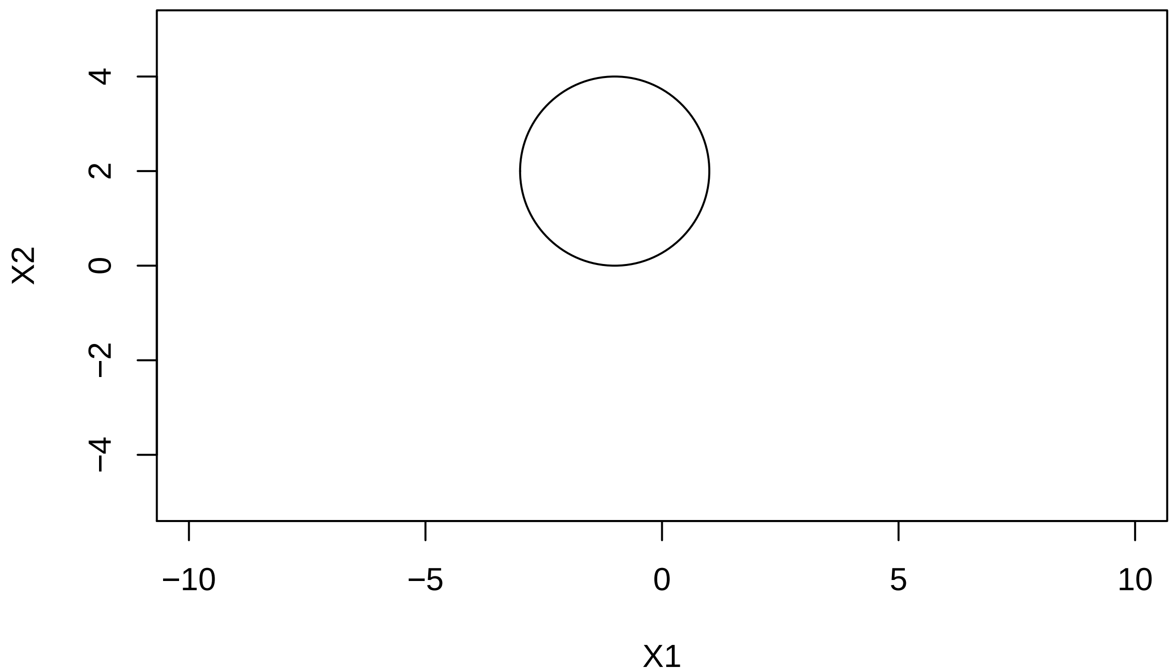
Citation: Code from labs in textbook.

Problem 1

Chapter 9, Exercise 2 (p. 368).

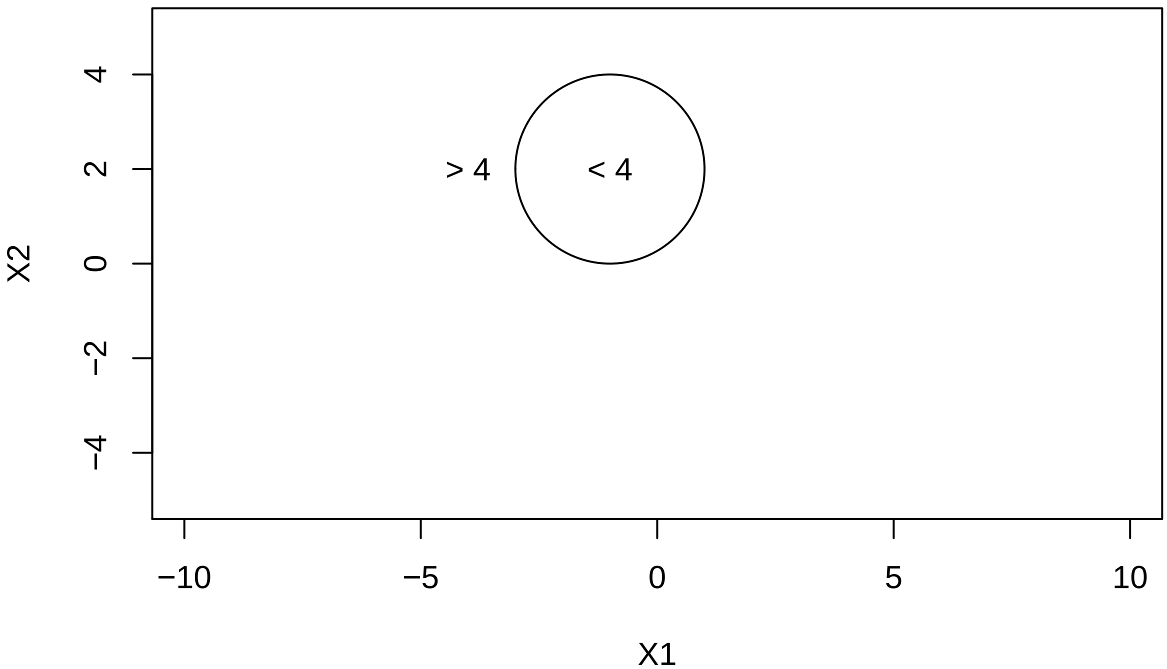
1 (a)

```
r = 2
plot(NA, NA, type = "n", xlim = c(-5, 5), ylim = c(-5, 5), asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(r), add = TRUE,inches = FALSE)
```



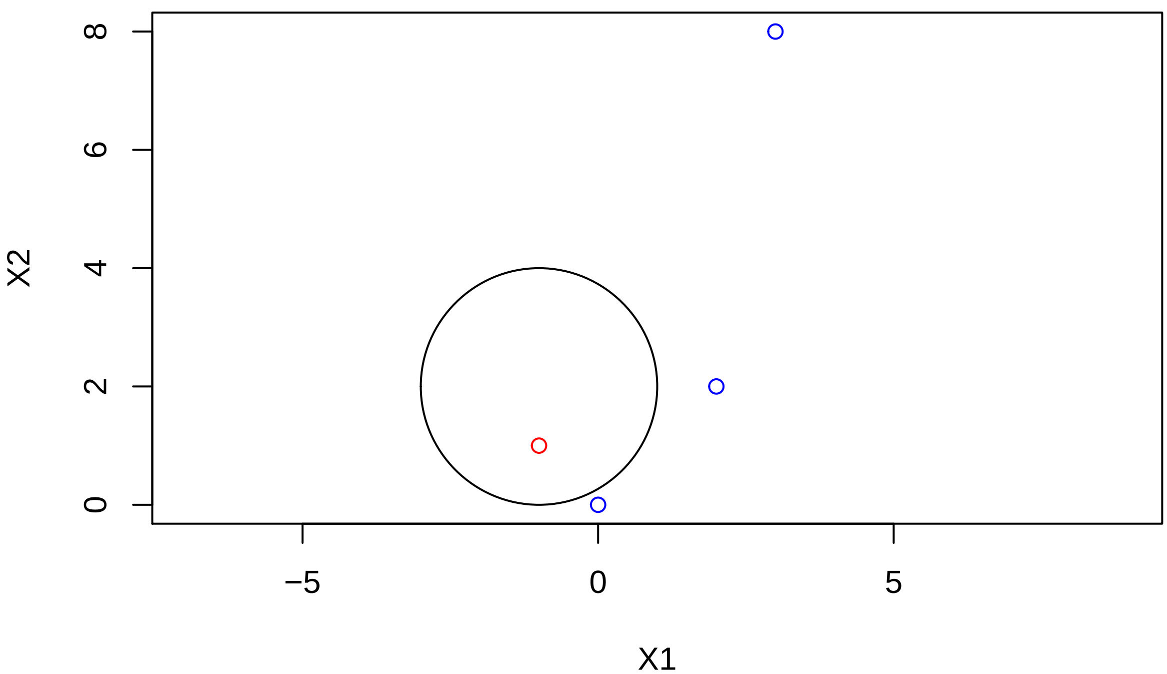
1 (b)

```
r = 2
plot(NA, NA, type = "n", xlim = c(-5, 5), ylim = c(-5, 5), asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(r), add = TRUE, inches = FALSE)
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```



1 (c)

```r
r = 2
plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"), type = "p", asp = 1, xlab =
symbols(c(-1), c(2), circles = c(r), add = TRUE, inches = FALSE)
```



(0, 0), (2, 2), (3, 8) are classified to the blue class. (-1, 1) is classified to the red class.

1 (d)

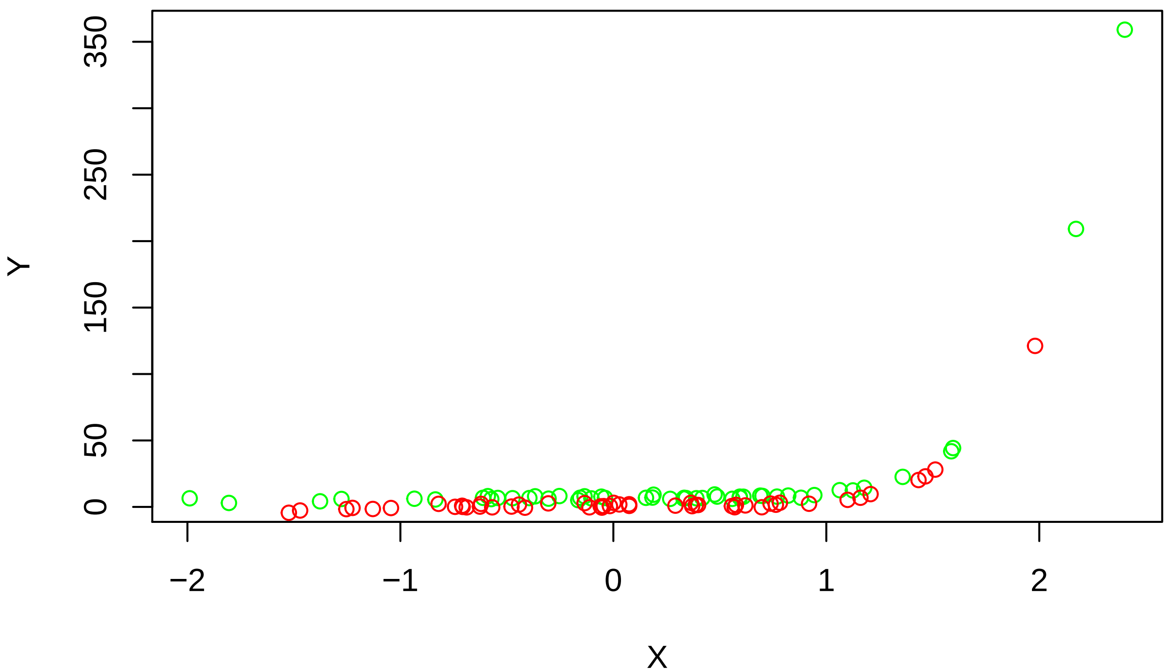
We can expand the equation of the boundary, Since $(1+X_1)^2+(2-X_2)^2 = 4$, we get $X_1^2+X_2^2+2*X_1-4*X_2+1 = 0$. Thus, it is linear in terms of X_1, X_1^2, X_2, X_2^2.

Problem 2

Chapter 9, Exercise 4 (p. 369).

```
set.seed(1)
x <- rnorm(100)
y <- 2*x^5 + x^6 + 4 + rnorm(100)
train <- sample(100, 50)
y[train] <- y[train] + 3
y[-train] <- y[-train] - 3

# Plot
plot(x[train], y[train], col = "green", xlab = "X", ylab = "Y")
points(x[-train], y[-train], col = "red")
```

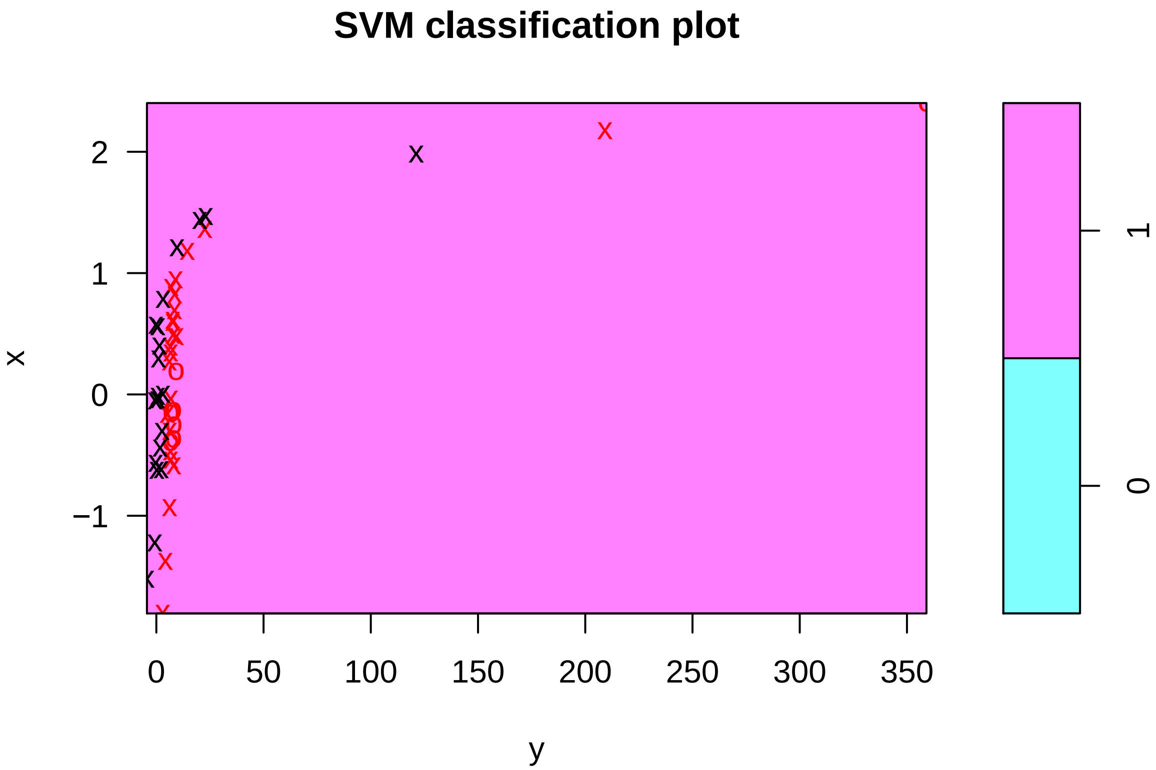


SVM - Training

```
library(e1071)
set.seed(2)
z <- rep(0, 100)
z[train] <- 1

data <- data.frame(x = x, y = y, z = as.factor(z))
train <- sample(100, 50)
data.train <- data[train, ]
data.test <- data[-train, ]
```

```r
svm.linear <- svm(z ~ ., data = data.train, kernel = "linear", cost = 10)
plot(svm.linear, data.train)
```




```r
table = table(predict(svm.linear, data.train), data.train$z) #confusion matrix
table
```

```
##
##      0  1
##   0  0  0
##   1 20 30
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

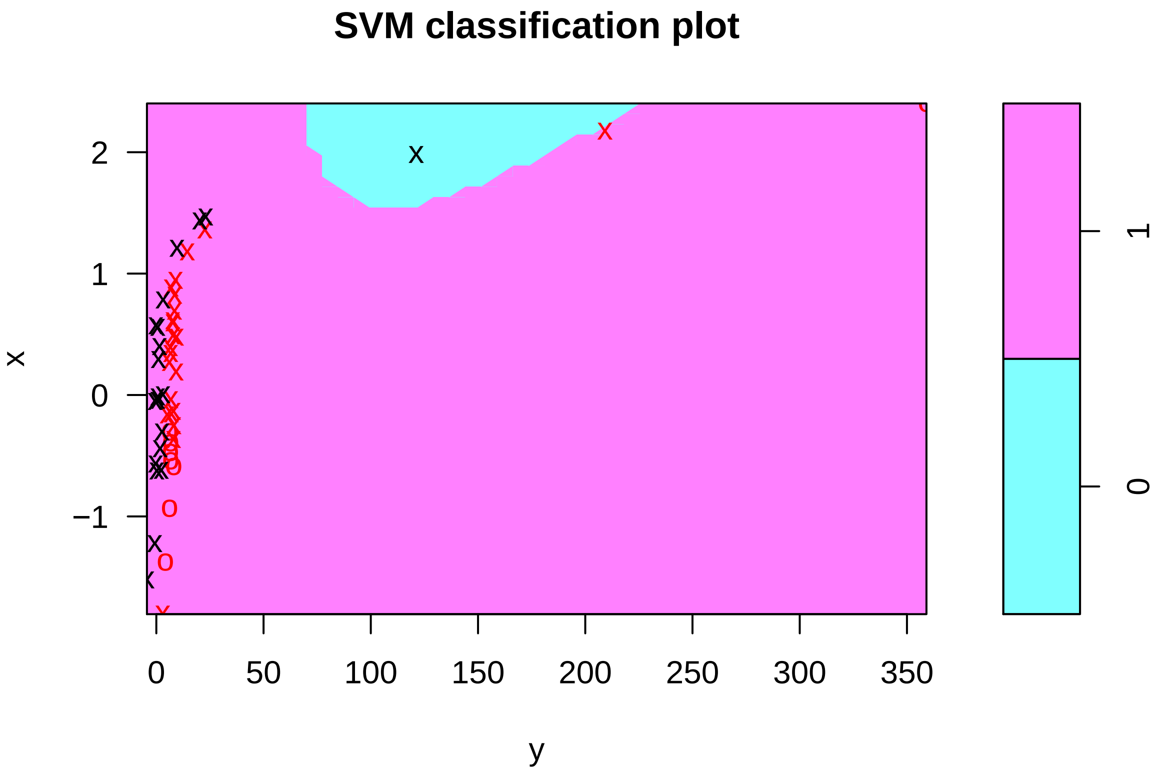
```
## [1] 0.4
```

```r
cat("The Training Error Rate is: ", error)
```

```
## The Training Error Rate is:  0.4
```

SVM - Poly - Training

```r
set.seed(4)
svm.poly <- svm(z ~ ., data = data.train, kernel = "polynomial", cost = 10)
plot(svm.poly, data.train)
```




```
table = table(predict = predict(svm.poly, data.train), truth = data.train$z)
table
```

```
##          truth
## predict   0  1
##        0  1  0
##        1 19 30
```

```
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

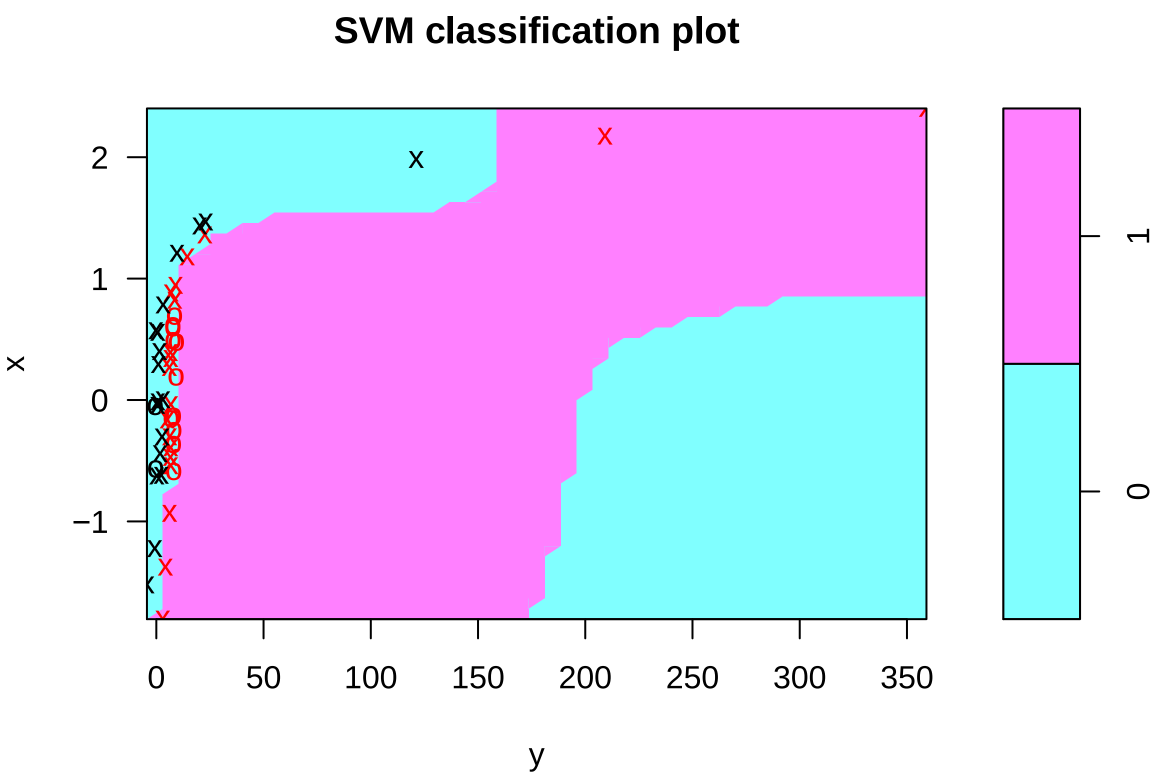
```
## [1] 0.38
```

```
cat("The Training Error Rate is: ", error)
```

```
## The Training Error Rate is:  0.38
```

Third, we fit a support vector machine with a radial kernel.

```
svm.radial <- svm(z ~ ., data = data.train, kernel = "radial", cost = 20)
plot(svm.radial, data.train)
```




```
table = table(predict = predict(svm.radial, data.train), truth = data.train$z)
table
```

```
##         truth
## predict  0  1
##        0 20  1
##        1  0 29
```

```
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.02
```

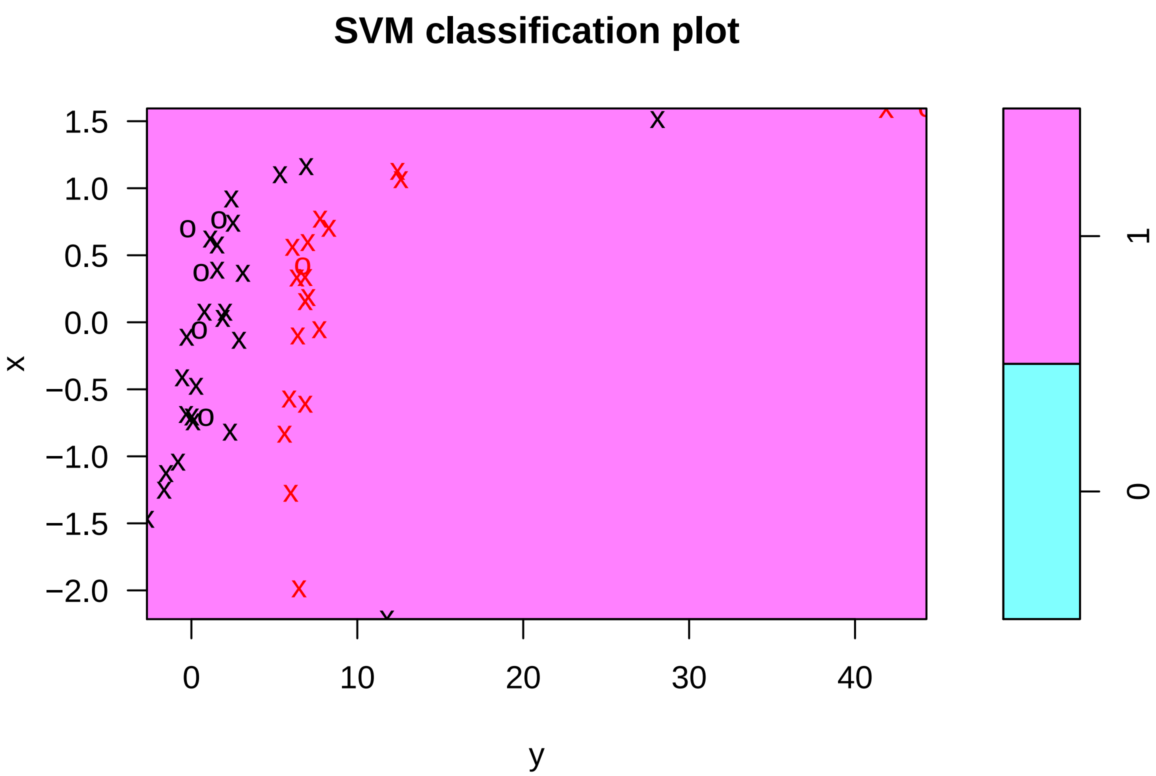
```
cat("The Training Error Rate is: ", error)
```

```
## The Training Error Rate is:  0.02
```

We see that from all the training data error, both the polynomial-kernal (0.38) and Radial kernel(0.02) outperforms the SVM classifier (0.4).

SVM Linear - Test

```
plot(svm.linear, data.test)
```




```
table = table(predict = predict(svm.linear, data.test), truth = data.test$z)
table
```

```
##         truth
## predict  0  1
##       0  0  0
##       1 30 20
```

```
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

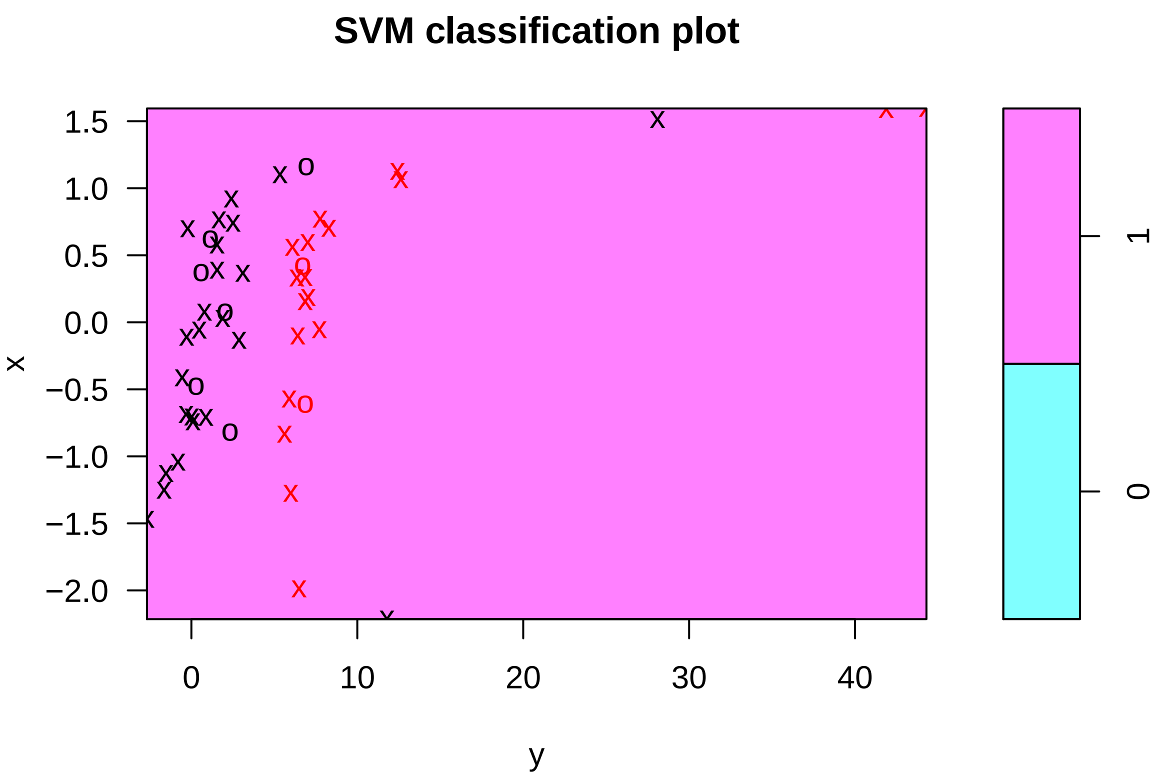
```
## [1] 0.6
```

```
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.6
```

SVM - poly - test

```
plot(svm.poly, data.test)
```




```r
table = table(predict = predict(svm.poly, data.test), truth = data.test$z)
table
```

```
##         truth
## predict  0  1
##       0  0  0
##       1 30 20
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

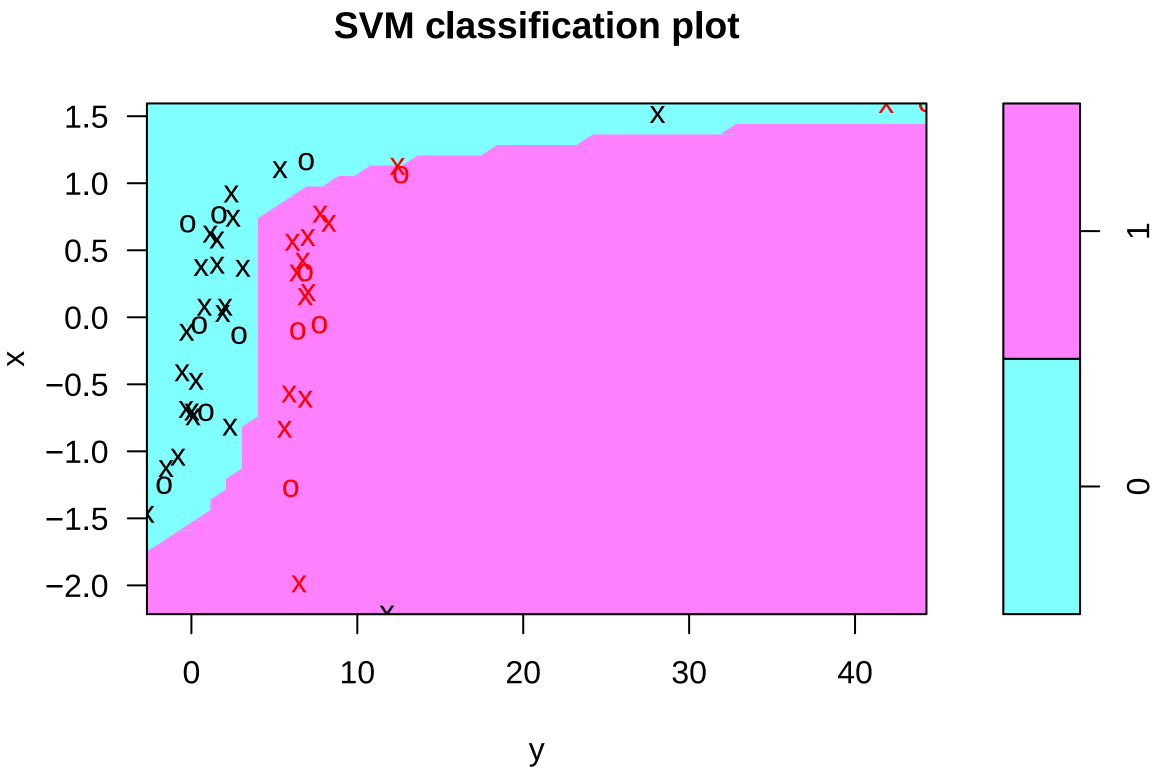
```
## [1] 0.6
```

```r
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.6
```

SVM - radial -

```r
plot(svm.radial, data.test)
```




```
table  = table(predict = predict(svm.radial, data.test), truth = data.test$z)
table
```

```
##        truth
## predict  0  1
##       0 29  2
##       1  1 18
```

```
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.06
```

```
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.06
```

We see that the linear, polynomial and radial support vector machines has a 0.6, 0.6, and 0.06 test error, respectively. So, radial kernel is the best model in this setting, with a multiple non-linear terms in the new boundary.

Problem 3

Chapter 9, Exercise 5 (p. 369).

3 (a)

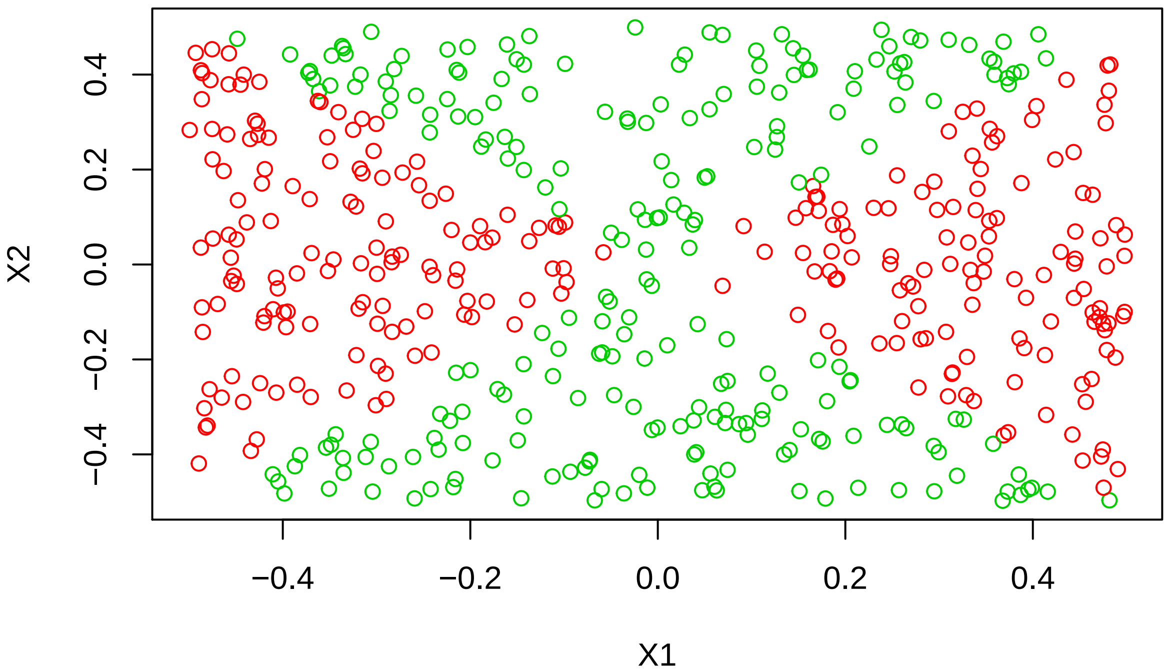
Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them.

```r
set.seed(2)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
```

3 (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.

```r
plot(x1, x2, xlab = "X1", ylab = "X2", col = (3 - y))
```



3 (c)

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

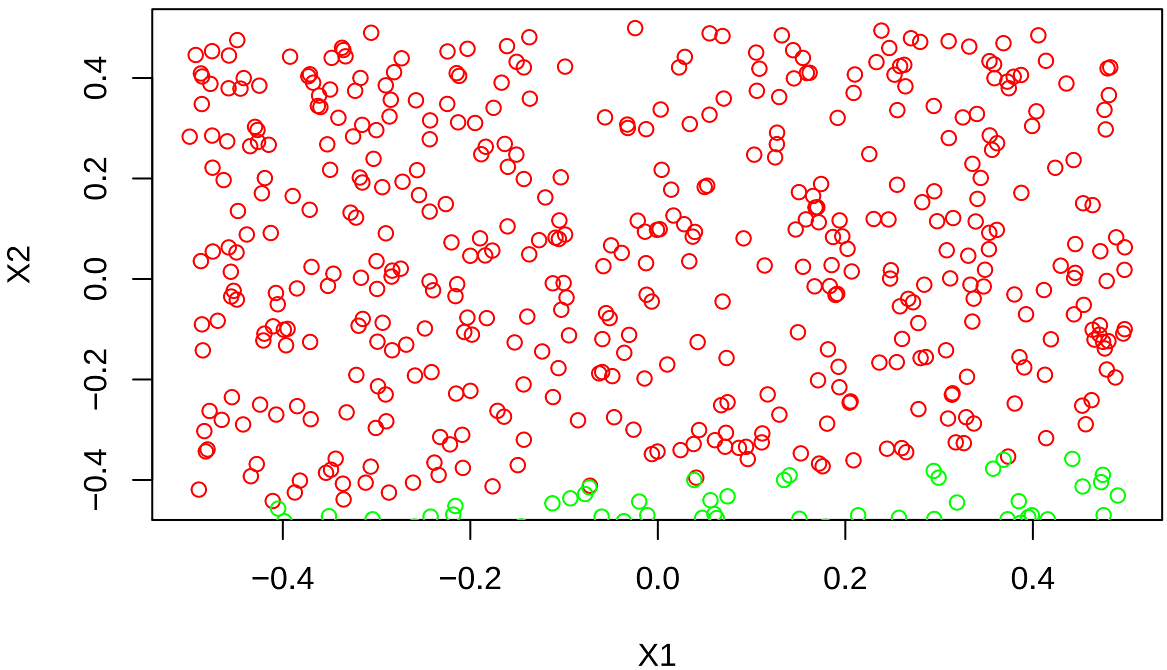
```r
fit <- glm(y ~ x1 + x2, family = "binomial")
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.271  -1.193   1.097   1.147   1.209
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.07138    0.08959   0.797    0.426
## x1          -0.03532    0.29825  -0.118    0.906
```

```
## x2           0.27548    0.30762   0.896    0.370
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.50  on 499  degrees of freedom
## Residual deviance: 691.67  on 497  degrees of freedom
## AIC: 697.67
##
## Number of Fisher Scoring iterations: 3
```

3 (d)

```r
data <- data.frame(x1 = x1, x2 = x2, y = y)
probs = predict(fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.49] = 1
plot(data[preds == 1,]$x1, data[preds == 1,]$x2, col = "red", xlab = "X1", ylab = "X2")
points(data[preds == 0,]$x1, data[preds == 0,]$x2, col = "green")
```



The decision boundary between the green dots and the red dots is linear (the very flat line at the bottom of the graph)

3 (e)

Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors.

```r
fit_nl <- glm(y ~ poly(x1, 3) + poly(x2, 2) + I(x1 * x2), family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

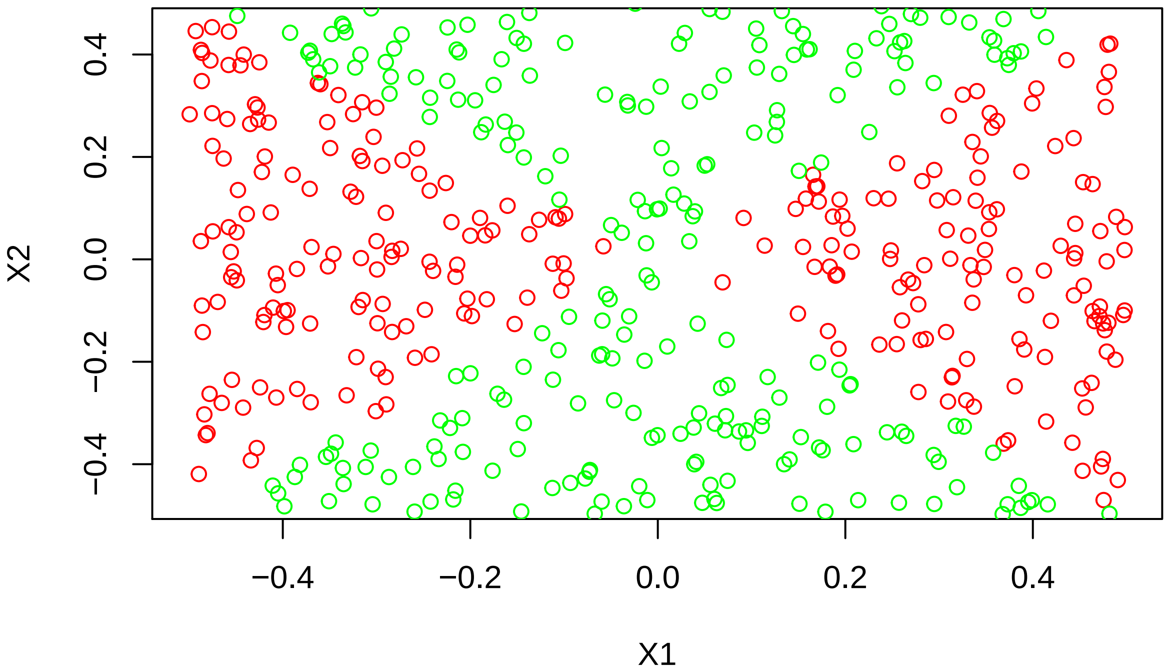
```r
summary(fit_nl)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 3) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.275e-03  -2.000e-08   2.000e-08   2.000e-08   9.567e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)      83.52    3032.70   0.028    0.978
## poly(x1, 3)1   -262.79   38079.44  -0.007    0.994
## poly(x1, 3)2  26835.91  837602.91   0.032    0.974
## poly(x1, 3)3   -255.24   26750.33  -0.010    0.992
## poly(x2, 2)1    594.00   38344.53   0.015    0.988
## poly(x2, 2)2 -27011.68  842660.05  -0.032    0.974
## I(x1 * x2)       87.45   13359.71   0.007    0.995
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9250e+02  on 499  degrees of freedom
## Residual deviance: 3.5668e-06  on 493  degrees of freedom
## AIC: 14
##
## Number of Fisher Scoring iterations: 25
```

3 (f)

As bellow, the decision boundary is non-linear. The non-linear decision boundary is surprisingly very similar to the true decision boundary.

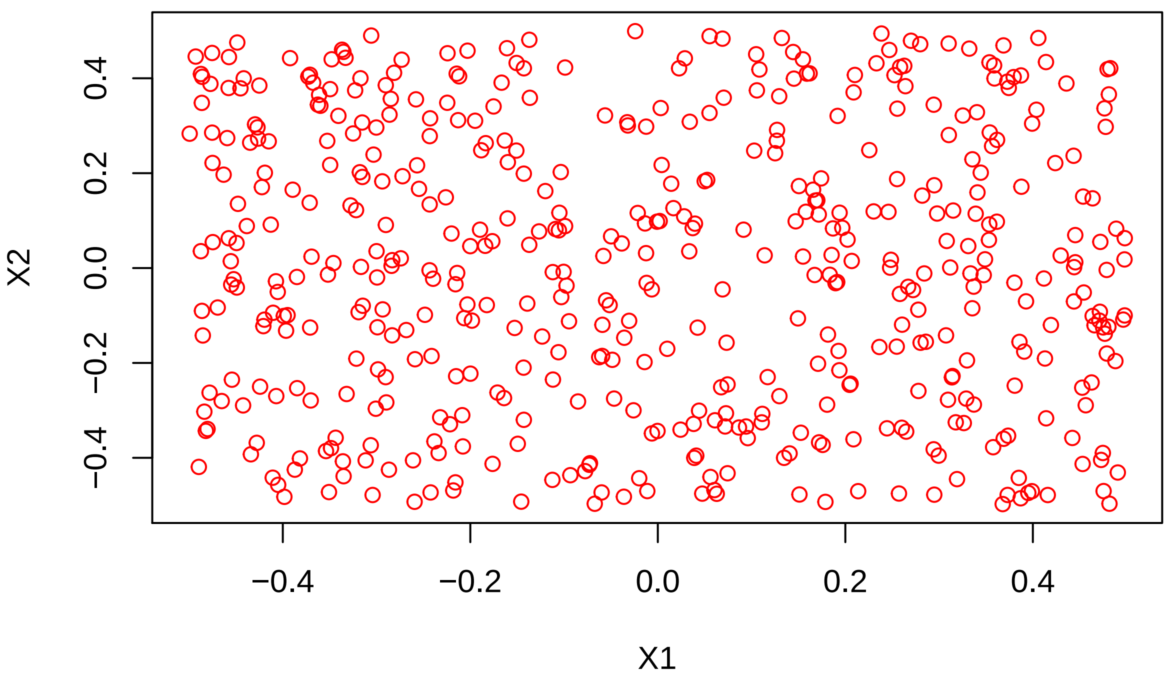
```r
probs <- predict(fit_nl, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.49] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = "red", xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = "green")
```



3 (g)

As we can see from the graph, a linear kernel with low cost classifies all points to a single class.

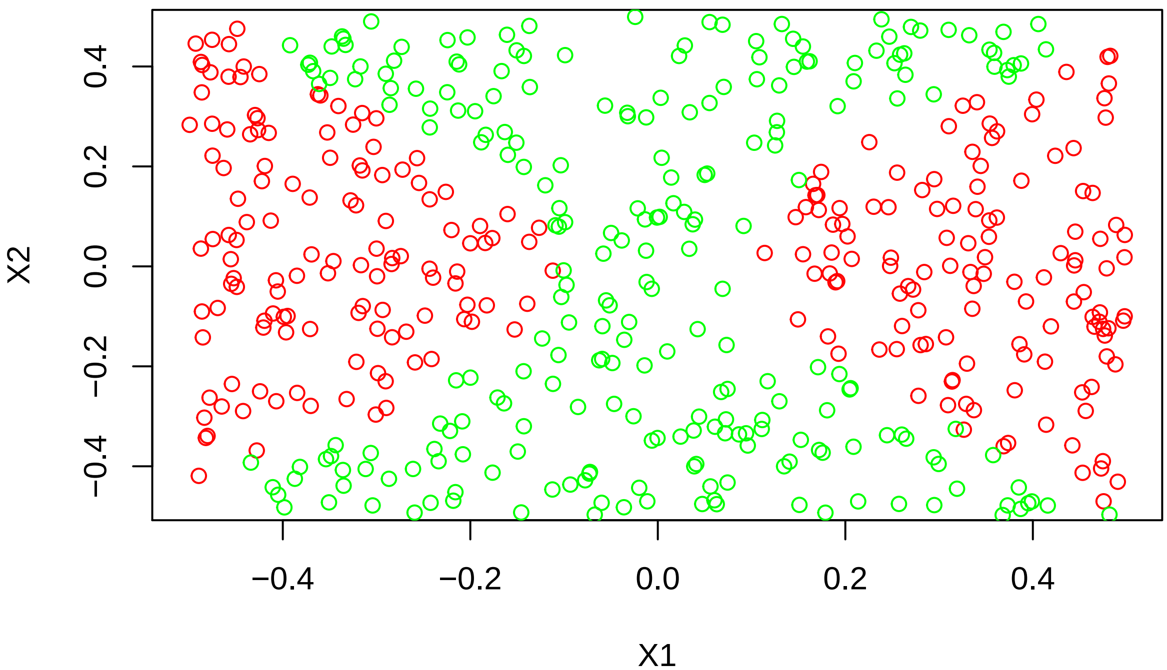
```
data$y <- as.factor(data$y)
svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.01)
preds= predict(svm.fit, data)
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = "red", xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = "green")
```



3 (h)

Similar to the previous parts, the radial kernel estimates the decision boundary well and resembles the true decision boundary.

```
data$y <- as.factor(data$y)
svmnl.fit <- svm(y ~ x1 + x2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = "red", xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = "green")
```



3 (i) Comment

We can see that SVMs with non-linear kernel and logistic regression with interaction terms are equally very powerful in finding non-linear boundaries. Also, SVM with linear kernel and logistic regression without any interaction term does not do well in finding non-linear decision boundaries. However, there is some manual efforts involved in tuning or picking right interaction term when it is a non-linear model. When we have a large number of features, the feature selection might become arbitrary. However, we can also implement cross-validation technique to make sure that we choose the best parameter.

Problem 4

Chapter 9, Exercise 8 (p. 371).

4 (a) & (b)

Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. 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
library(ISLR)
set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]

svm.linear <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

As described in the summary data, the SVM model uses 432 support vectors. 215 of the vectors are in the CH class and 217 of the vectors are in the MM class.

4 (c)

What are the training and test error rates?

```r
train.pred <- predict(svm.linear, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 439  55
##   MM  78 228
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2])
error
```

```
## [1] 0.1994003
```

```r
cat("The Train Error Rate is: ", error, "\n")
```

```
## The Train Error Rate is:  0.1994003
```

```r
test.pred <- predict(svm.linear, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  31  80
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2])
error
```

```
## [1] 0.2217195
```

```r
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.2217195
```

4 (d)

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

```r
set.seed(1)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        cost
##  0.03162278
##
## - best performance: 0.17375
##
## - Detailed performance results:
##          cost   error dispersion
## 1  0.01000000 0.17500 0.03996526
## 2  0.03162278 0.17375 0.03884174
## 3  0.10000000 0.17875 0.03821086
## 4  0.31622777 0.17625 0.03701070
## 5  1.00000000 0.17750 0.03717451
## 6  3.16227766 0.18000 0.03496029
## 7 10.00000000 0.18000 0.04005205
```

The best forming cost is cost = 0.03162278

4 (e)

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

```r
set.seed(1)

svm.linear <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost =  0.03162278)
```

```r
train.pred <- predict(svm.linear, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 438  56
##   MM  72 234
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2])
error
```

```
## [1] 0.1904762
```

```r
cat("The Train Error Rate is: ", error, "\n")
```

```
## The Train Error Rate is:  0.1904762
```

```r
test.pred <- predict(svm.linear, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 138  21
##   MM  31  80
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2])
error
```

```
## [1] 0.2385321
```

```r
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.2385321
```

4 (f)

```r
set.seed(1)

svm.rad <- svm(Purchase ~ ., data = OJ.train, kernel = "radial")

train.pred <- predict(svm.rad, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 455  39
##   MM  77 229
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.145
```

```r
cat("The Train Error Rate is: ", error, "\n")
```

```
## The Train Error Rate is:  0.145
```

```r
test.pred <- predict(svm.rad, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  28  83
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.1703704
```

```r
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.1703704
```

```r
#tune the model
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.3162278
##
## - best performance: 0.17125
##
## - Detailed performance results:
##          cost   error dispersion
## 1  0.01000000 0.38250 0.05596378
## 2  0.03162278 0.37250 0.06341004
## 3  0.10000000 0.17875 0.04168749
## 4  0.31622777 0.17125 0.05001736
## 5  1.00000000 0.17500 0.04750731
## 6  3.16227766 0.18000 0.04830459
## 7 10.00000000 0.18250 0.04866267
```

```r
svm.rad <- svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost =  0.3162278)

train.pred <- predict(svm.rad, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 448  46
##   MM  78 228
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.155
```

```r
cat("The Train Error Rate is: ", error, "\n")
```

```
## The Train Error Rate is:  0.155
```

```r
test.pred <- predict(svm.rad, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 144  15
##   MM  29  82
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.162963
```

```r
cat("The New Test Error Rate with best Cost is: ", error)
```

```
## The New Test Error Rate with best Cost is:  0.162963
```

4 (g)

```r
set.seed(1)

svm.poly <- svm(Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2)

train.pred <- predict(svm.poly, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 461  33
##   MM 105 201
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.1725
```

```r
cat("The Train Error Rate is: ", error, "\n")
```

```
## The Train Error Rate is:  0.1725
```

```r
test.pred <- predict(svm.poly, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 149  10
##   MM  41  70
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.1888889
```

```r
cat("The Test Error Rate is: ", error)
```

```
## The Test Error Rate is:  0.1888889
```

```r
#tune the model
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", ranges = list(cost = 10^seq
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.162278
##
## - best performance: 0.175
##
## - Detailed performance results:
##          cost   error dispersion
## 1  0.01000000 0.36750 0.05041494
## 2  0.03162278 0.34625 0.05894029
## 3  0.10000000 0.30250 0.07402139
## 4  0.31622777 0.20250 0.06368324
## 5  1.00000000 0.18750 0.05368374
## 6  3.16227766 0.17500 0.05559027
## 7 10.00000000 0.18125 0.04686342
```

The best performing is cost = 3.162278

```r
svm.poly <- svm(Purchase ~ ., data = OJ.train, kernel = "polynomial", cost =  3.162278)

train.pred <- predict(svm.poly, OJ.train)
table = table(OJ.train$Purchase, train.pred)
table
```

```
##     train.pred
##       CH  MM
##   CH 460  34
##   MM  76 230
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.1375
```

```r
cat("The New Train Error Rate with best cost is: ", error, "\n")
```

```
## The New Train Error Rate with best cost is:  0.1375
```

```r
test.pred <- predict(svm.poly, OJ.test)
table = table(OJ.test$Purchase, test.pred)
table
```

```
##     test.pred
##       CH  MM
##   CH 142  17
##   MM  37  74
```

```r
error = (table[1,2] + table[2, 1]) / (table[1,1] + table[2, 2] + table[1,2] + table[2, 1])
error
```

```
## [1] 0.2
```

```r
cat("The New Test Error Rate with best Cost is: ", error)
```

```
## The New Test Error Rate with best Cost is:  0.2
```

4 (h)

In terms of training error, the tuned radial kernel SVM has the smallest training error.

In terms of testing error, the tuned polynomial kernal SVM has the smallest testing error

Overall, the non-linear SVM kernel seem to give the best results on this data with smaller training/test error.