ISLR | Chapter 9 Exercises

Marshall McQuillen

11/23/2018

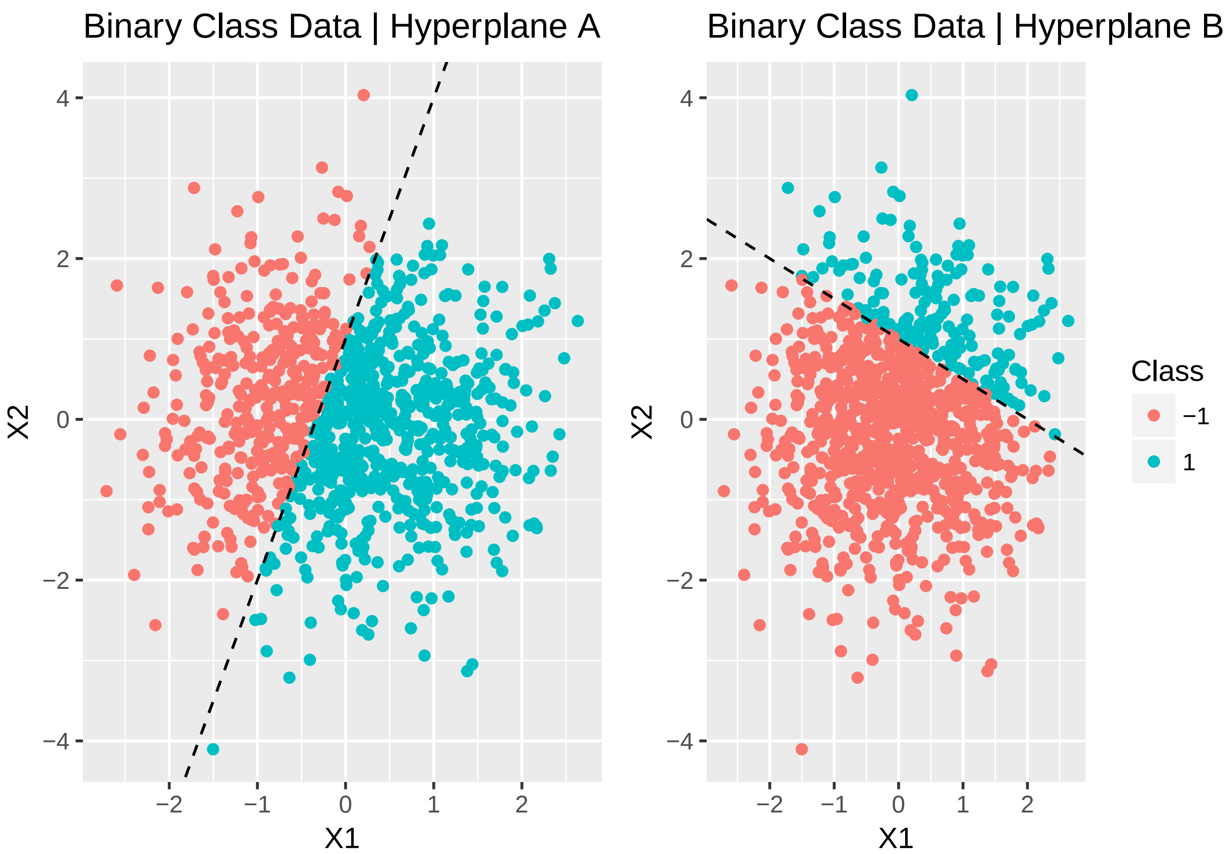
Conceptual

1

- **A & B**.

```r
suppressPackageStartupMessages(library(gridExtra))
suppressPackageStartupMessages(library(ggplot2))

# data setup
x <- matrix(rnorm(1000*2), ncol = 2)

# define hyperplane 1 (part A)
hyperplane1 <- 1 + 3*x[, 1] - x[, 2]
y <- ifelse(hyperplane1 > 0, 1, -1)

# define hyperplane 2 (part b)
hyperplane2 <- -2 + x[, 1] + 2*x[, 2]
y2 <- ifelse(hyperplane2 > 0, 1, -1)

par(mfrow = c(1, 2))

# plot hyperplanes
plot1 <- ggplot(data.frame(x = x,
                  y = factor(y, levels = c(-1, 1))), aes(x.1, x.2, color=y)) +
            geom_point(show.legend = FALSE) +
            geom_abline(intercept = 1,
                        slope = 3,
                        linetype = 'dashed') +
            ggtitle("Binary Class Data | Hyperplane A") +
            xlab("X1") +
            ylab("X2") +
            labs(color = "Class")

plot2 <- ggplot(data.frame(x = x,
                  y = factor(y2, levels = c(-1, 1))), aes(x.1, x.2, color=y)) +
            geom_point() +
            geom_abline(intercept = 1,
                        slope = -0.5,
                        linetype = 'dashed') +
            ggtitle("Binary Class Data | Hyperplane B") +
            xlab("X1") +
            ylab("X2") +
            labs(color = "Class")
```

```r
grid.arrange(plot1, plot2, ncol = 2)
```

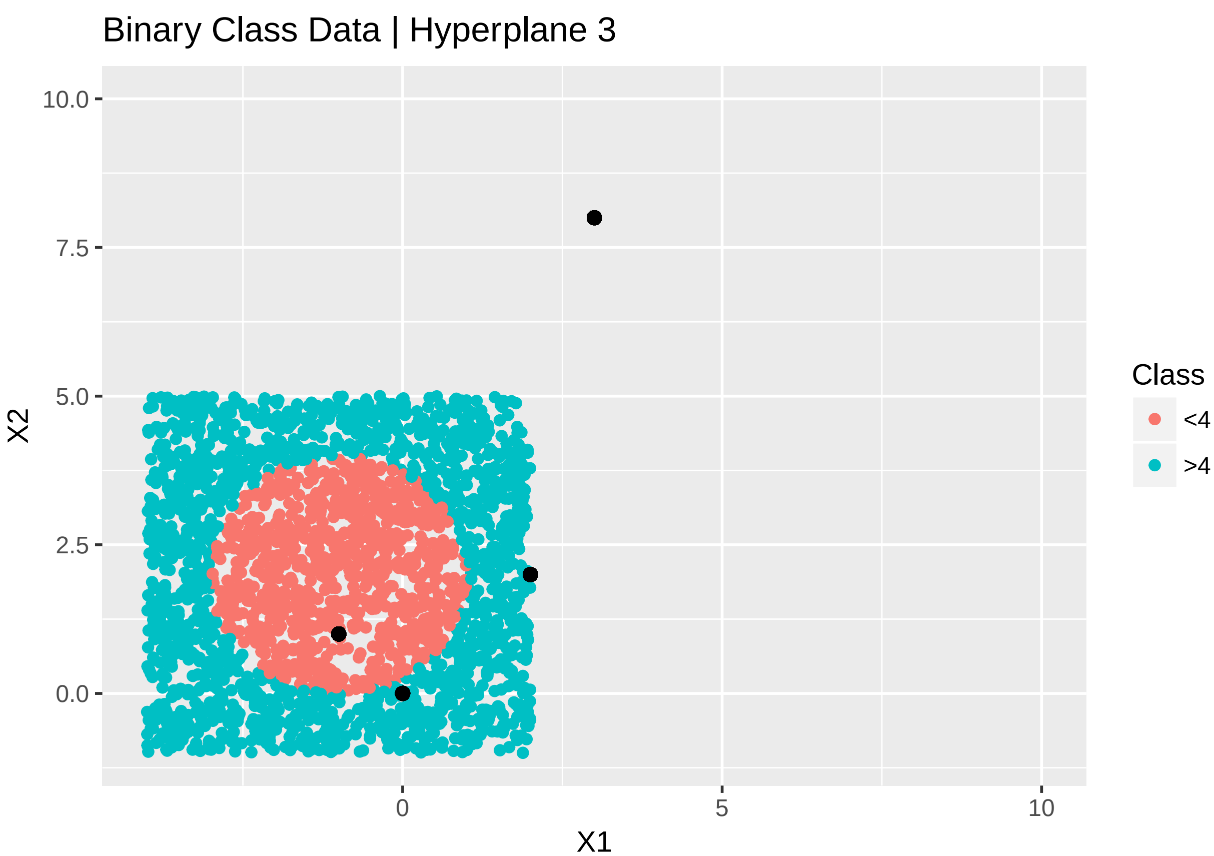


2

- **A, B & C**. The hyperplane is the circle encompassing the pink/orange data points below, where those data points are the ones who's value, when plugged into the equation $f(X_1, X_2) = (1 + X_1)^2 + (2 - X_2)^2 - 4$, will be negative. The blue data points output value of the above equation would be positive. The 4 data points plotted in black are those requested in part **C**, and it is clear which class they would fall into.

```r
# data setup
x_1 <- runif(2500, min = -4, max = 2)
x_2 <- runif(2500, min = -1, max = 5)

# define hyperplane
hyperplane3 <- (1 + x_1)^2 + (2 - x_2)^2 - 4
y3 <- factor(ifelse(hyperplane3 > 0, ">4", "<4"))

df <- data.frame(x1 = x_1,
                 x2 = x_2,
                 y = y3)

# plot hyperplane with additional points in black
ggplot(df, aes(x1, x2, color=y)) +
```

```r
  geom_point() +
  geom_point(x = 0,
             y = 0,
             col = 'black',
             cex = 2) +
  geom_point(x = -1,
             y = 1,
             col = 'black',
             cex = 2) +
  geom_point(x = 2,
             y = 2,
             col = 'black',
             cex = 2) +
  geom_point(x = 3,
             y = 8,
             col = 'black',
             cex = 2) +
  ggtitle("Binary Class Data | Hyperplane 3") +
  scale_y_continuous(limits = c(-1, 10)) +
  scale_x_continuous(limits = c(-4, 10)) +
  xlab("X1") +
  ylab("X2") +
  labs(color = "Class")
```



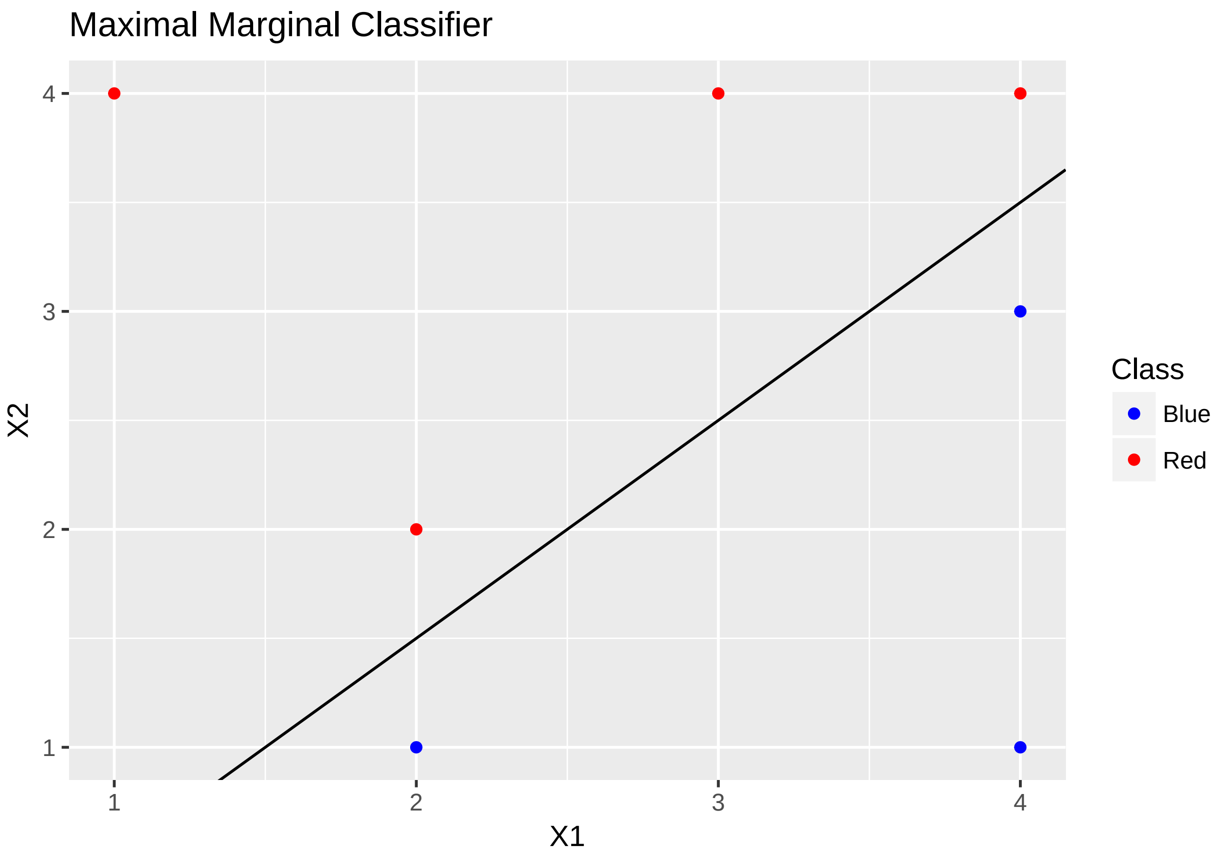
- **D**. One can see that the equation given in the text is non-linear with regard to X_1 and X_2. However, when expanded and refactored, it is clear that the hyperplane is linear with regard to to X_1, X_2, X_1^2 and X_2^2.

$$\begin{aligned}
(1 + X_1)^2 + (2 - X_2)^2 &= 4 && (1)\\
(1 + X_1)^2 + (2 - X_2)^2 - 4 &= 0 && (2)\\
(1 + 2X_1 + X_1^2) + (4 - 4X_2 + X_2^2) - 4 &= 0 && (3)\\
1 + 2X_1 + X_1^2 - 4X_2 + X_2^2 &= 0 && (4)\\
& && (5)
\end{aligned}$$

3

- **A & B**. The separating hyperplane has the equation $X_1 - X_2 - 0.5 = 0$.

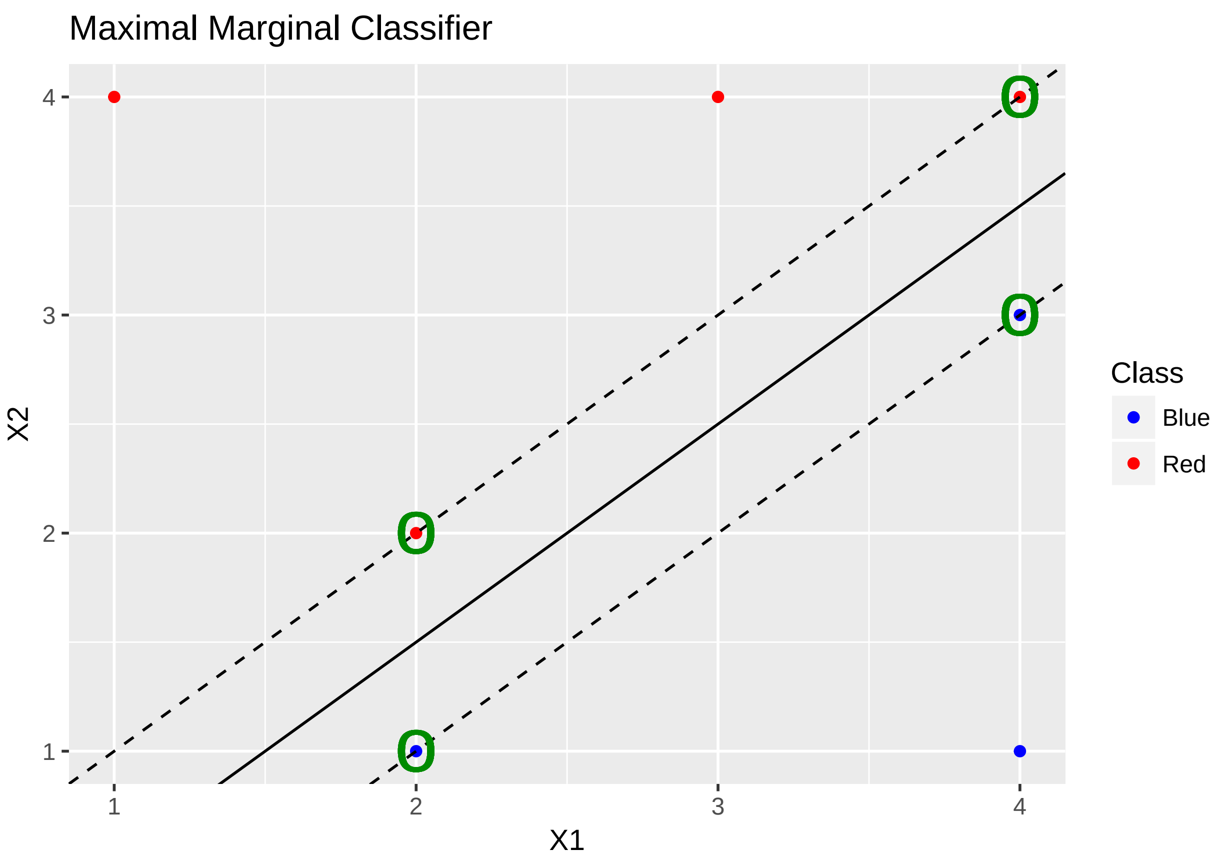
```r
# data setup
df <- data.frame(x_1 = c(3,2,4,1,2,4,4),
                 x_2 = c(4,2,4,4,1,3,1),
                 y = c(rep("Red", 4), rep("Blue", 3)))

# plot MMC
ggplot(df, aes(x_1, x_2, color = y)) +
    geom_point() +
    geom_abline(intercept = -0.5,
                slope = 1,
                linetype = 'solid') +
    scale_color_manual(values = c("Red" = 'red', "Blue" = 'blue')) +
    ggtitle("Maximal Marginal Classifier") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```



- **C**. Classify a test observation x_t to Red if $(1)X_1 + (-1)X_2 - 0.5 < 0$, otherwise classify to Blue.
- **D**. The support vectors are those data points circled in green, the edge of the margins are the dashed lines.

```
# illustrate support vectors and margins
ggplot(df, aes(x_1, x_2, color = y)) +
    geom_point() +
    geom_abline(intercept = -0.5,
                slope = 1,
                linetype = 'solid') +
    geom_abline(intercept = -0,
                slope = 1,
                linetype = 'dashed') +
    geom_abline(intercept = -1,
                slope = 1,
                linetype = 'dashed') +
    geom_point(x = 2,
               y = 2,
               color = 'green4',
               size = 10,
               shape = "o") +
    geom_point(x = 2,
               y = 1,
               color = 'green4',
```

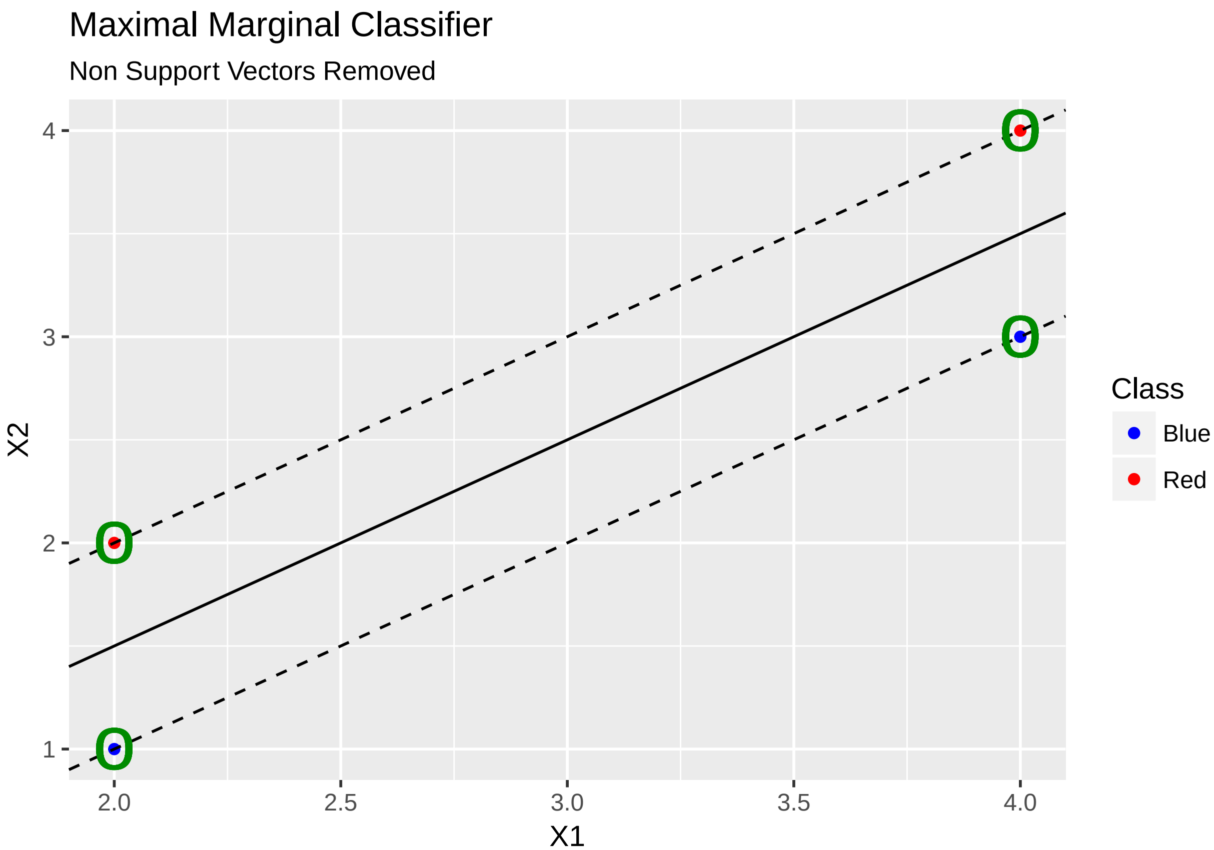
```r
               size = 10,
               shape = "o") +
    geom_point(x = 4,
               y = 3,
               color = 'green4',
               size = 10,
               shape = "o") +
    geom_point(x = 4,
               y = 4,
               color = 'green4',
               size = 10,
               shape = "o") +
    scale_color_manual(values = c("Red" = 'red', "Blue" = 'blue')) +
    ggtitle("Maximal Marginal Classifier") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```



- **F**. Not only can the seventh observation be removed from the data set without affecting the hyperplane, **any observation that is not a support vector can be removed without distorting the hyperplane,** as shown in the plot below.

```r
# remove non support vectors
sub_df <- df[-c(1, 4, 7), ]
```

```r
# remove non support vectors
ggplot(sub_df, aes(x_1, x_2, color = y)) +
    geom_point() +
    geom_abline(intercept = -0.5,
                slope = 1,
                linetype = 'solid') +
    geom_abline(intercept = -0,
                slope = 1,
                linetype = 'dashed') +
    geom_abline(intercept = -1,
                slope = 1,
                linetype = 'dashed') +
    geom_point(x = 2,
               y = 2,
               color = 'green4',
               size = 10,
               shape = "o") +
    geom_point(x = 2,
               y = 1,
               color = 'green4',
               size = 10,
               shape = "o") +
    geom_point(x = 4,
               y = 3,
               color = 'green4',
               size = 10,
               shape = "o") +
    geom_point(x = 4,
               y = 4,
               color = 'green4',
               size = 10,
               shape = "o") +
    scale_color_manual(values = c("Red" = 'red', "Blue" = 'blue')) +
    ggtitle("Maximal Marginal Classifier",
            "Non Support Vectors Removed") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```

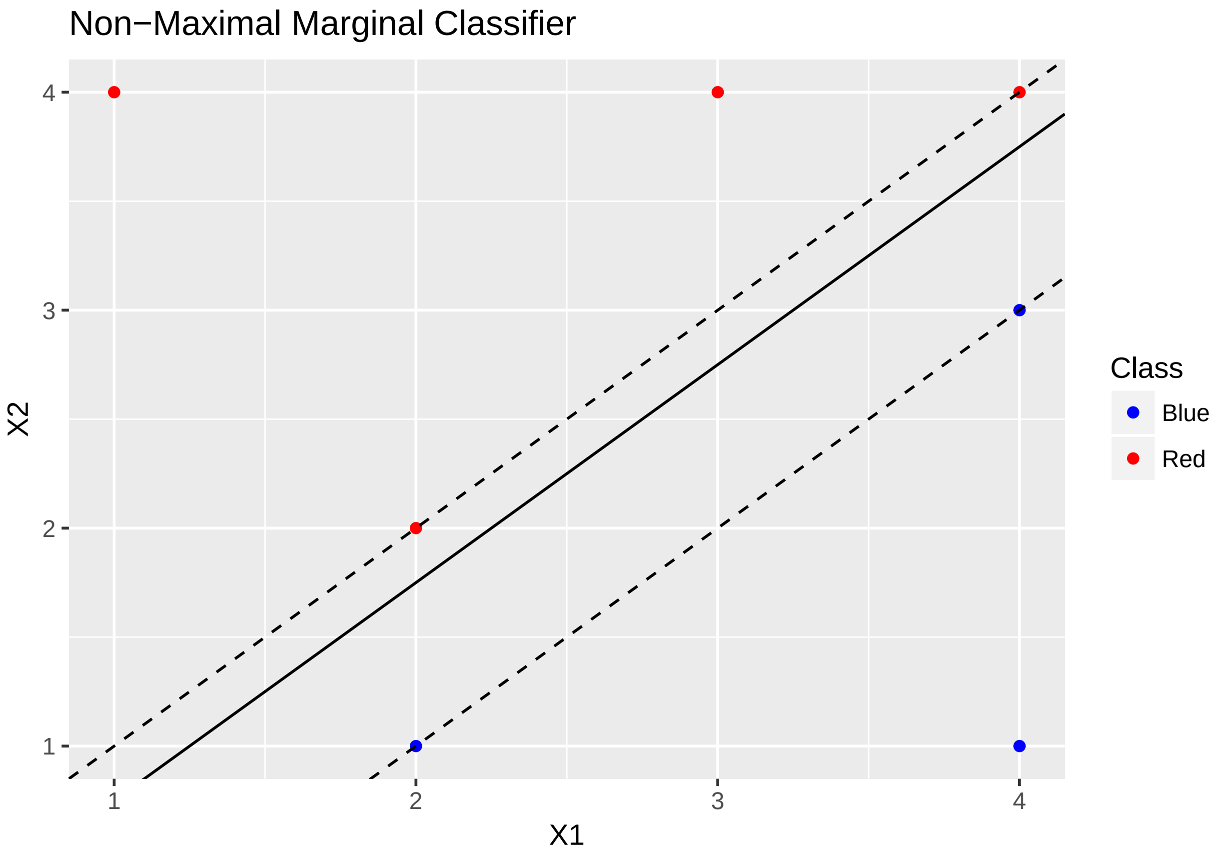


- **G**. The below hyperplane, with the equation $(1)X_1 + (-1)X_2 - 0.25$ would not be the **maximal** separating hyperplane because the margin between the red support vectors and the hyperplane is smaller than the margin between the blue support vectors and the machine.

  The reason that this is less desirable than the Maximal Marginal Classifier is because the hyperplane, *in this scenario*, is favoring the blue data points, **with no "supporting" evidence (pun intended)**. It is giving them (the blue support vectors) a wider berth than is necessary, at the cost of the berth to the red support vectors.

```r
# illustrate non-maximal margin classifier
ggplot(df, aes(x_1, x_2, color = y)) +
    geom_point() +
    geom_abline(intercept = -0.25,
                slope = 1,
                linetype = 'solid') +
    geom_abline(intercept = -0,
                slope = 1,
                linetype = 'dashed') +
    geom_abline(intercept = -1,
                slope = 1,
                linetype = 'dashed') +
    scale_color_manual(values = c("Red" = 'red', "Blue" = 'blue')) +
    ggtitle("Non-Maximal Marginal Classifier") +
    xlab("X1") +
    ylab("X2") +
```

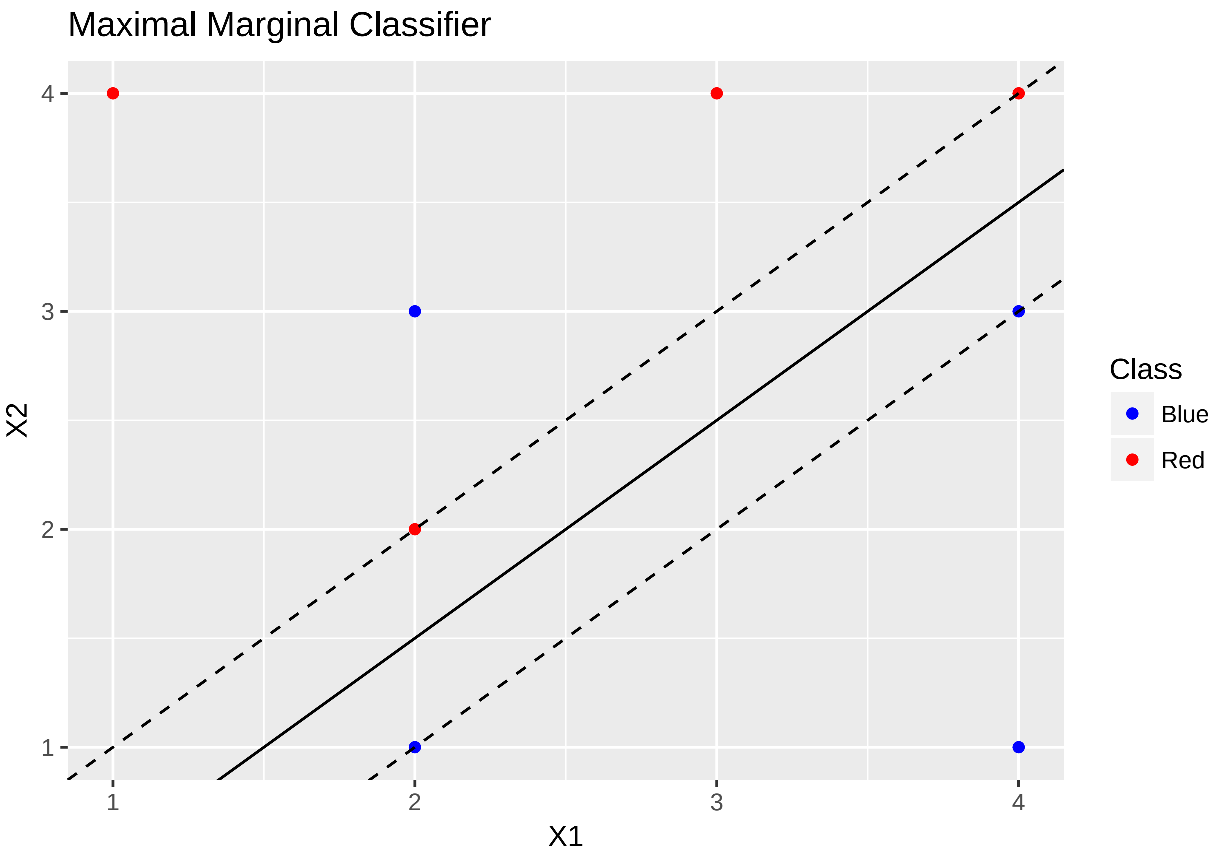
```
    labs(color = 'Class')
```



- **H**. A data set where the Maximal Marginal Classifier does not exist.

```
new.point <- data.frame(x_1 = 2,
                        x_2 = 3,
                        y = "Blue")
df <- rbind(df, new.point)

# illustrate tha data set where a non-maximal is not possible
ggplot(df, aes(x_1, x_2, color = y)) +
    geom_point() +
    geom_abline(intercept = -0.5,
                slope = 1,
                linetype = 'solid') +
    geom_abline(intercept = -0,
                slope = 1,
                linetype = 'dashed') +
    geom_abline(intercept = -1,
                slope = 1,
                linetype = 'dashed') +
    scale_color_manual(values = c("Red" = 'red', "Blue" = 'blue')) +
    ggtitle("Maximal Marginal Classifier") +
    xlab("X1") +
```

```
    ylab("X2") +
    labs(color = 'Class')
```



Applied

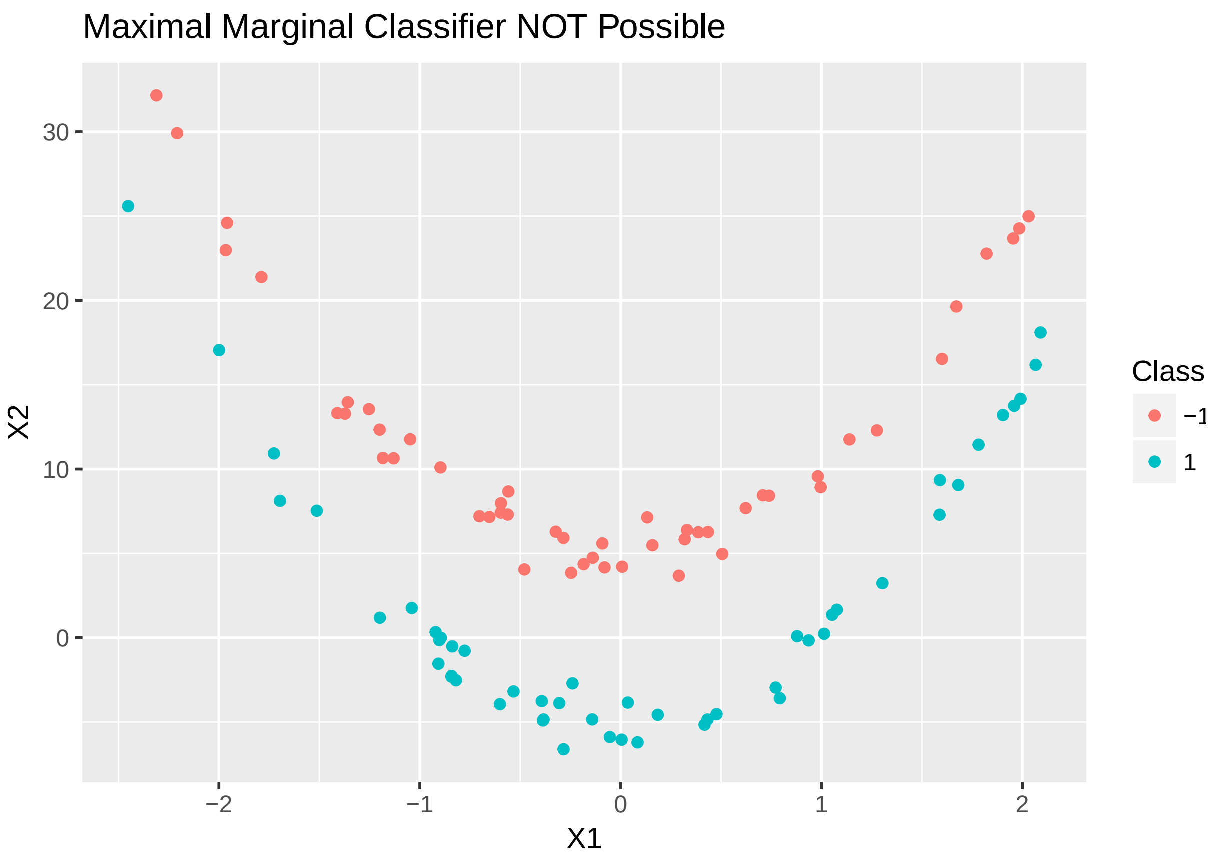
4

As shown below, a linear decision boundary is not possible with the data provided. Knowing this, it is unlikely that a linear SVM would outperform a more complex model. This intuition is confirmed in the following three plots, showing that the radial SVM (with $\gamma = 2$) is the model that fits the training data the best.

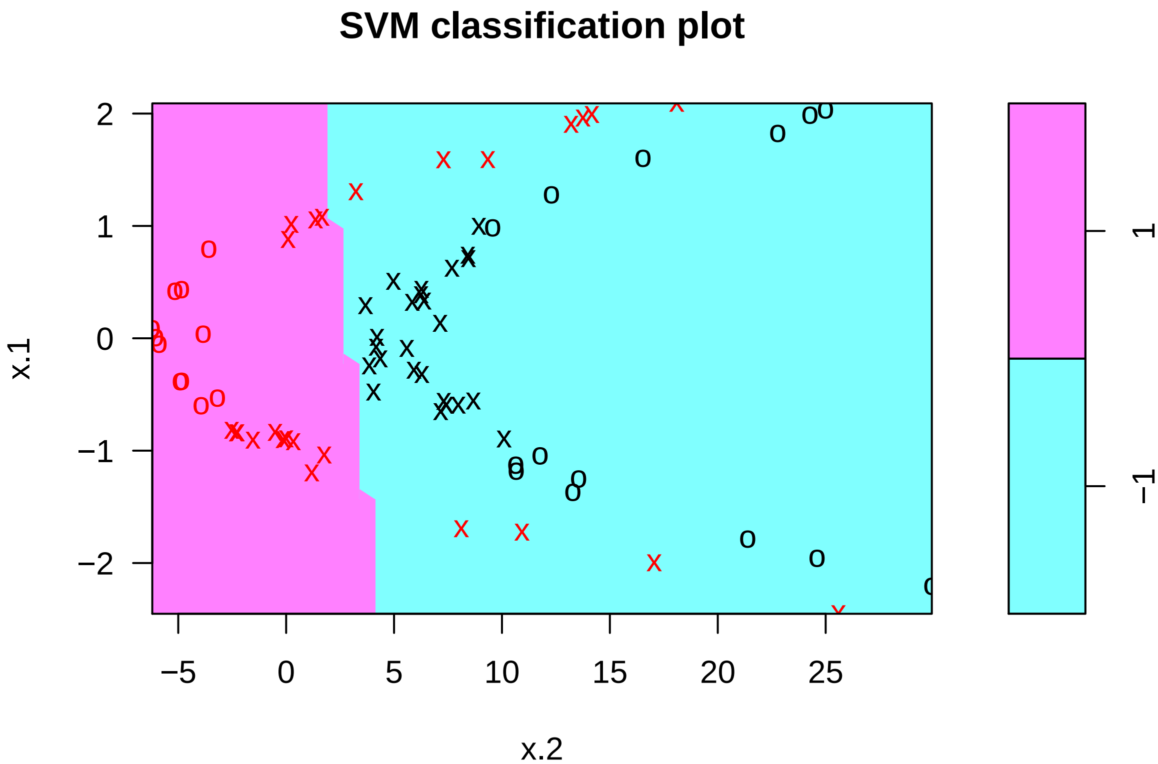
Not surprisingly, the radial SVM outperforms the linear and polynomial SVM's on the test set as well. Interestingly, the radial SVM also has fewer support vectors, therefore making the model more computationally efficient to store.

```
suppressPackageStartupMessages(library(e1071))
suppressPackageStartupMessages(library(caret))
set.seed(2)

# data setup
x_1 <- rnorm(100)
x_2 <- 5*(x_1^2) + rnorm(100)
idx <- sample(100, 50)
```

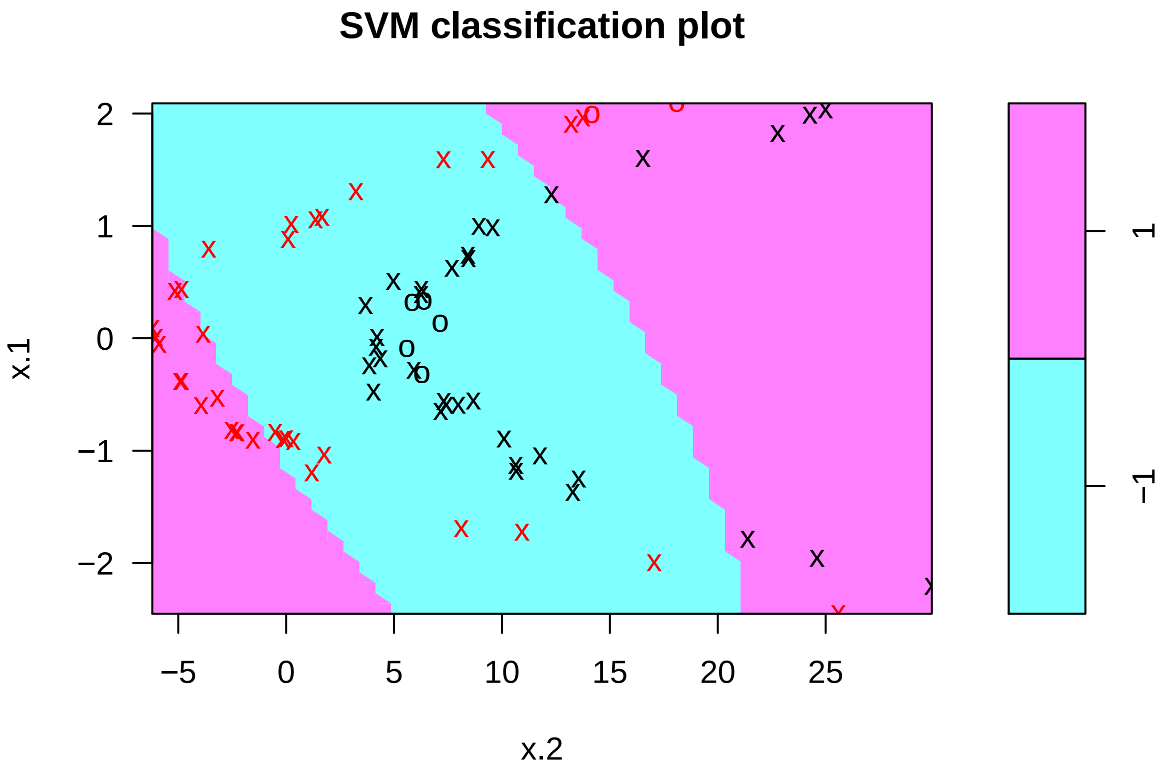
```r
x_2[idx] <- x_2[idx] + 5
x_2[-idx] <- x_2[-idx] - 5
y <- factor(ifelse(x_2 > 5*(x_1^2) + rnorm(100), -1, 1), levels = c(-1, 1))

df <- data.frame(x.1 = x_1,
                 x.2 = x_2,
                 y = y)

# plot data
ggplot(df, aes(x.1, x.2, color = y)) +
    geom_point() +
    ggtitle("Maximal Marginal Classifier NOT Possible") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```



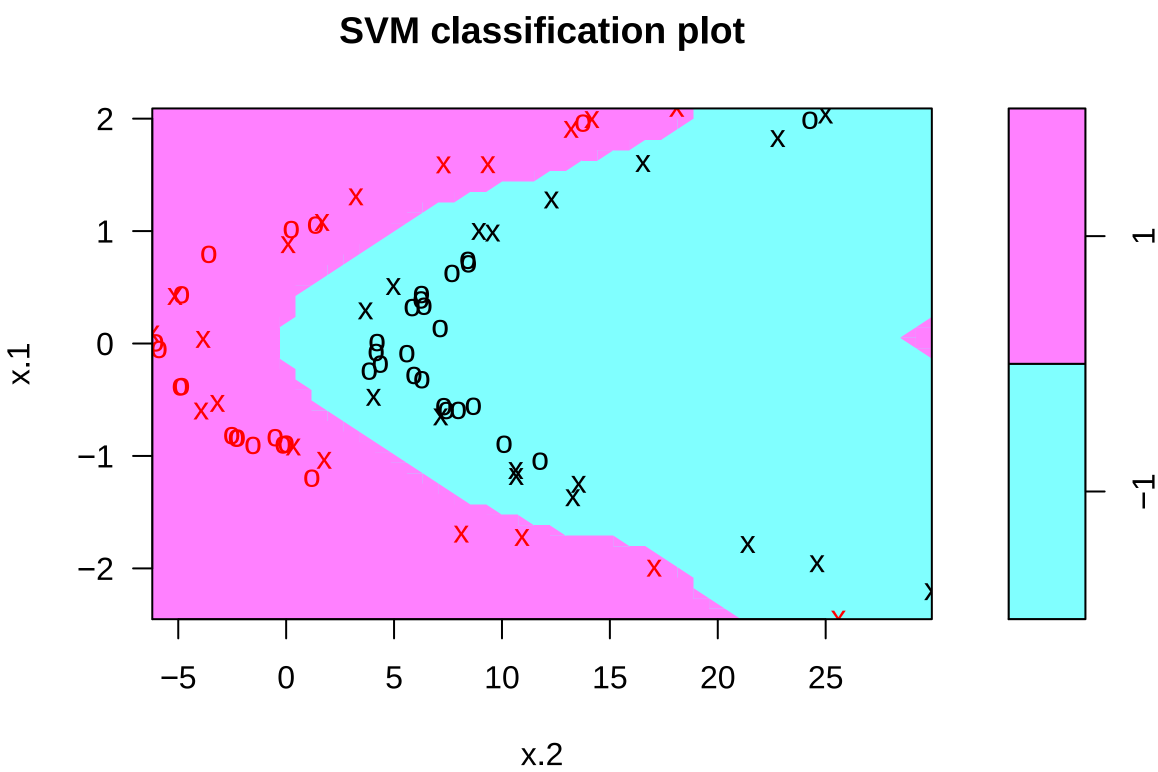

```r
set.seed(5)
train <- sample(100, 75)

# linear support vector machine
linear.svm <- svm(y ~ .,
                  data = df[train,],
                  kernel = 'linear')
plot(linear.svm, df[train,])
```




```r
# polynomial support vector machine
poly.svm <- svm(y ~ .,
                data = df[train,],
                kernel = 'polynomial',
                degree = 2)
plot(poly.svm, df[train,])
```




```r
# radial support vector machine
radial.svm <- svm(y ~ .,
                  data = df[train,],
                  kernel = 'radial',
                  gamma = 2)
plot(radial.svm, df[train,])
```




```r
# prediction
linear.svm.yhat <- predict(linear.svm, newdata = df[-train, ])
poly.svm.yhat <- predict(poly.svm, newdata = df[-train, ])
radial.svm.yhat <- predict(radial.svm, newdata = df[-train, ])

linear.svm.acc <- confusionMatrix(df[-train, 'y'], linear.svm.yhat)$overall[1]
poly.svm.acc <- confusionMatrix(df[-train, 'y'], poly.svm.yhat)$overall[1]
radial.svm.acc <- confusionMatrix(df[-train, 'y'], radial.svm.yhat)$overall[1]

print(paste("Linear SVM test accuracy =",
            linear.svm.acc,
            "with",
            linear.svm$tot.nSV,
            "support vectors."))
```

```
## [1] "Linear SVM test accuracy = 0.84 with 50 support vectors."
```

```r
print(paste("Polynomial SVM test accuracy =",
            poly.svm.acc,
            "with",
            poly.svm$tot.nSV,
            "support vectors."))
```

```
## [1] "Polynomial SVM test accuracy = 0.64 with 68 support vectors."
```

```r
print(paste("Radial SVM test accuracy =",
            radial.svm.acc,
            "with",
            radial.svm$tot.nSV,
            "support vectors."))
```

```
## [1] "Radial SVM test accuracy = 1 with 36 support vectors."
```

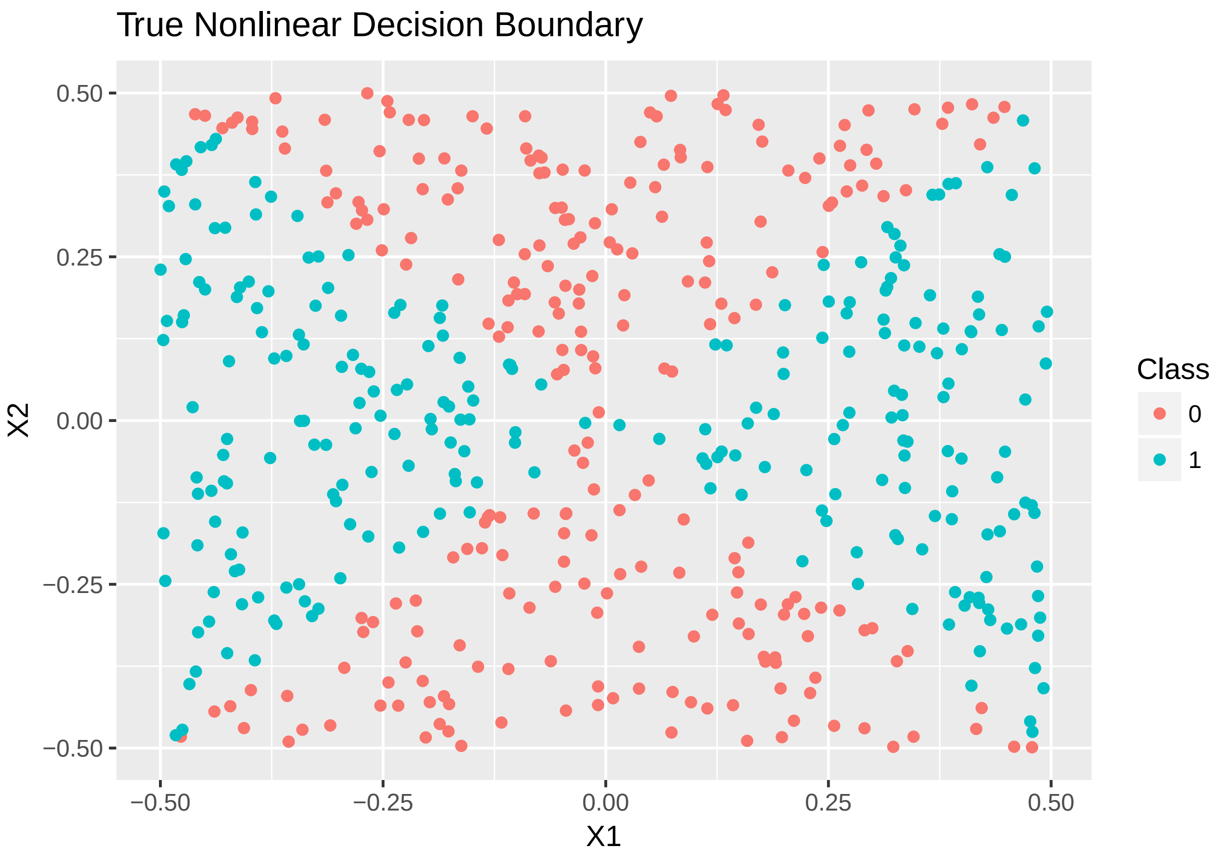
5

- **A.**

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

- **B.**

```r
df <- data.frame(x1 = x1,
                 x2 = x2,
                 y = factor(y, levels = c(0, 1)))

ggplot(df, aes(x1, x2, color = y)) +
    geom_point() +
    ggtitle("True Nonlinear Decision Boundary") +
    xlab("X1") +
    ylab("X2") +
    labs(color = "Class")
```

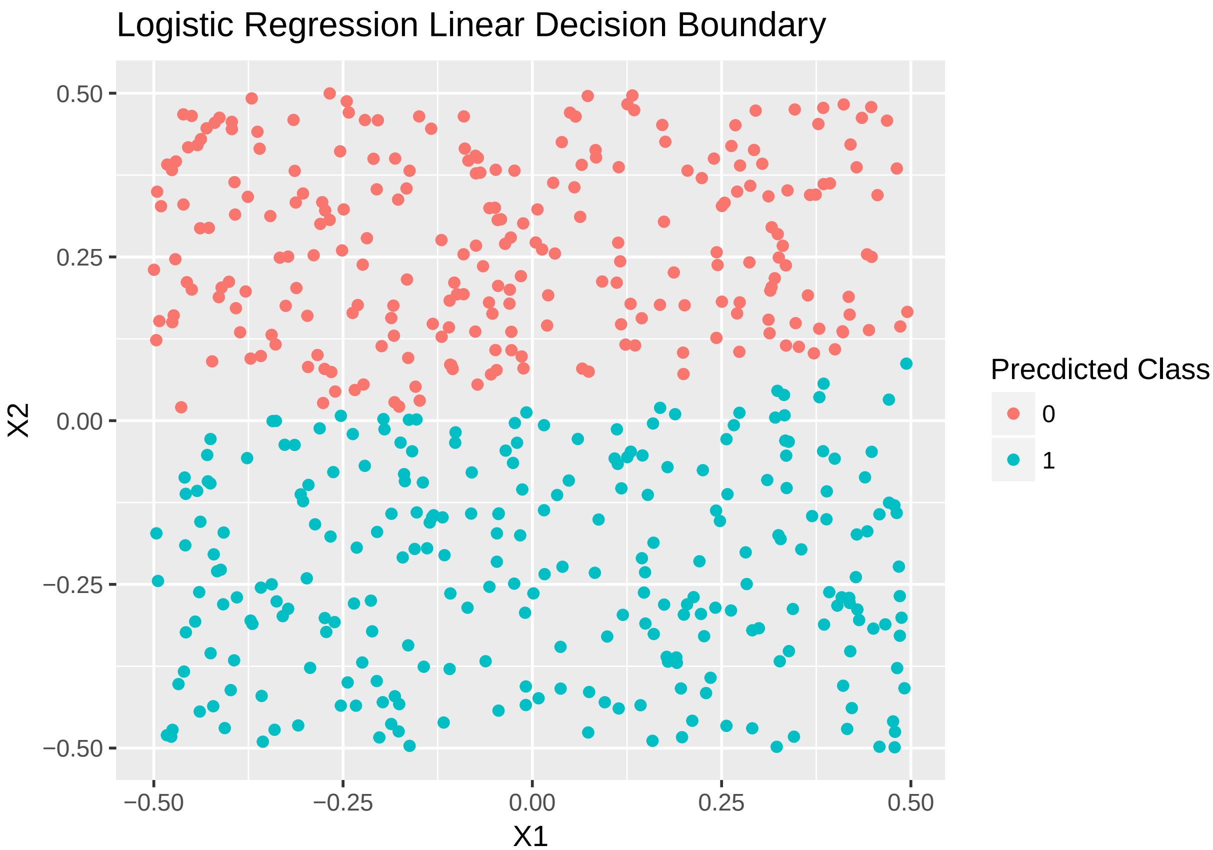


- **C.**

```
log.mod <- glm(y ~ .,
               data = df,
               family = 'binomial')
```

- **D.**

```
log.mod.y.prob <- predict(log.mod, newdata = df, type = "response")
threshold <- 0.5
log.mod.y.hat <- ifelse(log.mod.y.prob >= threshold, 1, 0)

df$y_hat <- factor(log.mod.y.hat, levels = c(0, 1))

ggplot(df, aes(x1, x2, color = y_hat)) +
    geom_point() +
    ggtitle("Logistic Regression Linear Decision Boundary") +
    xlab("X1") +
    ylab("X2") +
    labs(color = "Precdicted Class")
```



- **E**.

```
non.linear.log.mod <- glm(y ~ poly(x1, 2) + poly(x2, 2) + x1*x2,
                          data = df,
                          family = 'binomial')
```

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

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

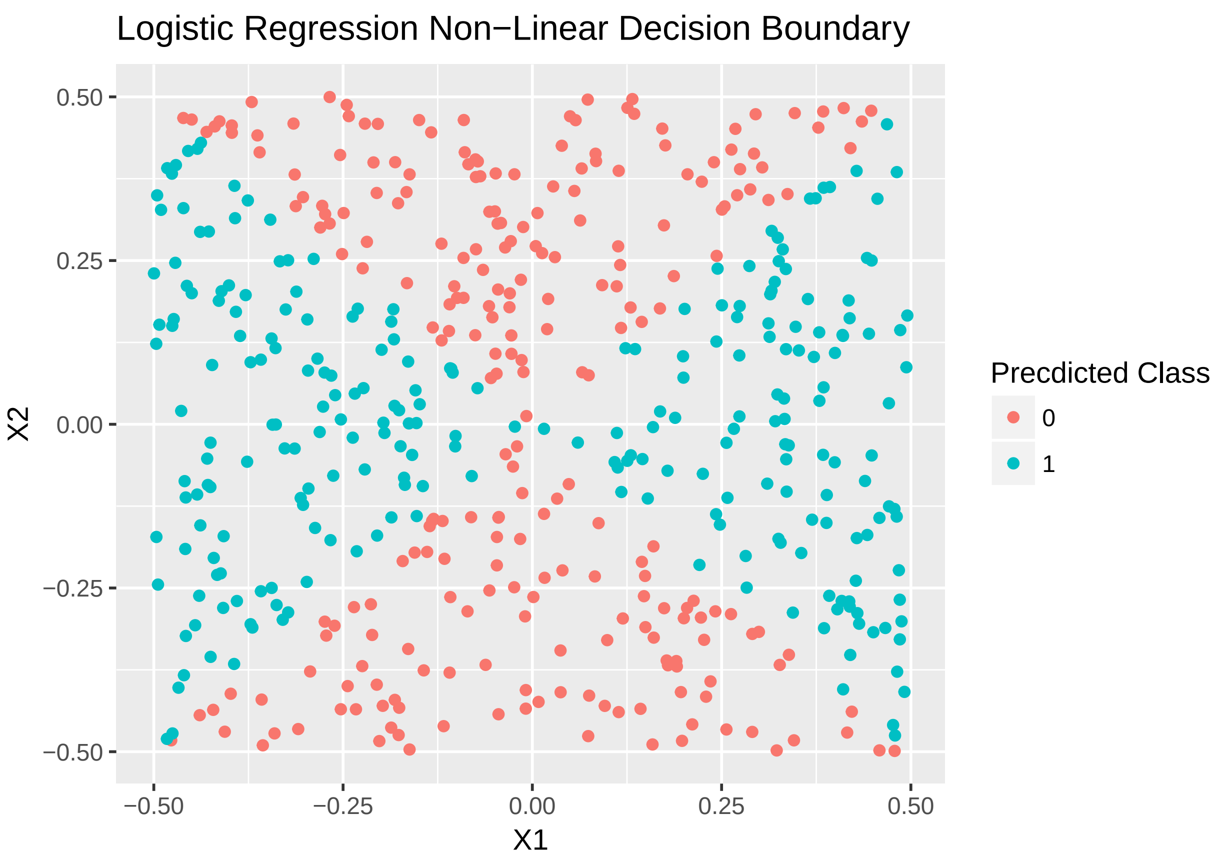
- **F**.

```
non.linear.log.mod.y.prob <- predict(non.linear.log.mod,
                                     newdata = df,
                                     type = 'response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

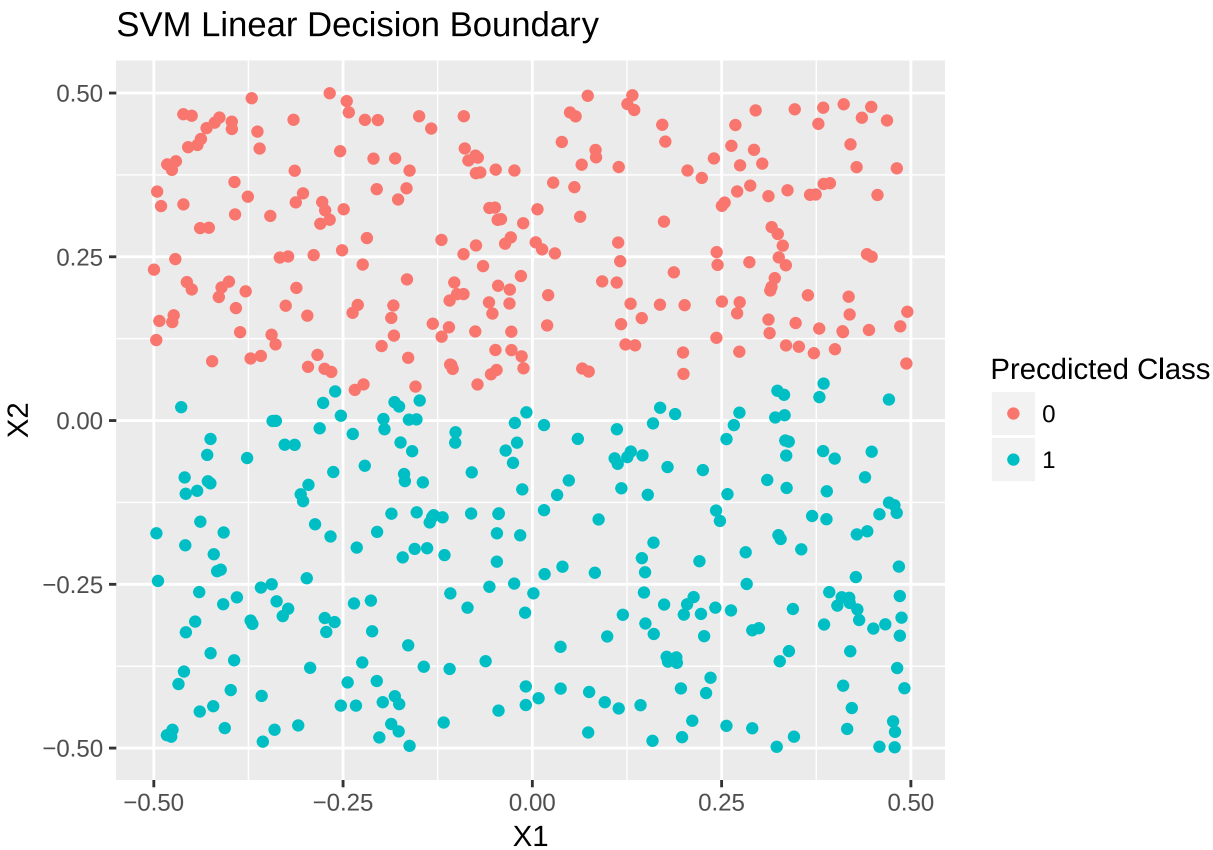
```
threshold <- 0.5
non.linear.log.mod.y.hat <- ifelse(non.linear.log.mod.y.prob >= threshold,
                                   1, 0)

df$non_lin_y_hat <- factor(non.linear.log.mod.y.hat, levels = c(0, 1))
```

```r
ggplot(df, aes(x1, x2, color = non_lin_y_hat)) +
    geom_point() +
    ggtitle("Logistic Regression Non-Linear Decision Boundary") +
    xlab("X1") +
    ylab("X2") +
    labs(color = "Precdicted Class")
```



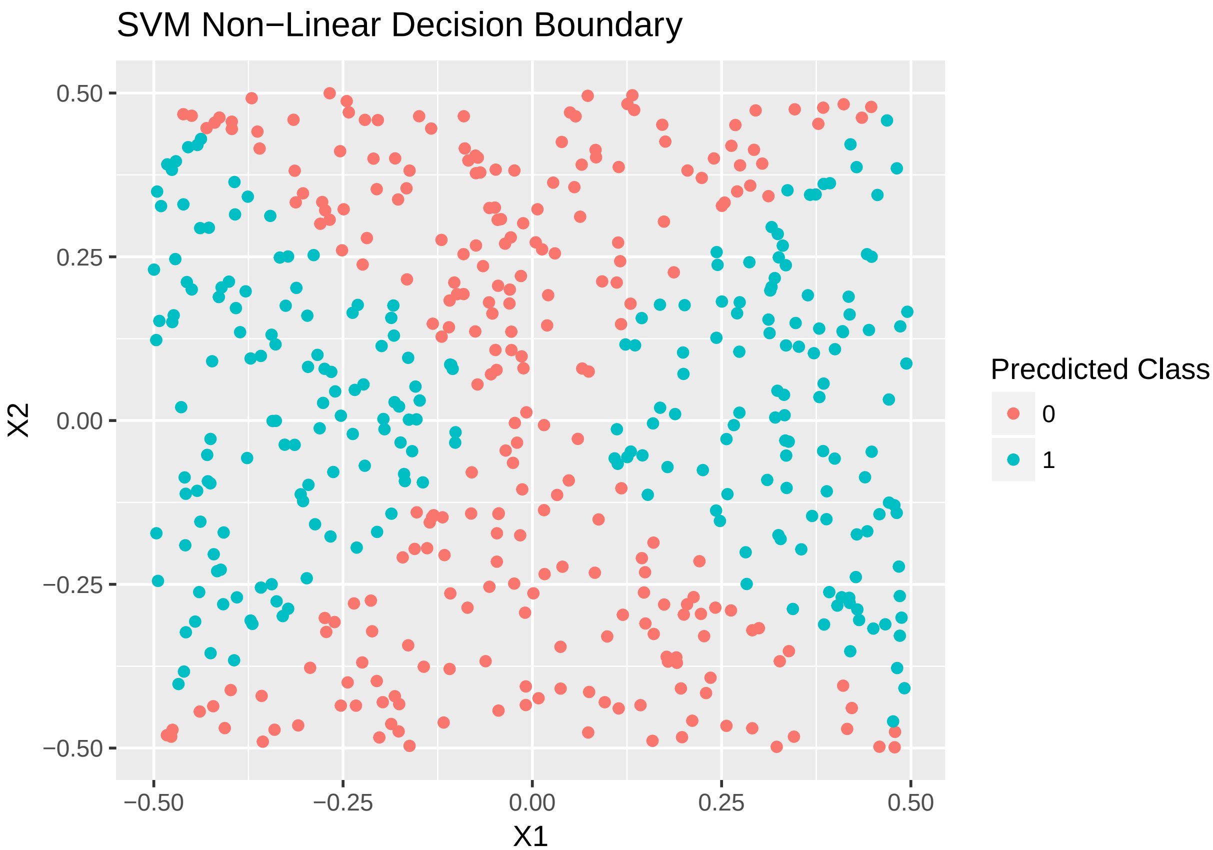
- **G**.

```r
svm.mod <- svm(y ~ x1 + x2,
               data = df,
               kernel = 'linear')
svm.mod.y.hat <- predict(svm.mod, newdata = df)

df$svm_y_hat <- svm.mod.y.hat

ggplot(df, aes(x1, x2, color = svm_y_hat)) +
    geom_point() +
    ggtitle("SVM Linear Decision Boundary") +
    xlab("X1") +
    ylab("X2") +
    labs(color = "Precdicted Class")
```



- **H.**

```
svm.mod.nonlin <- svm(y ~ x1 + x2,
               data = df,
               kernel = 'polynomial',
               degree = 2)
svm.mod.nonlin.y.hat <- predict(svm.mod.nonlin, newdata = df)

df$svm_y_hat_nonlin <- svm.mod.nonlin.y.hat

ggplot(df, aes(x1, x2, color = svm_y_hat_nonlin)) +
    geom_point() +
    ggtitle("SVM Non-Linear Decision Boundary") +
    xlab("X1") +
    ylab("X2") +
    labs(color = "Precdicted Class")
```


SVM Non−Linear Decision Boundary
0.50
0.25
0.00
−0.25
−0.50
X2
−0.50
−0.25
0.00
0.25
0.50
X1
Precdicted Class
0
1

- **I**. The comparison between SVM's and Logistic Regression above illustrates an interesting concept when seeking to model some natural phenomena; instead of toggling the different statistical learning methods that one would use to model said phenomena, *transforming the attributes that are fed into a statistical learning method can have a similar affect on modeling that phenomena.*

  Typically, one is given a matrix $\mathbf{X}$ and a vector y, and a variety of models/functions, $f_1(\mathbf{X}, y),\ f_2(\mathbf{X}, y),\ f_3(\mathbf{X}, y),\ f_j(\mathbf{X}, y)$, are used to try and model the relationship between $\mathbf{X}$ and y.

  However, as introduced in chapter 7, *basis functions* can be used to transform the attributes of $\mathbf{X}$ into a new matrix, $b(\mathbf{X})$, where $b_j(x_j)$ is a mathematical transformation of the j^{th} column of $\mathbf{X}$ (illustrated below).

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,p} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \cdots & x_{n,p} \end{pmatrix}$$

$$b(\mathbf{X}) = \begin{pmatrix} b_1(x_{1,1}) & b_2(x_{1,2}) & b_3(x_{1,3}) & \cdots & b_\infty(x_{1,\infty}) \\ b_1(x_{2,1}) & b_2(x_{2,2}) & b_3(x_{2,3}) & \cdots & b_\infty(x_{2,\infty}) \\ b_1(x_{3,1}) & b_2(x_{3,2}) & b_3(x_{3,3}) & \cdots & b_\infty(x_{3,\infty}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_1(x_{n,1}) & b_2(x_{n,2}) & b_3(x_{n,3}) & \cdots & b_\infty(x_{n,\infty}) \end{pmatrix}$$

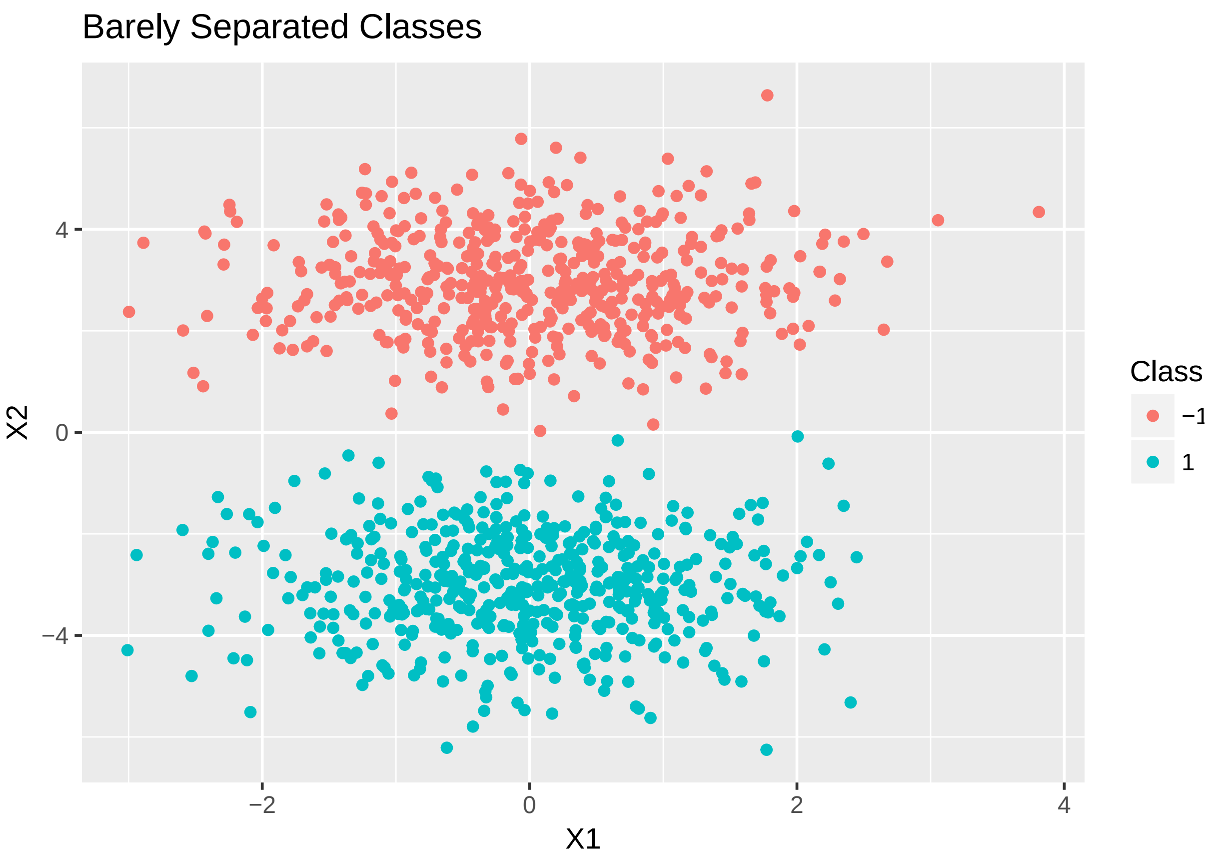
  This opens up up a whole new world of possibilities, where the challenge becomes finding the proper basis functions as opposed to finding the proper model framework. Once these basis functions are found, the model framework can vary and still produce similar results, as shown by pitting Logistic Regression against SVM's with the same basis functions, and obtaining similar decision boundaries.

6

- **A**.

```
# set seed for reproducibility
set.seed(1)
# randomly generate data from a Gaussian Distribution
x_1 <- rnorm(1000)
x_2 <- rnorm(1000)
# create an index that will be used to separate the classes
idx <- sample(1000, 500)
x_2[idx] <- x_2[idx] + 3
x_2[-idx] <- x_2[-idx] - 3

# create separating hyperplane at 0.5
hyperplane <- 0
y <- ifelse(x_2 <= hyperplane, 1, -1)

train.df <- data.frame(x1 = x_1,
                       x2 = x_2,
                       y = factor(y, levels = c(-1, 1)))
```

```r
# plot to illustrate barely separated classes
ggplot(train.df, aes(x1, x2, color = y)) +
    geom_point() +
    ggtitle("Barely Separated Classes") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```



- **B**. As one can see from the table below, cost values of 0.1, 1 and 2 all corresponded to 0 training errors.

```r
cost.values <- c(0.1, 1, 5, 10, 50, 100, 1000, 10000)

tuned.svm <- tune(svm,
                  y ~ .,
                  data = train.df,
                  kernel = 'linear',
                  ranges = list(cost = cost.values))

error.df <- data.frame(Cost = tuned.svm$performances$cost,
                       Train_MisClass = tuned.svm$performances$error * 1000)
error.df
```

```
##    Cost Train_MisClass
## 1 1e-01              2
```

```
## 2 1e+00              1
## 3 5e+00              2
## 4 1e+01              3
## 5 5e+01              4
## 6 1e+02              3
## 7 1e+03              0
## 8 1e+04              0
```

- **C**.

```r
# generate test data by changing seed
set.seed(5)
x_1 <- rnorm(1000)
x_2 <- rnorm(1000) + rnorm(1000)
idx <- sample(1000, 500)
x_2[idx] <- x_2[idx] + 1
x_2[-idx] <- x_2[-idx] - 1

hyperplane <- 0
y <- ifelse(x_2 <= hyperplane, 1, -1)

test.df <- data.frame(x1 = x_1,
                      x2 = x_2,
                      y = factor(y, levels = c(-1, 1)))

# plot to illustrate barely separated classes
ggplot(test.df, aes(x1, x2, color = y)) +
    geom_point() +
    ggtitle("Test Data") +
    xlab("X1") +
    ylab("X2") +
    labs(color = 'Class')
```


```
# prediction
test.errors <- rep(0, length(cost.values))
for (i in 1:length(cost.values)) {
    svm.mod <- svm(y ~ .,
                   data = train.df,
                   kernel = 'linear',
                   cost = cost.values[i])
    y.hat <- predict(svm.mod, newdata = test.df)
    test.errors[i] <- sum(y.hat != test.df$y)
}


error.df$Test_MisClass <- test.errors
error.df
```

```
##    Cost Train_MisClass Test_MisClass
## 1 1e-01              2            12
## 2 1e+00              1            11
## 3 5e+00              2             8
## 4 1e+01              3            28
## 5 5e+01              4            16
## 6 1e+02              3            24
## 7 1e+03              0            28
## 8 1e+04              0            28
```

- **D**. The point that is trying to be driven home here is one of the Bias-Variance Trade off; even if a Support Vector Classifier is able to correctly classifiy all *training* observations due to a high cost of violations to the margin/hyperplane, a *different* set of data (aka **test** data) might not be as cleanly separated as the training data, leading to overfitting if a high cost model is chosen. This is shown comparing the training misclassifications to the testing misclassifications, where the high cost SVM clearly overfits the data.

7

- **A**.

```r
suppressPackageStartupMessages(library(ISLR))
attach(Auto)
```

```
## The following object is masked from package:ggplot2:
##
##     mpg
```

```r
Auto$mileage <- factor(ifelse(Auto$mpg >= median(Auto$mpg), 1, 0))
```

- **B**.

```r
cv.results <- tune(svm,
                   mileage ~ . - mpg,
                   data = Auto,
                   kernel = 'linear',
                   ranges = list(cost = c(0.1, 1, 5, 10, 100, 1000)))
summary(cv.results)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.09455128
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-01 0.09455128 0.04220315
## 2 1e+00 0.09967949 0.04443956
## 3 5e+00 0.10730769 0.04678970
## 4 1e+01 0.11500000 0.04437783
## 5 1e+02 0.12237179 0.03485258
## 6 1e+03 0.11474359 0.05807958
```

- **c.**

```r
# polynomial SVM
poly.cv.results <- tune(svm,
                        mileage ~ . - mpg,
                        data = Auto,
                        kernel = 'polynomial',
                        ranges = list(
                            cost = c(1, 5, 10, 100, 1000),
                            degree = c(2,4,6)))
summary(poly.cv.results)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      2
##
## - best performance: 0.2984615
##
## - Detailed performance results:
##    cost degree     error dispersion
## 1     1      2 0.5612179 0.04287078
## 2     5      2 0.5612179 0.04287078
## 3    10      2 0.5149359 0.12038017
## 4   100      2 0.3087821 0.06896385
## 5  1000      2 0.2984615 0.07729504
## 6     1      4 0.5612179 0.04287078
## 7     5      4 0.5612179 0.04287078
## 8    10      4 0.5612179 0.04287078
## 9   100      4 0.5612179 0.04287078
## 10 1000      4 0.5612179 0.04287078
## 11    1      6 0.5612179 0.04287078
## 12    5      6 0.5612179 0.04287078
## 13   10      6 0.5612179 0.04287078
## 14  100      6 0.5612179 0.04287078
## 15 1000      6 0.5612179 0.04287078
```

```r
# radial SVM
radial.cv.results <- tune(svm,
                        mileage ~ . - mpg,
                        data = Auto,
                        kernel = 'radial',
                        ranges = list(
                            cost = c(1, 5, 10, 100, 1000),
                            gamma = c(2,4,6)))
summary(radial.cv.results)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     2
##
## - best performance: 0.1199359
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1     1     2 0.1199359 0.05643166
## 2     5     2 0.1199359 0.05643166
## 3    10     2 0.1199359 0.05643166
## 4   100     2 0.1199359 0.05643166
## 5  1000     2 0.1199359 0.05643166
## 6     1     4 0.4798077 0.03837270
## 7     5     4 0.4798077 0.04371238
## 8    10     4 0.4798077 0.04371238
## 9   100     4 0.4798077 0.04371238
## 10 1000     4 0.4798077 0.04371238
## 11    1     6 0.5026923 0.03385765
## 12    5     6 0.4950641 0.03241397
## 13   10     6 0.4950641 0.03241397
## 14  100     6 0.4950641 0.03241397
## 15 1000     6 0.4950641 0.03241397
```

```r
# compare models
linear.summary <- summary(cv.results)
poly.summary <- summary(poly.cv.results)
radial.summary <- summary(radial.cv.results)

print(paste("Linear SVM Error =", linear.summary$best.performance,
            "with cost =", linear.summary$best.parameters[1]))
```

```
## [1] "Linear SVM Error = 0.094551282051282 with cost = 0.1"
```

```r
print(paste("Polynomial SVM Error =",
            poly.summary$best.performance,
            "with degree =", poly.summary$best.parameters[2],
            "and cost =", poly.summary$best.parameters[1]))
```

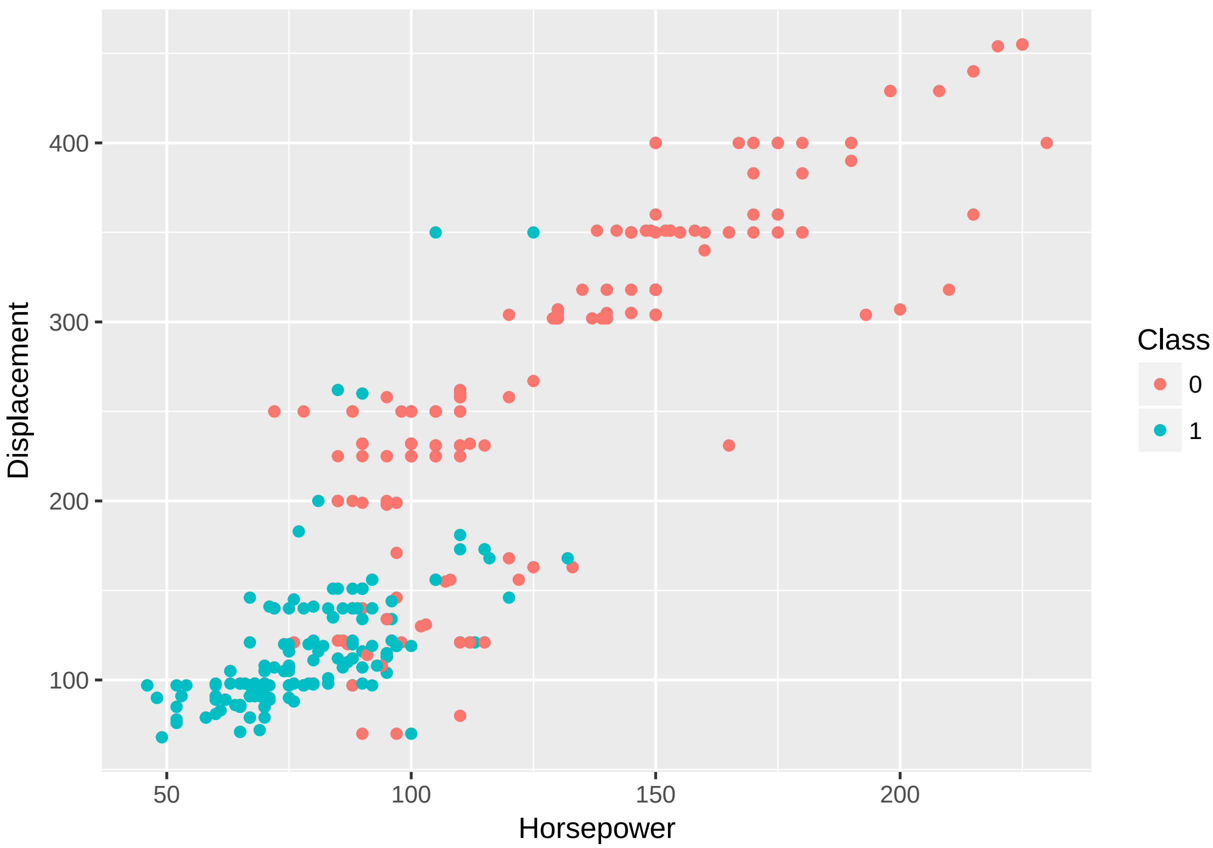
```
## [1] "Polynomial SVM Error = 0.298461538461538 with degree = 2 and cost = 1000"
```

```r
print(paste("Radial SVM Error |",
            radial.summary$best.performance,
            "with gamma =", radial.summary$best.parameters[2],
            "and cost =", radial.summary$best.parameters[1]))
```

```
## [1] "Radial SVM Error | 0.119935897435897 with gamma = 2 and cost = 1"
```

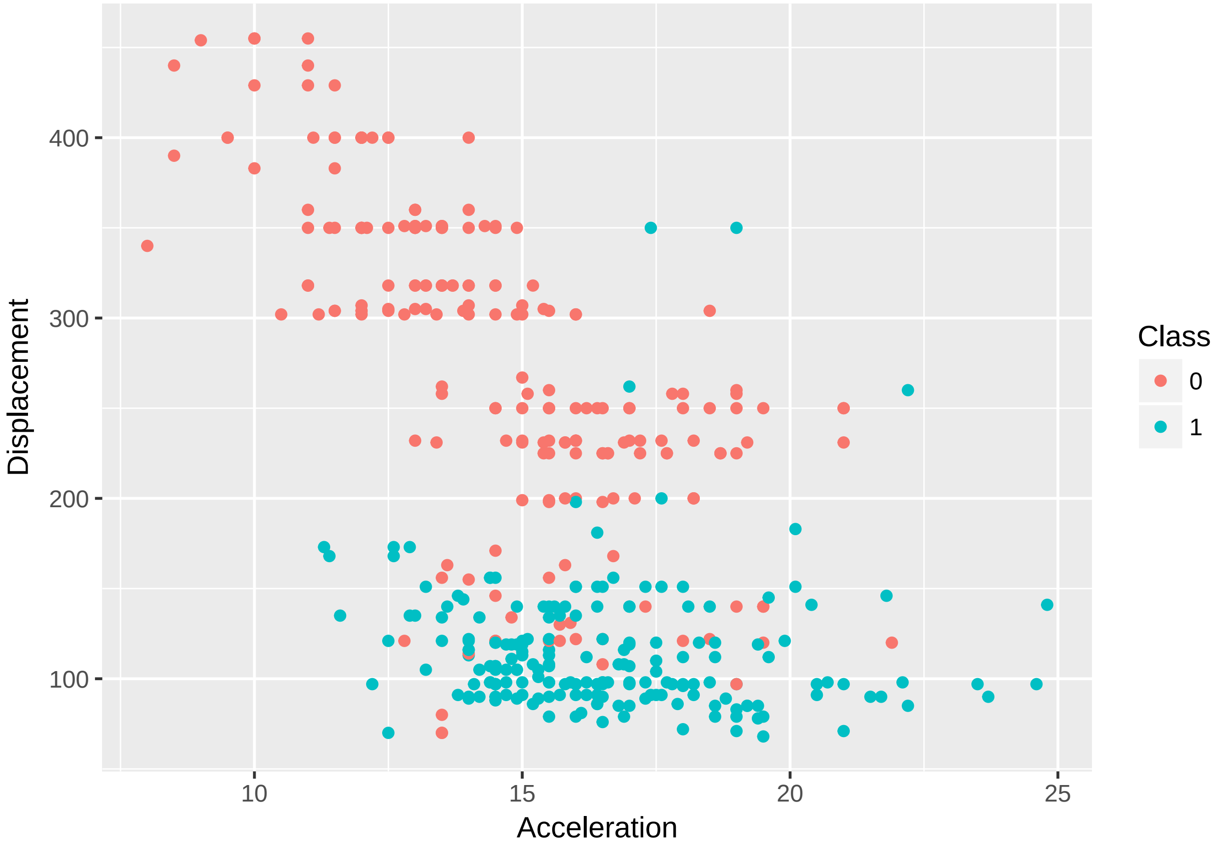
- **D**. With the linear kernel having the lowest testing error, this implies that the best dividing hyperplane can be found within the un-transformed feature space, with a few violations to the margin and/or hyperplane. The three plots below show the possible combinations of *Horsepower*, *Acceleration* and *Displacement* plotted against one another. There seems to be a marginally linear decision boundary in the final plot, although the true hyperplane is most likely in greater than two dimensions.

```
best.linear <- svm(mileage ~ . - mpg,
                   data = Auto,
                   kernel = 'linear',
                   cost = 1)
best.poly <- svm(mileage ~ . - mpg,
                 data = Auto,
                 kernel = 'polynomial',
                 degree = poly.summary$best.parameters[2],
                 cost = poly.summary$best.parameters[1])
best.radial <- svm(mileage ~ . - mpg,
                   data = Auto,
                   kernel = 'radial',
                   gamma = radial.summary$best.parameters[2],
                   cost = radial.summary$best.parameters[1])

# plot.svm not working for some reason...reverting to ggplot2
ggplot(Auto, aes(horsepower,
                 displacement,
                 color = factor(mileage, levels = c(0, 1)))) +
    geom_point() +
    xlab("Horsepower") +
    ylab("Displacement") +
    labs(color = "Class")
```

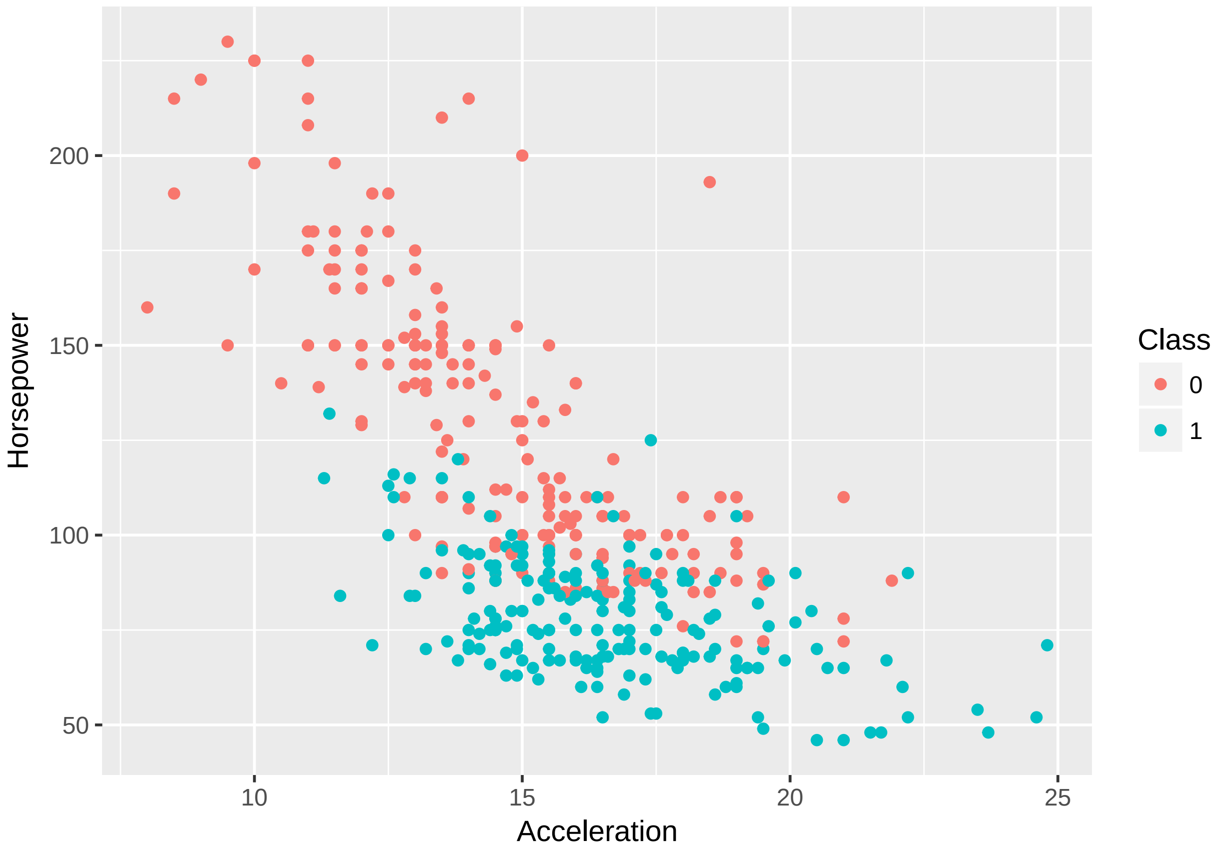



```
ggplot(Auto, aes(acceleration,
                 displacement,
```

```
                    color = factor(mileage, levels = c(0, 1)))) +
    geom_point() +
    xlab("Acceleration") +
    ylab("Displacement") +
    labs(color = "Class")
```




```
ggplot(Auto, aes(acceleration,
                 horsepower,
                 color = factor(mileage, levels = c(0, 1)))) +
    geom_point() +
    xlab("Acceleration") +
    ylab("Horsepower") +
    labs(color = "Class")
```



8

- **A**.

```
detach(Auto)
attach(OJ)

set.seed(5)
train <- sample(dim(OJ)[1], 800)
oj.train <- OJ[train,]
oj.test <- OJ[-train,]
```

- **B**. 219 of the 439 support vectors are of the CH (Citrus Hill) class, leaving 220 from the MM (Minute Maid) class.

```
svm.mod <- svm(Purchase ~ .,
               data = oj.train,
               kernel = 'linear',
               cost = 0.01)
summary(svm.mod)
```

```
##
## 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:  439
##
##  ( 219 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

- **C**. Interestingly, the test set predictions are more accurate than the training set predicitons, with an accuracy of 84.81% (as opposed to a training accuracy of 82.25%)

```
train.error <- predict(svm.mod, oj.train)
test.error <- predict(svm.mod, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 444  92
##         MM  50 214
##
##                Accuracy : 0.8225
##                  95% CI : (0.7942, 0.8484)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6142
##  Mcnemar's Test P-Value : 0.0005803
##
##             Sensitivity : 0.8988
##             Specificity : 0.6993
##          Pos Pred Value : 0.8284
##          Neg Pred Value : 0.8106
##              Prevalence : 0.6175
##          Detection Rate : 0.5550
```

```
##    Detection Prevalence : 0.6700
##       Balanced Accuracy : 0.7991
##
##        'Positive' Class : CH
##
```

```
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 143  25
##         MM  16  86
##
##                Accuracy : 0.8481
##                  95% CI : (0.7997, 0.8888)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6825
##  Mcnemar's Test P-Value : 0.2115
##
##             Sensitivity : 0.8994
##             Specificity : 0.7748
##          Pos Pred Value : 0.8512
##          Neg Pred Value : 0.8431
##              Prevalence : 0.5889
##          Detection Rate : 0.5296
##    Detection Prevalence : 0.6222
##       Balanced Accuracy : 0.8371
##
##        'Positive' Class : CH
##
```

- **D**.

```
cost.values <- c(0.01, 0.1, 1, 3, 5, 7, 10)
tune.out <- tune(svm,
                 Purchase ~ .,
                 data = oj.train,
                 kernel = 'linear',
                 ranges = list(cost = cost.values))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
```

```
##     1
##
## - best performance: 0.16625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.18000 0.04417453
## 2  0.10 0.17000 0.04571956
## 3  1.00 0.16625 0.04489571
## 4  3.00 0.16875 0.04458528
## 5  5.00 0.16625 0.04372023
## 6  7.00 0.16875 0.04573854
## 7 10.00 0.17000 0.04721405
```

- **E**. Using the tuned model (with a new cost of 1), the training error improves at the cost of the testing error.

```r
train.error <- predict(tune.out$best.model, oj.train)
test.error <- predict(tune.out$best.model, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 437  71
##         MM  57 235
##
##                Accuracy : 0.84
##                  95% CI : (0.8127, 0.8647)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6583
##  Mcnemar's Test P-Value : 0.2505
##
##             Sensitivity : 0.8846
##             Specificity : 0.7680
##          Pos Pred Value : 0.8602
##          Neg Pred Value : 0.8048
##              Prevalence : 0.6175
##          Detection Rate : 0.5463
##    Detection Prevalence : 0.6350
##       Balanced Accuracy : 0.8263
##
##        'Positive' Class : CH
##
```

```r
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 138  23
##         MM  21  88
##
##                Accuracy : 0.837
##                  95% CI : (0.7875, 0.879)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6625
##  Mcnemar's Test P-Value : 0.8802
##
##             Sensitivity : 0.8679
##             Specificity : 0.7928
##          Pos Pred Value : 0.8571
##          Neg Pred Value : 0.8073
##              Prevalence : 0.5889
##          Detection Rate : 0.5111
##    Detection Prevalence : 0.5963
##       Balanced Accuracy : 0.8304
##
##        'Positive' Class : CH
##
```

- **F**.

```r
svm.radial <- svm(Purchase ~ .,
                  data = oj.train,
                  kernel = 'radial',
                  cost = 0.01)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  616
##
##  ( 310 306 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# prediction
train.error <- predict(svm.radial, oj.train)
test.error <- predict(svm.radial, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 494 306
##         MM   0   0
##
##                Accuracy : 0.6175
##                  95% CI : (0.5828, 0.6513)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : 0.5156
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.6175
##          Neg Pred Value :    NaN
##              Prevalence : 0.6175
##          Detection Rate : 0.6175
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : CH
##
```

```r
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 159 111
##         MM   0   0
##
##                Accuracy : 0.5889
##                  95% CI : (0.5276, 0.6482)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : 0.5261
##
##                   Kappa : 0
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.5889
##          Neg Pred Value :    NaN
##              Prevalence : 0.5889
##          Detection Rate : 0.5889
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : CH
##
```

```r
# model tuning
cost.values <- c(0.01, 0.1, 1, 3, 5, 7, 10)
tune.out <- tune(svm,
                 Purchase ~ .,
                 data = oj.train,
                 kernel = 'radial',
                 ranges = list(cost = cost.values))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17875
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38250 0.05533986
## 2  0.10 0.19125 0.03064696
## 3  1.00 0.17875 0.02638523
## 4  3.00 0.18625 0.03356689
## 5  5.00 0.18625 0.03197764
## 6  7.00 0.18750 0.03435921
## 7 10.00 0.19500 0.03446012
```

```r
# tuned model evaluation
train.error <- predict(tune.out$best.model, oj.train)
test.error <- predict(tune.out$best.model, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
```

```
##         CH 453  81
##         MM  41 225
##
##                Accuracy : 0.8475
##                  95% CI : (0.8207, 0.8717)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6689
##  Mcnemar's Test P-Value : 0.0004142
##
##             Sensitivity : 0.9170
##             Specificity : 0.7353
##          Pos Pred Value : 0.8483
##          Neg Pred Value : 0.8459
##              Prevalence : 0.6175
##          Detection Rate : 0.5663
##    Detection Prevalence : 0.6675
##       Balanced Accuracy : 0.8261
##
##        'Positive' Class : CH
##
```

```r
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 143  26
##         MM  16  85
##
##                Accuracy : 0.8444
##                  95% CI : (0.7956, 0.8855)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6743
##  Mcnemar's Test P-Value : 0.1649
##
##             Sensitivity : 0.8994
##             Specificity : 0.7658
##          Pos Pred Value : 0.8462
##          Neg Pred Value : 0.8416
##              Prevalence : 0.5889
##          Detection Rate : 0.5296
##    Detection Prevalence : 0.6259
##       Balanced Accuracy : 0.8326
##
##        'Positive' Class : CH
##
```

- **G**.

```r
svm.poly <- svm(Purchase ~ .,
                data = oj.train,
                kernel = 'polynomial',
                degree = 2,
                cost = 0.01)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj.train, kernel = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  617
##
##  ( 311 306 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# prediction
train.error <- predict(svm.poly, oj.train)
test.error <- predict(svm.poly, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 493 291
##         MM   1  15
##
##                Accuracy : 0.635
##                  95% CI : (0.6006, 0.6684)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : 0.163
##
##                   Kappa : 0.0573
##  Mcnemar's Test P-Value : <2e-16
##
```

```
##             Sensitivity : 0.99798
##             Specificity : 0.04902
##          Pos Pred Value : 0.62883
##          Neg Pred Value : 0.93750
##              Prevalence : 0.61750
##          Detection Rate : 0.61625
##    Detection Prevalence : 0.98000
##       Balanced Accuracy : 0.52350
##
##        'Positive' Class : CH
##
```

```
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 158 105
##         MM   1   6
##
##                Accuracy : 0.6074
##                  95% CI : (0.5464, 0.666)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : 0.2898
##
##                   Kappa : 0.0556
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99371
##             Specificity : 0.05405
##          Pos Pred Value : 0.60076
##          Neg Pred Value : 0.85714
##              Prevalence : 0.58889
##          Detection Rate : 0.58519
##    Detection Prevalence : 0.97407
##       Balanced Accuracy : 0.52388
##
##        'Positive' Class : CH
##
```

```
# model tuning
cost.values <- c(0.01, 0.1, 1, 3, 5, 7, 10)
tune.out <- tune(svm,
                 Purchase ~ .,
                 data = oj.train,
                 kernel = 'polynomial',
                 degree = 2,
                 ranges = list(cost = cost.values))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.19375
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38250 0.06566963
## 2  0.10 0.32000 0.04533824
## 3  1.00 0.20250 0.03717451
## 4  3.00 0.20250 0.04362084
## 5  5.00 0.20000 0.03333333
## 6  7.00 0.19875 0.03356689
## 7 10.00 0.19375 0.03448530
```

```r
# tuned model evaluation
train.error <- predict(tune.out$best.model, oj.train)
test.error <- predict(tune.out$best.model, oj.test)

# training error
confusionMatrix(train.error, oj.train$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 453  88
##         MM  41 218
##
##                Accuracy : 0.8388
##                  95% CI : (0.8114, 0.8636)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6484
##  Mcnemar's Test P-Value : 5.12e-05
##
##             Sensitivity : 0.9170
##             Specificity : 0.7124
##          Pos Pred Value : 0.8373
##          Neg Pred Value : 0.8417
##              Prevalence : 0.6175
##          Detection Rate : 0.5663
##    Detection Prevalence : 0.6763
##       Balanced Accuracy : 0.8147
##
##        'Positive' Class : CH
##
```

```r
# testing error
confusionMatrix(test.error, oj.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 146  26
##         MM  13  85
##
##                Accuracy : 0.8556
##                  95% CI : (0.8079, 0.8952)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6963
##  Mcnemar's Test P-Value : 0.05466
##
##             Sensitivity : 0.9182
##             Specificity : 0.7658
##          Pos Pred Value : 0.8488
##          Neg Pred Value : 0.8673
##              Prevalence : 0.5889
##          Detection Rate : 0.5407
##    Detection Prevalence : 0.6370
##       Balanced Accuracy : 0.8420
##
##        'Positive' Class : CH
##
```

- **H**. With 85.56% accuracy on the test set and only a slight dip to 83.88% accuracy on the training set, the polynomial model of degree 2 and a cost of 10 seems to be the most appropriate model for this data.