ISL Chapter 9 Exercises

Jonathan Bryan

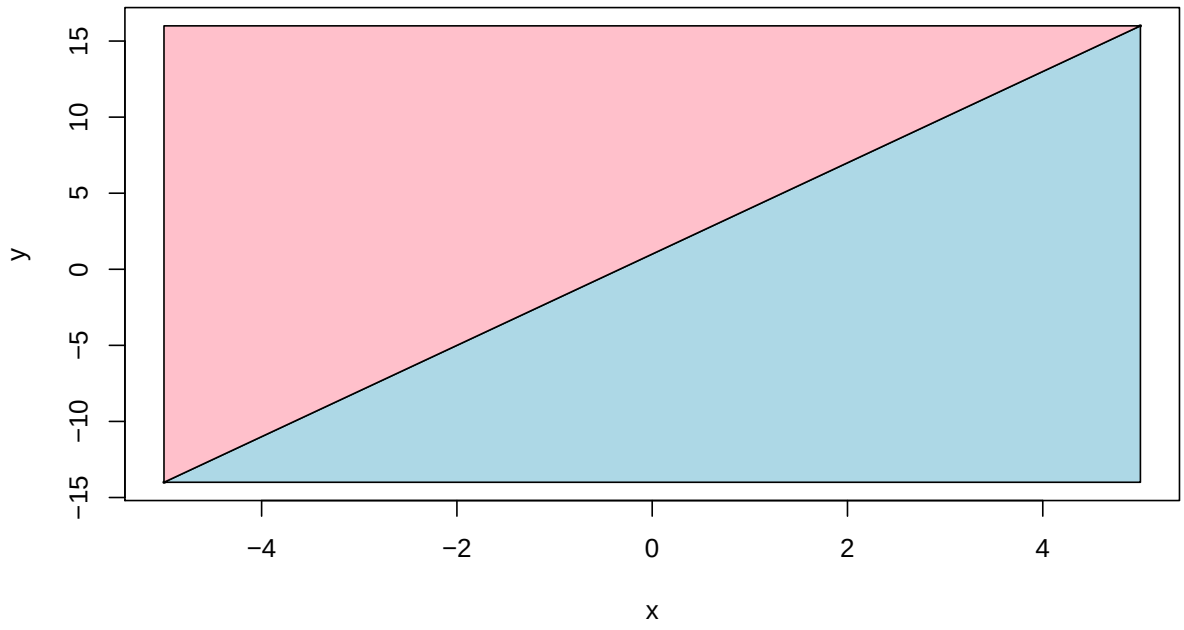
August 23, 2018

1. This problem involves hyperplanes in two dimensions.

(a) Sketch the hyperplane $1+3X_1-X_2=0$. Indicate the set of points for which $1+3X_1-X_2>0$, as well as the set of points for which $1+3X_1-X_2<0$.

The plot below give the line $X_2 = 3X_1 + 1$. $1+3X_1-X_2<0$ for all points above the line shaded pink and $1+3X_1-X_2>0$ all points below the line shaded blue.

```
x = seq(-5,5,by=1)
y = 1 + 3*x
plot(x,y, type = "l", lwd = 2)
polygon(x = c(-5,5,5), y = c(-14, 16, -14),col='lightblue')
polygon(x = c(-5,-5,5), y = c(-14, 16, 16),col='pink')
```

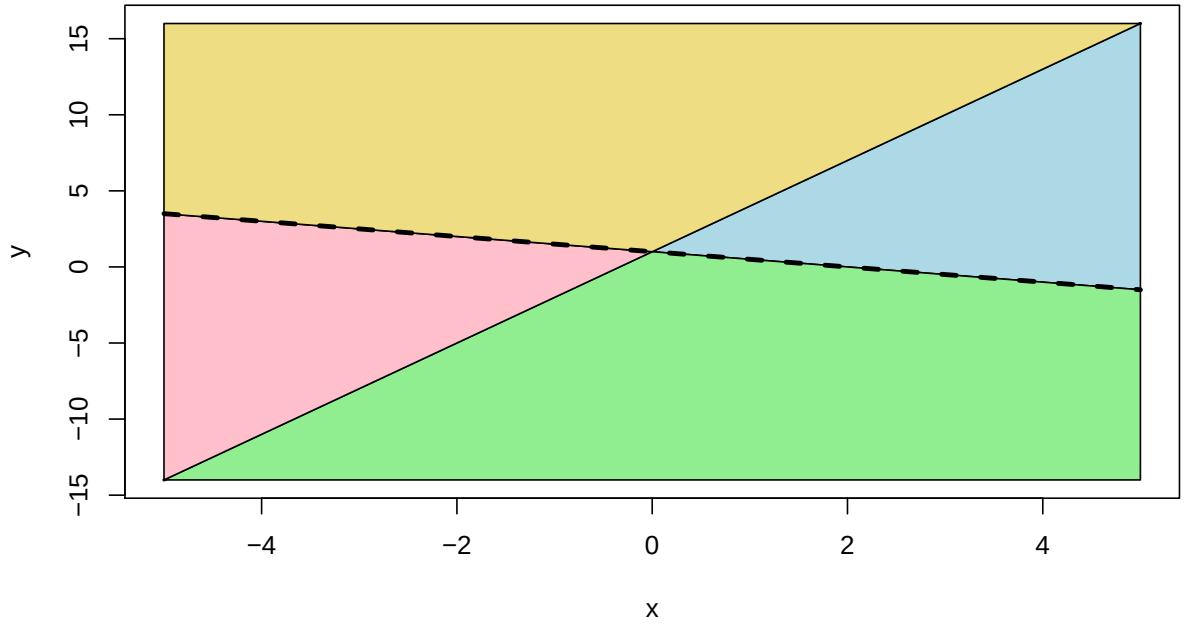


(b) On the same plot, sketch the hyperplane -2 + X_1 + 2X_2 = 0. Indicate the set of points for which -2 + X_1 + 2X_2 > 0, as well as the set of points for which -2 + X_1 + 2X_2 < 0.

The plot below adds the line $X_2 = -0.5X_1 + 1$. The golden region has points above both lines where $-2+X_1+2X_2>0$ and $1+3X_1-X_2<0$, while the green region has points below both lines where $-2+X_1+2X_2<0$ and $1+3X_1-X_2>0$. The pink region has points above line (a) and below line (b)

where each plane is less than zero while the blue region has points above line (b) and below line (a) where each plan is greater than zero.

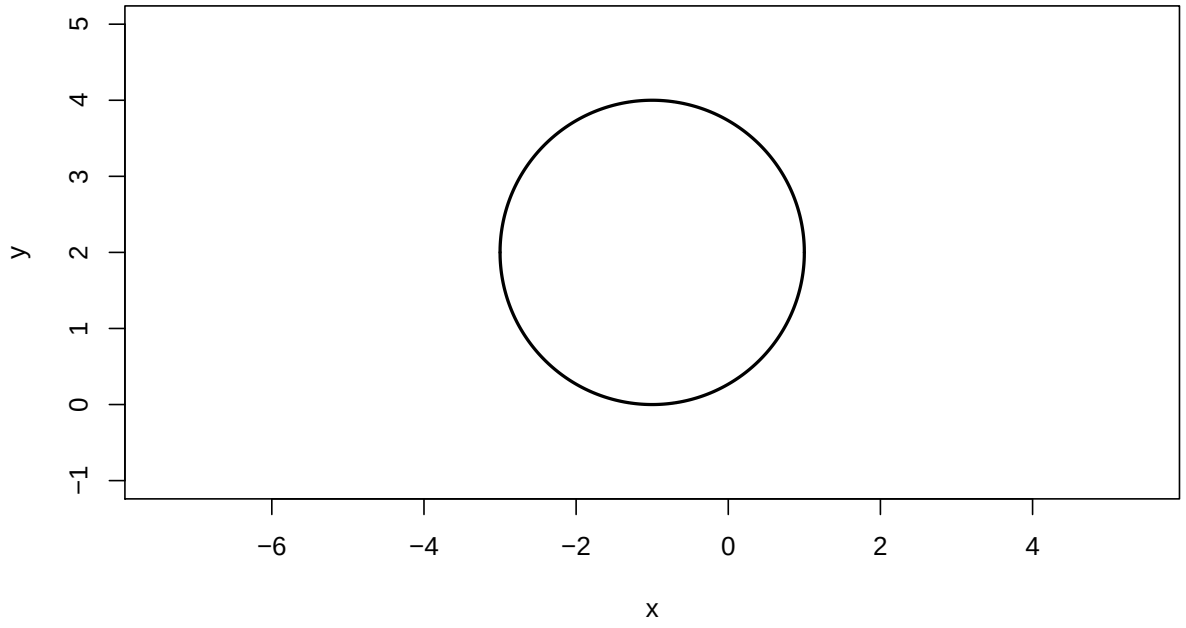
```
x = seq(-5,5,by=1)
y = 1 + 3*x
y.2 = 1 - 0.5*x
plot(x,y, type = "l", lwd = 2)
polygon(x = c(0,5,5), y = c(1, -1.5, 16),col='lightblue')
polygon(x = c(-5,0,5,5), y = c(-14, 1, -1.5,-14),col='lightgreen')
polygon(x = c(-5,-5,0), y = c(-14, 3.5, 1),col='pink')
polygon(x = c(-5,-5,5,0), y = c(3.5, 16, 16,1),col='lightgoldenrod')
lines(x,y.2, lwd = 3, lty = 2)
```



2. We have seen that in p = 2 dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

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

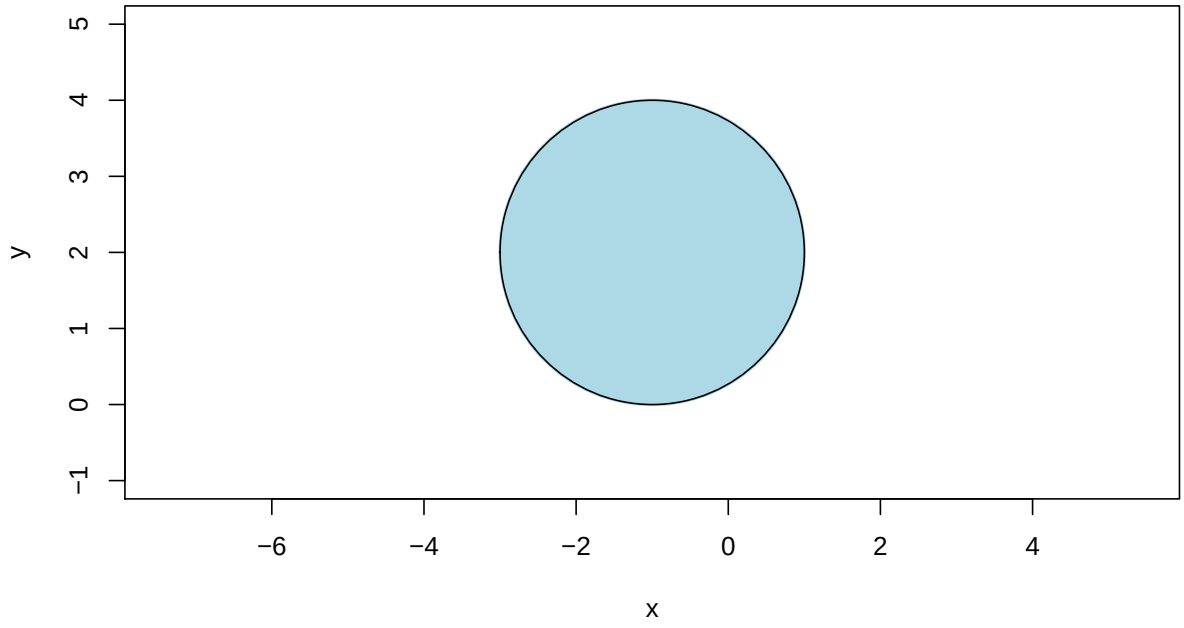
```
x = seq(-3,1,by=0.001)
y = sqrt(4-(1+x)^2) + 2
y.2 = -1*sqrt(4-(1+x)^2) + 2
plot(x,y, type = "l", lwd = 2,
     xlim= c(-4,2),
     ylim = c(-1,5),
     asp=1)
lines(x,y.2, lwd=2)
```



(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 $(1+X_1)^2+(2-X_2)^2\leq 4$.

In the figure below, the white space is where $(1+X_1)^2+(2-X_2)^2>4$ and the lightblue space is where $(1+X_1)^2+(2-X_2)^2\leq 4$.

```
x = seq(-3,1,by=0.001)
y = sqrt(4-(1+x)^2) + 2
y.2 = -1*sqrt(4-(1+x)^2) + 2
plot(x,y, type = "l", lwd = 2,
     col = "lightblue",
     xlim= c(-4,2),
     ylim = c(-1,5),
     asp=1)
lines(x,y.2, lwd=2, col = "lightblue")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE, bg = "lightblue")
```



(c) Suppose that a classifier assigns an observation to the blue class if (1 + X1)^2 + (2 - X2)^2 > 4, and to the red class otherwise. To what class is the observation (0, 0) classified? (-1, 1)? (2, 2)? (3, 8)?

(0,0), (2, 2), and (3, 8) are classified as blue. (-1, 1) is classified as red.

(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.

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

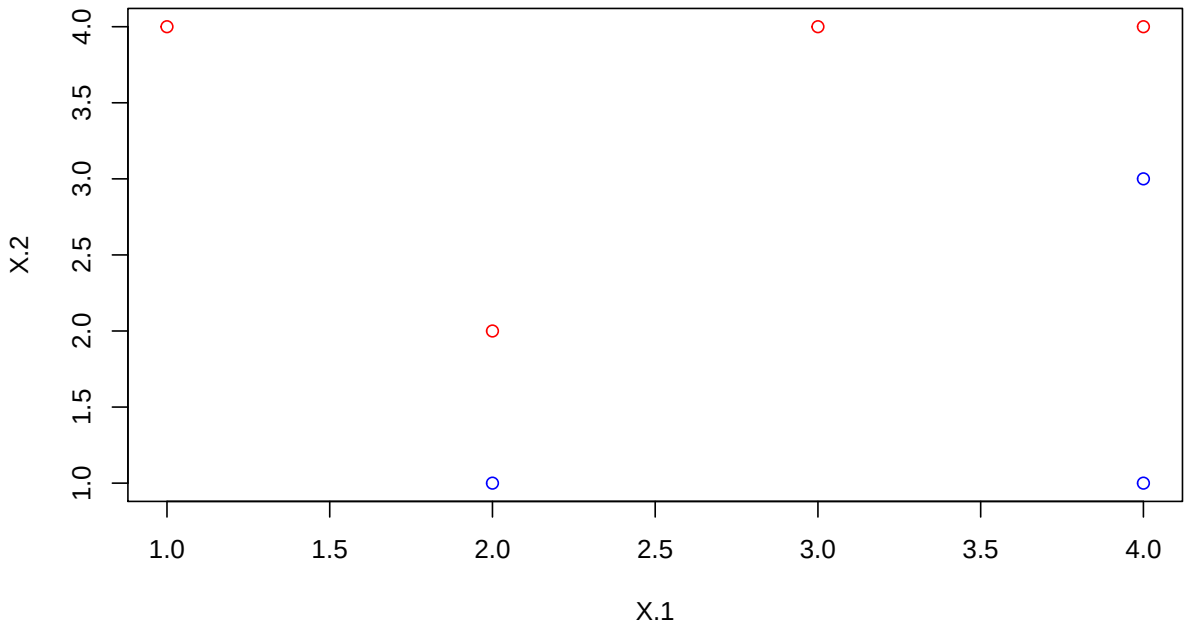
As shown above, when we expand the quadratic terms we get and additive linear model given X_1, X_1^2, X_2, and X_2^2.

3. Here we explore the maximal margin classifier on a toy data set.

(a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label. Sketch the observations.

```
X = data.frame(X.1 = c(3,2,4,1,2,4,4),
               X.2 = c(4,2,4,4,1,3,1))
```

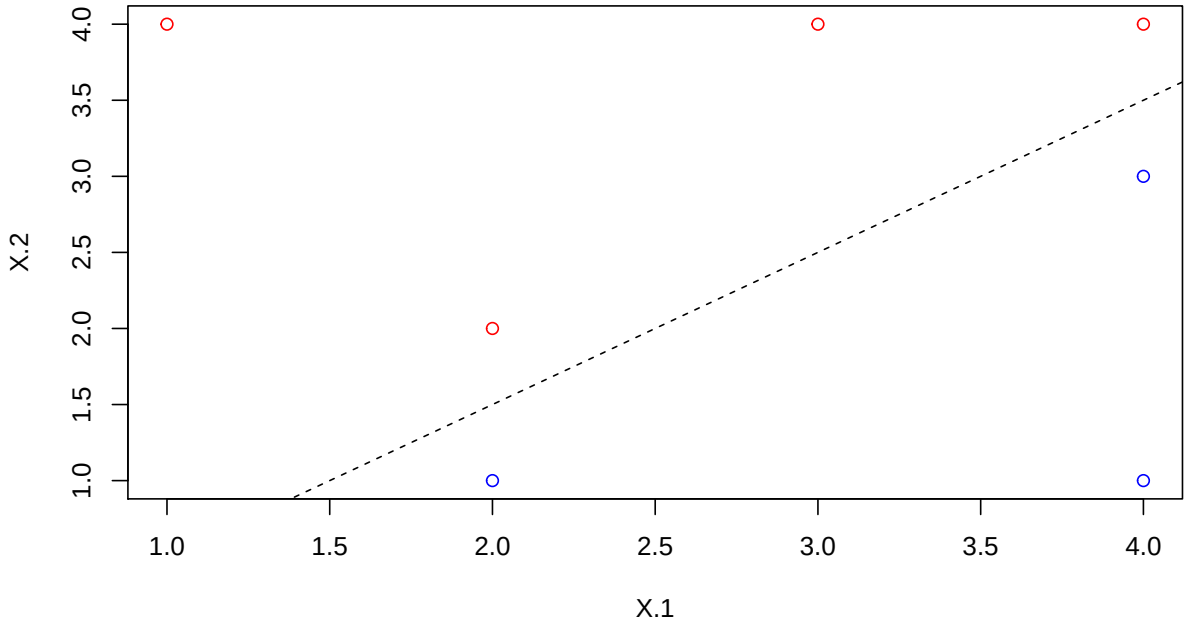
```
Y = c("red",
      "red",
      "red",
      "red",
      "blue",
      "blue",
      "blue")
plot(X, col = Y)
```



(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

The equation for the sketched optimal separating hyperplane is $X_1 - X_2 - 0.5 = 0$

```
x = seq(0,5)
x.hyp = x - 0.5
plot(X, col = Y)
lines(x, x.hyp, lty = 2)
```



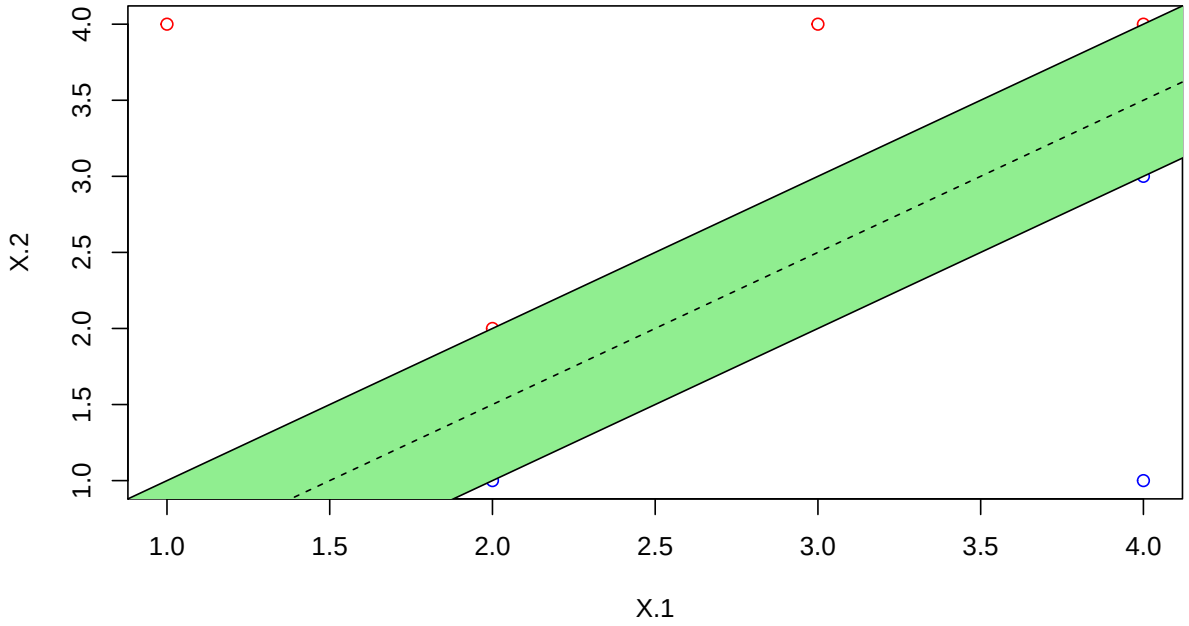
(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

Classify to Red if $X_1 - X_2 - 0.5 < 0$ and classify to Blue if $X_1 - X_2 - 0.5 \geq 0$. $\beta_0 = -0.5$, $\beta_1 = 1$, and $\beta_2 = -1$

(d) On your sketch, indicate the margin for the maximal margin hyperplane.

The maximal margin hyperplane is found in green below.

```
plot(X, col = Y)
polygon(c(0,0,5,5), c(0,-1,4,5), col = "lightgreen")
lines(x, x.hyp, lty = 2)
```



(e) Indicate the support vectors for the maximal margin classifier.

The support vectors are (2,2), (2,1), (4,3), and (4,4).

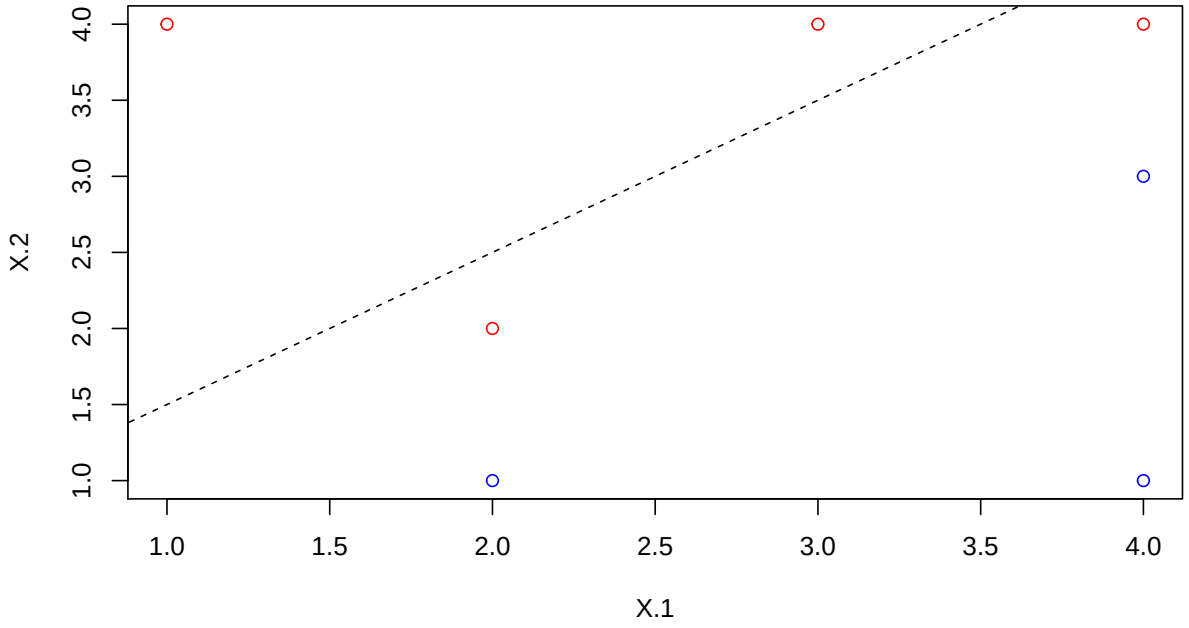
(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

Any slight movement of the (4,1) vector would not make it a support vector and therefore not move the maximal margin hyperplane as it is not close to the margin.

(g) Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

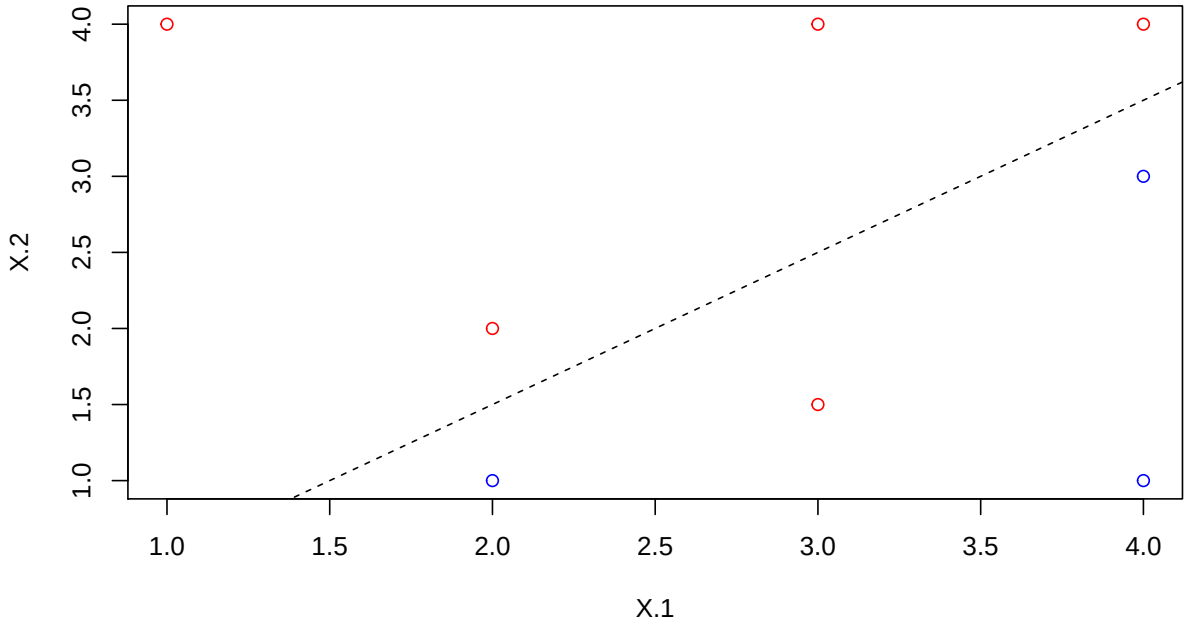
A non-optimal separating hyperplane is shown below. The equation is $X_1 - X_2 + 0.5 = 0$.

```
x = seq(0,5)
x.hyp = x + 0.5
plot(X, col = Y)
lines(x, x.hyp, lty = 2)
```



(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

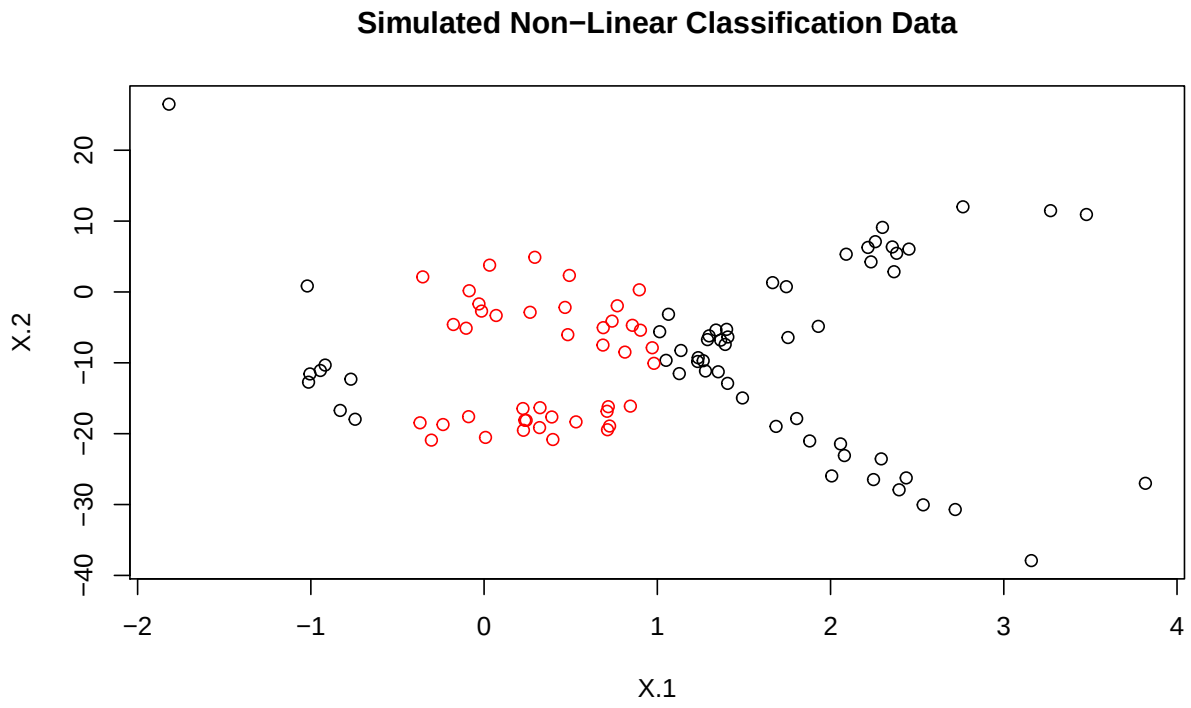
```
x = seq(0,5)
x.hyp = x - 0.5
X = rbind(X,c(3,1.5))
Y = c(Y, "red")
plot(X, col = Y)
lines(x, x.hyp, lty = 2)
```



4. Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

Given a non-linear data set ($p = 2,\ n = 100$) between two classes we find that the radial svm classifier had the highest training and test set error rate compared to a linear svm classifier and polynomial svm classier of degree 3. Each svm was 10-fold cross-validated against a range of cost values ($c = \{10^{-3}, 10^{-2}, ..., 10^{4}\}$). As suggested by the receiver operating curves, the radial svm performs the best.

```
set.seed(319)
X.1 = rnorm(100, 1, 1)
X.2 = rep(NA,100)
X.2[1:50] = X.1[1:50] - 10*X.1[1:50]^2 + 2*X.1[1:50]^3 + rnorm(50,0,3)
X.2[51:100] = - X.1[51:100] + 10*X.1[51:100]^2 - 2*X.1[51:100]^3 - 20 + rnorm(50,0,3)
y = rep("black", 100)
y.red.index = which(X.1 >= -0.4 & X.1 <= 1 )
y[y.red.index] = "red"
plot(X.1,X.2, col = y, main = "Simulated Non-Linear Classification Data")
```




```r
library(e1071)
set.seed(1)

data.sim = data.frame(y,X.1,X.2)

#create training and test sets
set.seed(23)
train = sample(1:100, size = 70 , replace = FALSE)
data.train = data.sim[train,]
data.test = data.sim[-train,]
#linear svm with 10-fold cv
svm.linear = tune(svm, as.factor(y) ~ .,
                  data = data.train,
                  kernel = "linear",
                  ranges = list(cost = 10^(-3:4))
                )
#poly svm with 10-fold cv
svm.poly = tune(svm, as.factor(y) ~ .,
                  data = data.train,
                  kernel = "polynomial",
                  ranges = list(cost = 10^(-3:4))
                )
#radial svm with 10-fold cv
svm.radial = tune(svm, as.factor(y) ~ .,
                  data = data.train,
                  kernel = "radial",
                  ranges = list(cost = 10^(-3:4),
                                gamma = c(0.25, 0.5, 1, 2)
                                )
```

```
              )

#cv summaries
summary(svm.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1285714
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4000000  0.1621846
## 2 1e-02 0.4000000  0.1621846
## 3 1e-01 0.1571429  0.1420613
## 4 1e+00 0.1714286  0.1756104
## 5 1e+01 0.1285714  0.1572150
## 6 1e+02 0.1285714  0.1572150
## 7 1e+03 0.1285714  0.1572150
## 8 1e+04 0.1285714  0.1572150
```

```
summary(svm.poly)
```

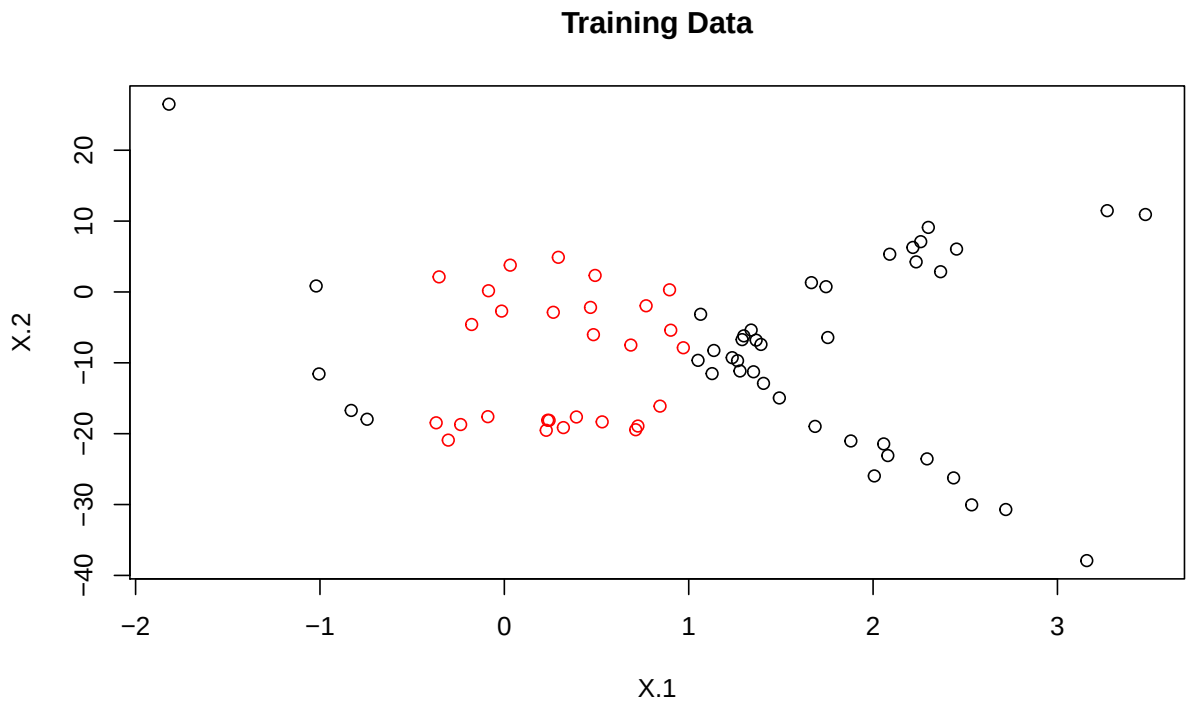
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2571429
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4000000  0.2841226
## 2 1e-02 0.4142857  0.2647037
## 3 1e-01 0.4714286  0.2432769
## 4 1e+00 0.4857143  0.1380131
## 5 1e+01 0.2571429  0.2213133
## 6 1e+02 0.2571429  0.2213133
## 7 1e+03 0.2571429  0.2213133
## 8 1e+04 0.2571429  0.2213133
```

```
summary(svm.radial)
```

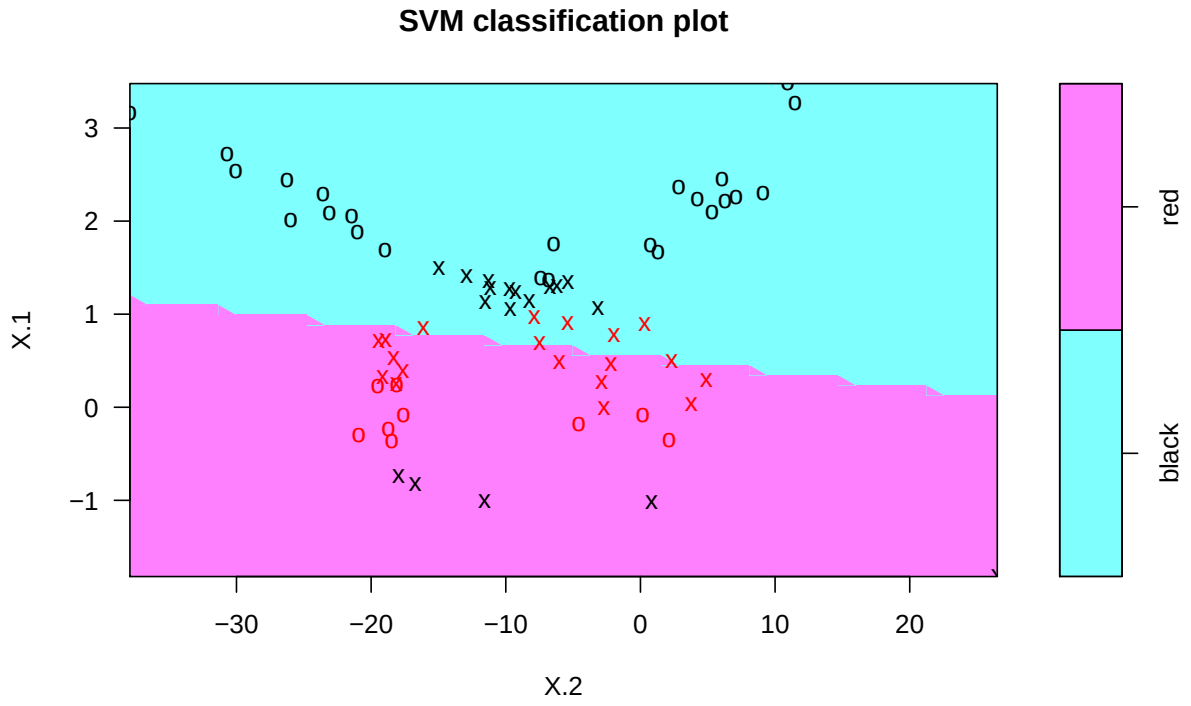
```
##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.01428571
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03  0.25 0.40000000 0.13127665
## 2  1e-02  0.25 0.40000000 0.13127665
## 3  1e-01  0.25 0.38571429 0.17880937
## 4  1e+00  0.25 0.10000000 0.11761037
## 5  1e+01  0.25 0.02857143 0.06023386
## 6  1e+02  0.25 0.04285714 0.06900656
## 7  1e+03  0.25 0.02857143 0.06023386
## 8  1e+04  0.25 0.02857143 0.06023386
## 9  1e-03  0.50 0.40000000 0.13127665
## 10 1e-02  0.50 0.40000000 0.13127665
## 11 1e-01  0.50 0.34285714 0.18070158
## 12 1e+00  0.50 0.05714286 0.07377111
## 13 1e+01  0.50 0.01428571 0.04517540
## 14 1e+02  0.50 0.02857143 0.06023386
## 15 1e+03  0.50 0.01428571 0.04517540
## 16 1e+04  0.50 0.01428571 0.04517540
## 17 1e-03  1.00 0.40000000 0.13127665
## 18 1e-02  1.00 0.40000000 0.13127665
## 19 1e-01  1.00 0.28571429 0.20203051
## 20 1e+00  1.00 0.05714286 0.07377111
## 21 1e+01  1.00 0.04285714 0.06900656
## 22 1e+02  1.00 0.02857143 0.06023386
## 23 1e+03  1.00 0.02857143 0.06023386
## 24 1e+04  1.00 0.02857143 0.06023386
## 25 1e-03  2.00 0.40000000 0.13127665
## 26 1e-02  2.00 0.40000000 0.13127665
## 27 1e-01  2.00 0.32857143 0.17880937
## 28 1e+00  2.00 0.05714286 0.07377111
## 29 1e+01  2.00 0.05714286 0.07377111
## 30 1e+02  2.00 0.08571429 0.07377111
## 31 1e+03  2.00 0.08571429 0.07377111
## 32 1e+04  2.00 0.08571429 0.07377111
```

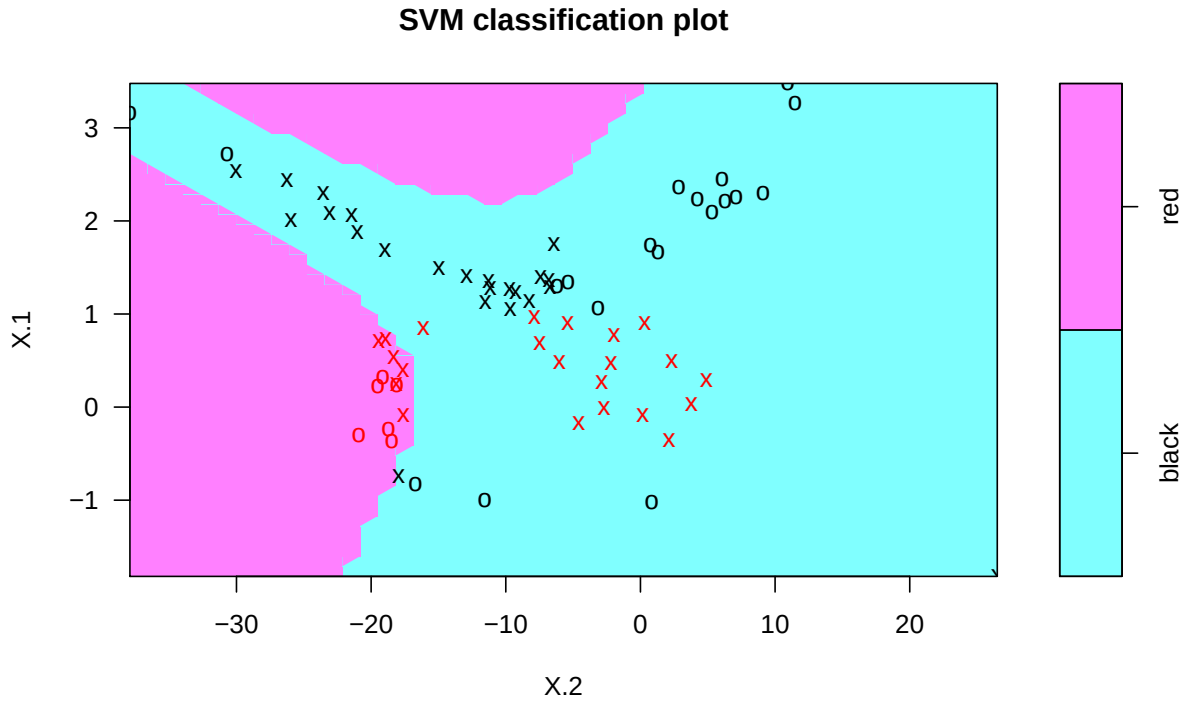
```r
#svm plots
plot(data.train[,c(2,3)], col = data.train[,c(1)] , main = "Training Data")
```



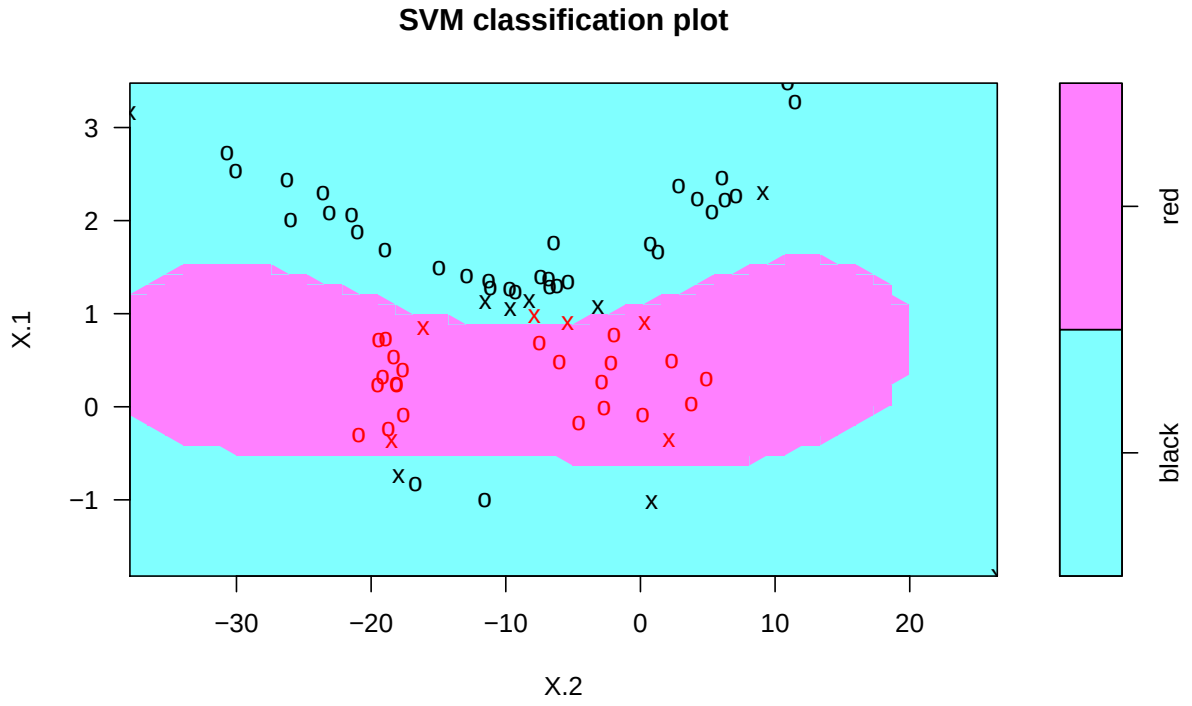

```
plot(svm.linear$best.model, data= data.train, main = "Linear SVM")
```



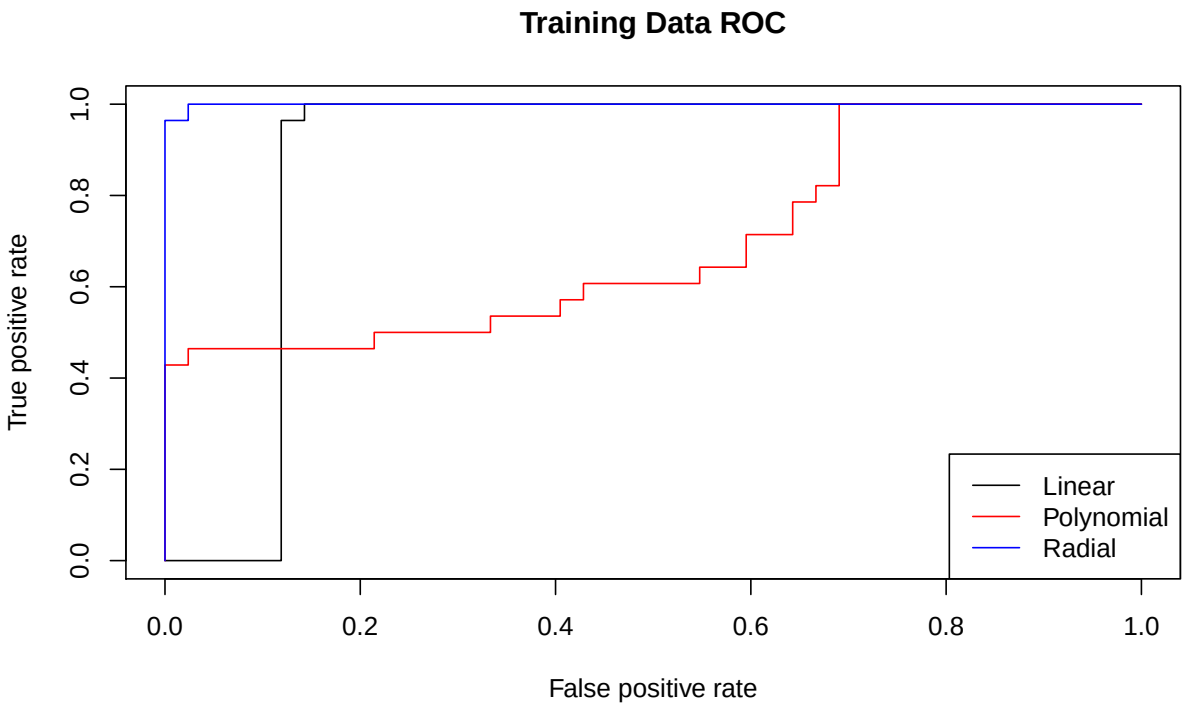
```
plot(svm.poly$best.model, data= data.train, main = "Polynomial SVM")
```



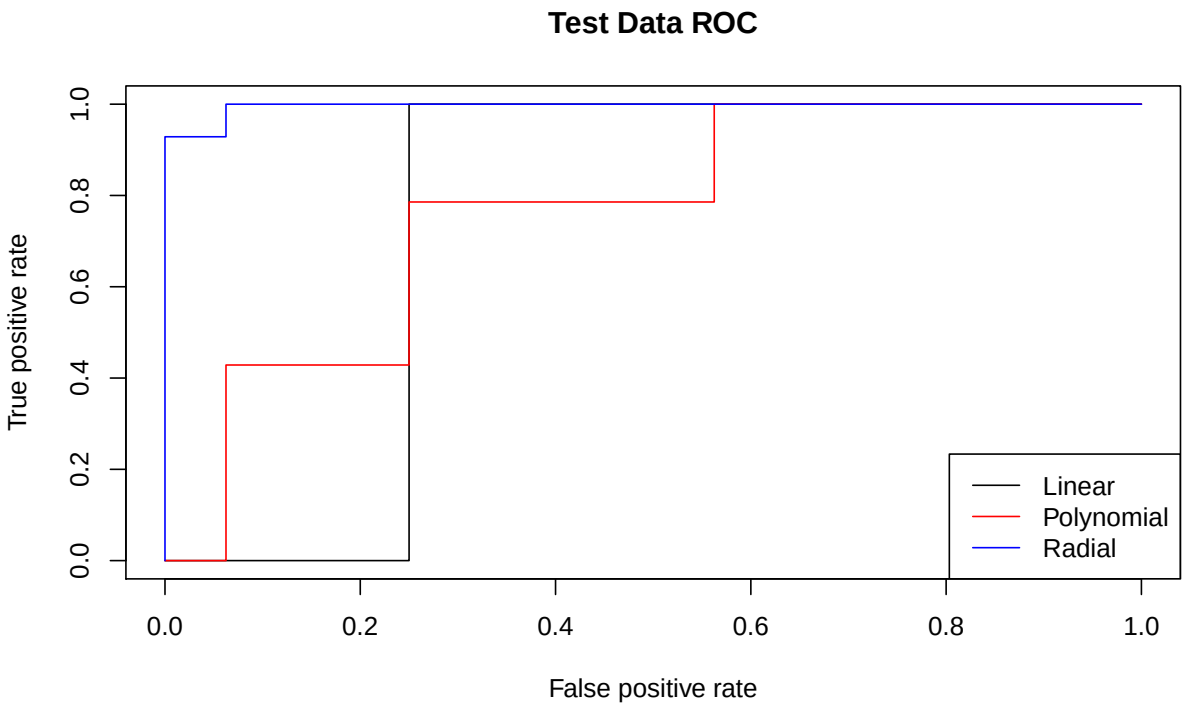

```
plot(svm.radial$best.model, data= data.train, main = "Radial SVM")
```



```r
#ROC curves for svm models
library (ROCR)

rocplot = function (pred, truth, ...){
  predob = prediction(pred, truth )
  perf = performance(predob, "tpr", "fpr")
  plot(perf ,...)
}

fitted.linear = -1*attributes(predict(svm.linear$best.model,data.train, decision.values = TRUE))$decisi

fitted.poly = -1*attributes(predict(svm.poly$best.model,data.train, decision.values = TRUE))$decision.v

fitted.radial = -1*attributes(predict(svm.radial$best.model,data.train, decision.values = TRUE))$decisi

rocplot(fitted.linear, data.train[,1],
        main= "Training Data ROC")

rocplot(fitted.poly, data.train[,1],
        add = TRUE,
        col = "red")

rocplot(fitted.radial, data.train[,1],
        add = TRUE,
        col = "blue")
legend("bottomright", legend = c("Linear",
                                 "Polynomial",
                                 "Radial"),
       col = c("black",
               "red",
               "blue"),
       lty = 1)
```




```
fitted.linear = -1*attributes(predict(svm.linear$best.model,data.test, decision.values = TRUE))$decisio

fitted.poly = -1*attributes(predict(svm.poly$best.model,data.test, decision.values = TRUE))$decision.va

fitted.radial = -1*attributes(predict(svm.radial$best.model,data.test, decision.values = TRUE))$decisio

rocplot(fitted.linear, data.test[,1],
        main= "Test Data ROC")

rocplot(fitted.poly, data.test[,1],
        add = TRUE,
        col = "red")

rocplot(fitted.radial, data.test[,1],
        add = TRUE,
        col = "blue")
legend("bottomright", legend = c("Linear",
                                 "Polynomial",
                                 "Radial"),
       col = c("black",
               "red",
               "blue"),
       lty = 1)
```



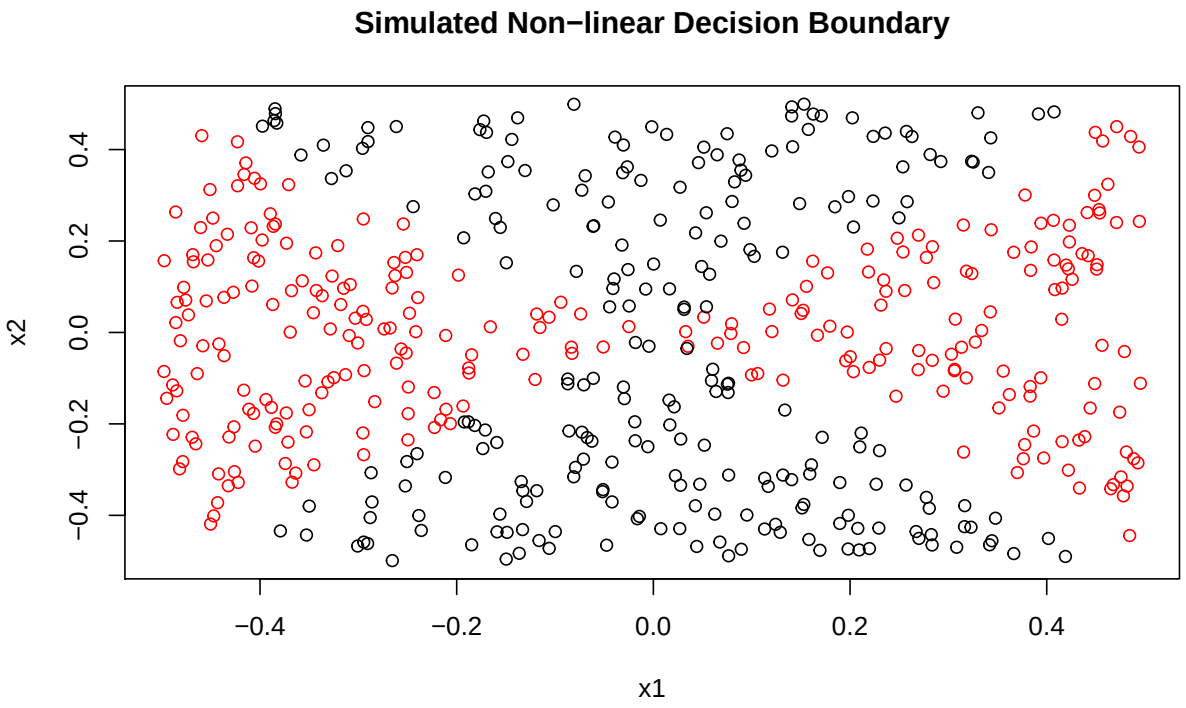
5. We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(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:

```
#generate simulated dataset
set.seed(340)
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 yaxis.

```
plot(x1,x2, col = y+1, main = "Simulated Non-linear Decision Boundary")
```

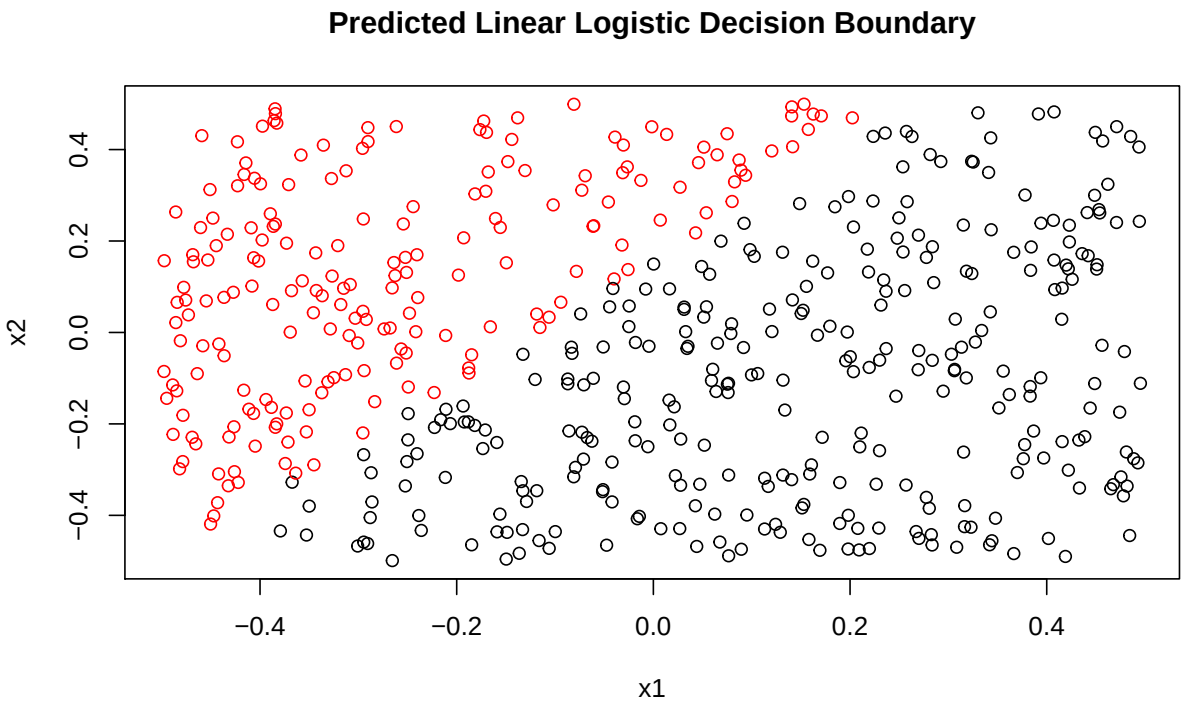


(c) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

```
log.reg = glm(y ~ x1 + x2,
              family = binomial)
```

(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.

```
library(pROC)
preds = predict(log.reg,
                newdata = data.frame(x1,x2),
                type = "response")

roc = roc(y, preds)
thres = coords(roc, "best", ret = "threshold")
y.pred = y
y.pred[preds > thres] = 1
y.pred[preds < thres] = 0
plot(x1,x2, col = y.pred+1, main = "Predicted Linear Logistic Decision Boundary")
```



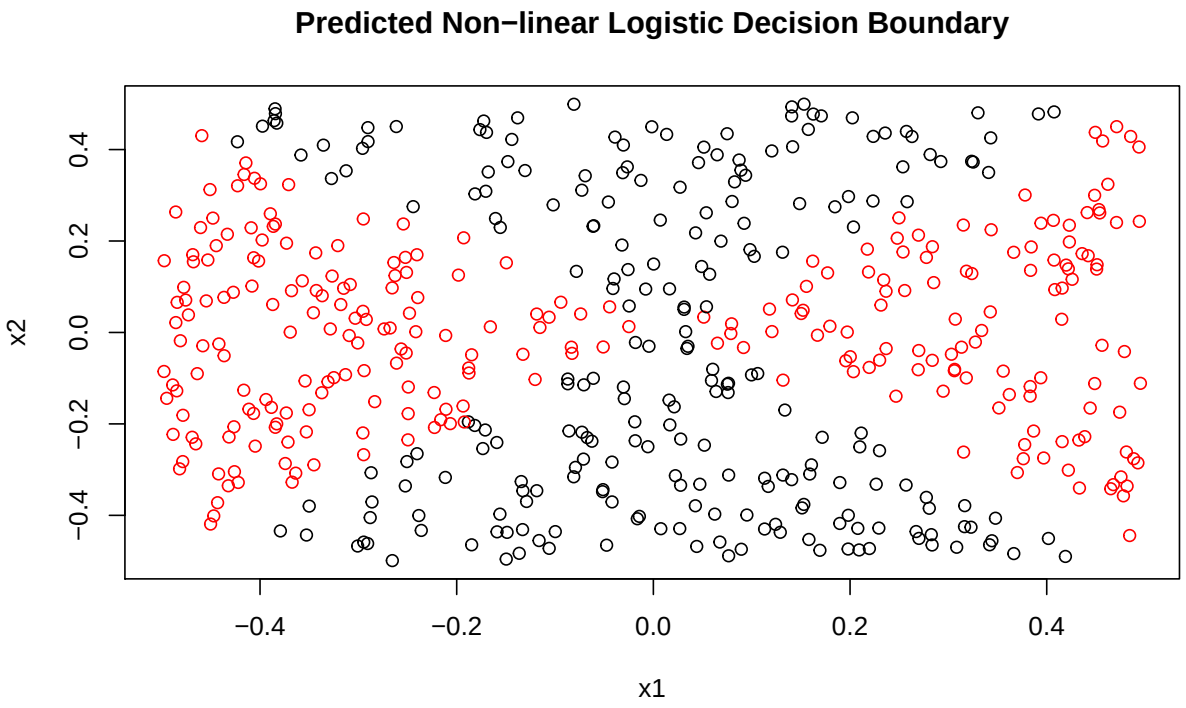
(e) Now fit a logistic regression model to the data using non-linear functions of X_1 and X_2 as predictors (e.g. X_1^2, $X_1 \times X_2$, $log(X_2)$, and so forth).

```
log.reg.nonlin = glm(y ~ x1 + x2 + exp(x1) + exp(x2),
            family = binomial)
```

(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.

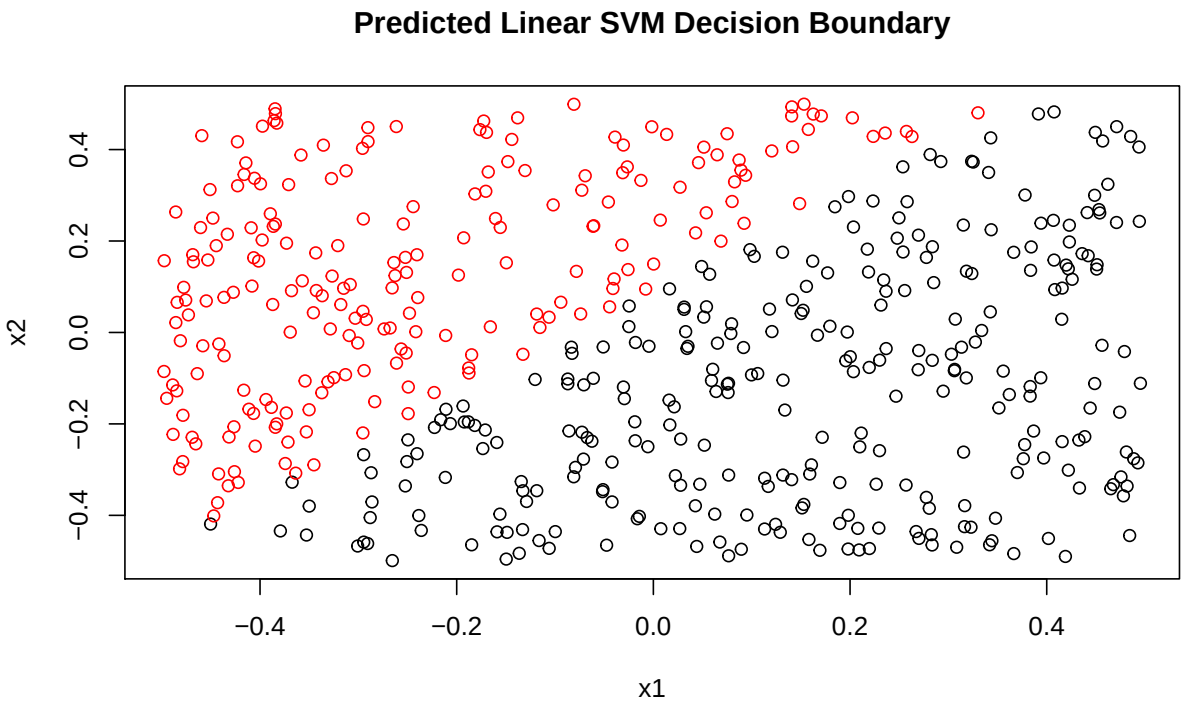
By adding exponential transformations of the predictors ($e^{X_1} + e^{X_2}$) we are able to achieve a highly nonlinear decision boundary and much better accuracy.

```
preds = predict(log.reg.nonlin,
                newdata = data.frame(x1,x2),
                type = "response")

roc = roc(y, preds)
thres = coords(roc, "best", ret = "threshold")
y.pred = y
y.pred[preds > thres] = 1
y.pred[preds < thres] = 0
plot(x1,x2, col = y.pred+1, main = "Predicted Non-linear Logistic Decision Boundary")
```



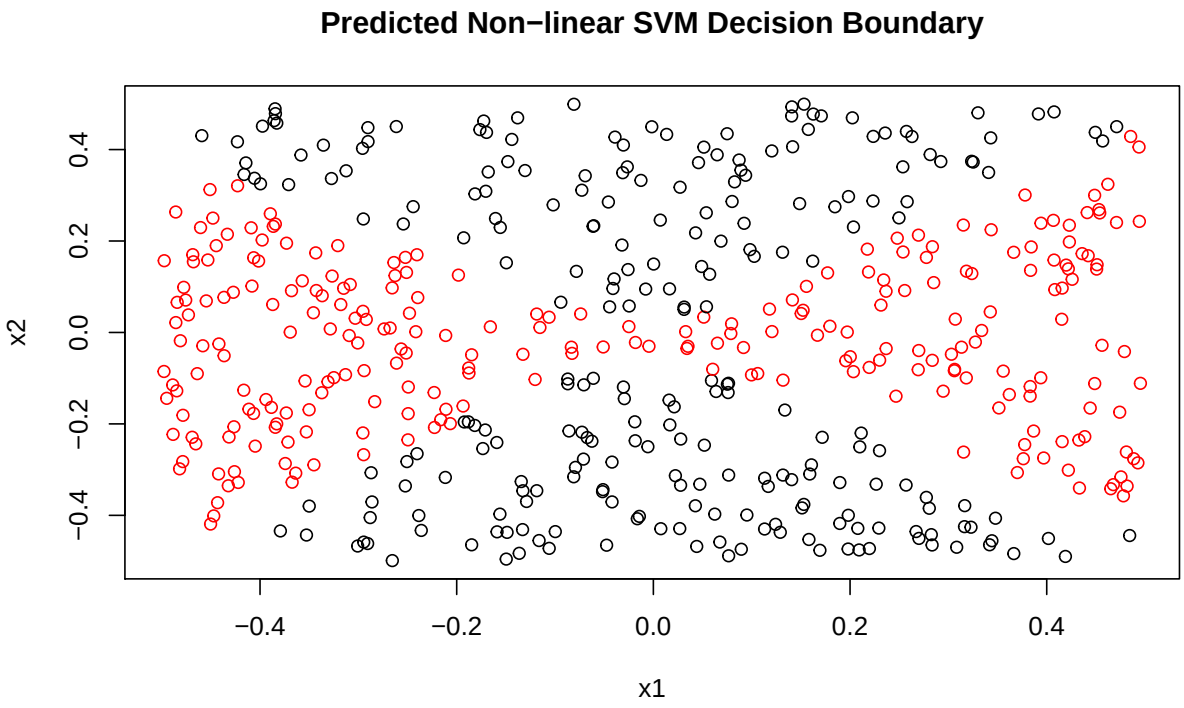
(g) Fit a support vector classifier to the data with X_1 and X_2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
data.svm = data.frame(y,x1,x2)
svm.fit = svm(y ~ .,
              data = data.svm,
              kernel = "linear"
              )
preds = predict(svm.fit,
                newdata = data.svm,
                type = "response")
roc = roc(y, preds)
thres = coords(roc, "best", ret = "threshold")
y.pred = y
y.pred[preds > thres] = 1
y.pred[preds < thres] = 0
plot(x1,x2, col = y.pred+1, main = "Predicted Linear SVM Decision Boundary")
```



(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.

```
set.seed(1)
svm.fit.nonlin = tune(svm, y ~ .,
              data = data.svm,
              ranges = list(cost = c(5,10,15),
                            degree = c(2,3,4)),
              kernel = "polynomial"
              )
preds = predict(svm.fit.nonlin$best.model,
                newdata = data.svm,
                type = "response")
roc = roc(y, preds)
thres = coords(roc, "best", ret = "threshold")
y.pred = y
y.pred[preds > thres] = 1
y.pred[preds < thres] = 0
plot(x1,x2, col = y.pred+1, main = "Predicted Non-linear SVM Decision Boundary")
```



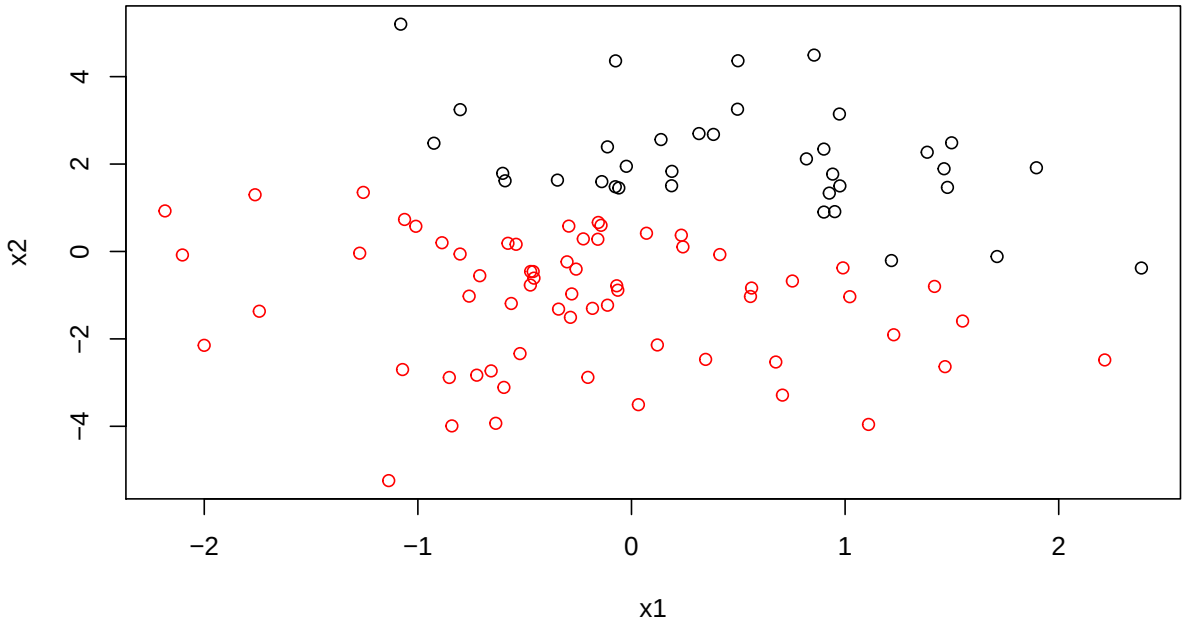
(i) Comment on your results.

We observe that a non-linear decision boundary can be generated using both logistic regression and support vector machines with non-linear transformations of the predictors and non-linear kernels, respectively.

6. At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations. You will now investigate this claim.

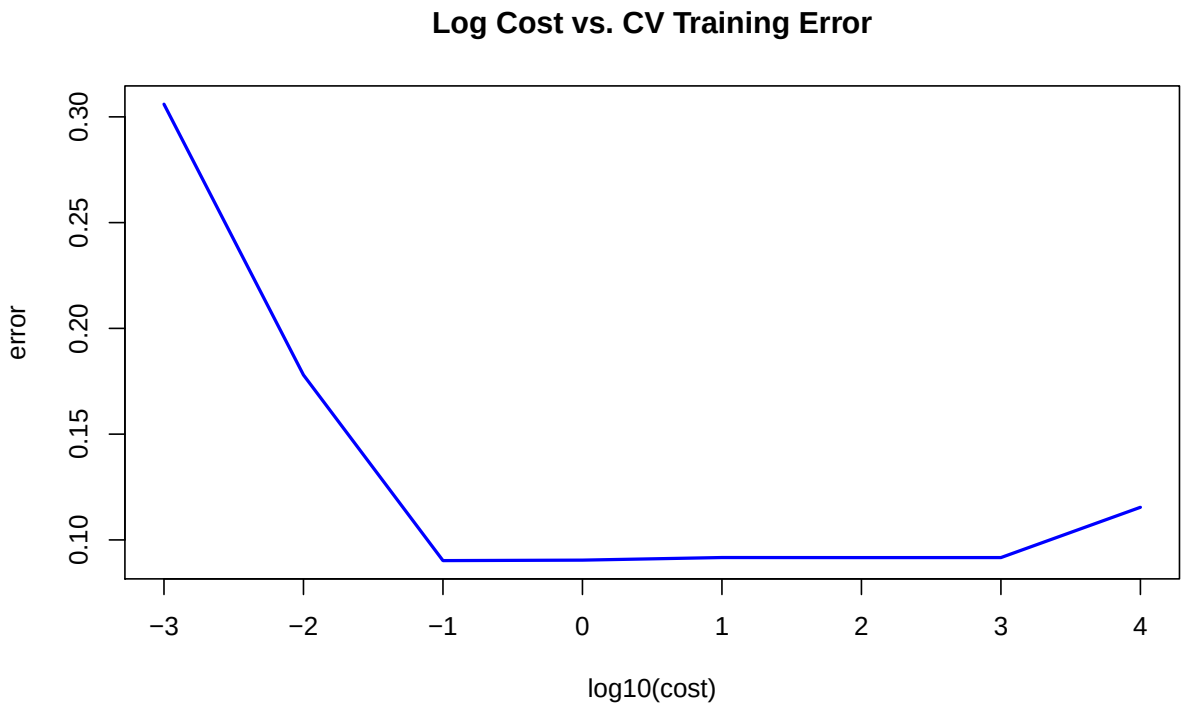
(a) Generate two-class data with p = 2 in such a way that the classes are just barely linearly separable.

```
#siumulate barely separable classes with p=2
set.seed(5)
x1 = rnorm(100, 0, 1)
x2 = rnorm(100, 0 ,2)
y = rep(1,100)
y[x1 + x2 < 1 ] = 2

#plot simulated data
plot(x1,x2, col = y)
```



(b) Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training errors are misclassified for each value of cost considered, and how does this relate to the cross-validation errors obtained?

```r
#cv linear svm
set.seed(1)
svm.cv = tune(svm,
              y ~.,
              data = data.frame(x1 = x1,
                                x2 = x2,
                                y = y),
              kernel = "linear",
              ranges = list(cost  = 10^(-3:4)))

#plot errors against cost
cost = 10^(-3:4)
error = summary(svm.cv)$performances$error
plot(log10(cost), error, type = "l",
     col = "blue",
     lwd = 2,
     main = "Log Cost vs. CV Training Error")
```

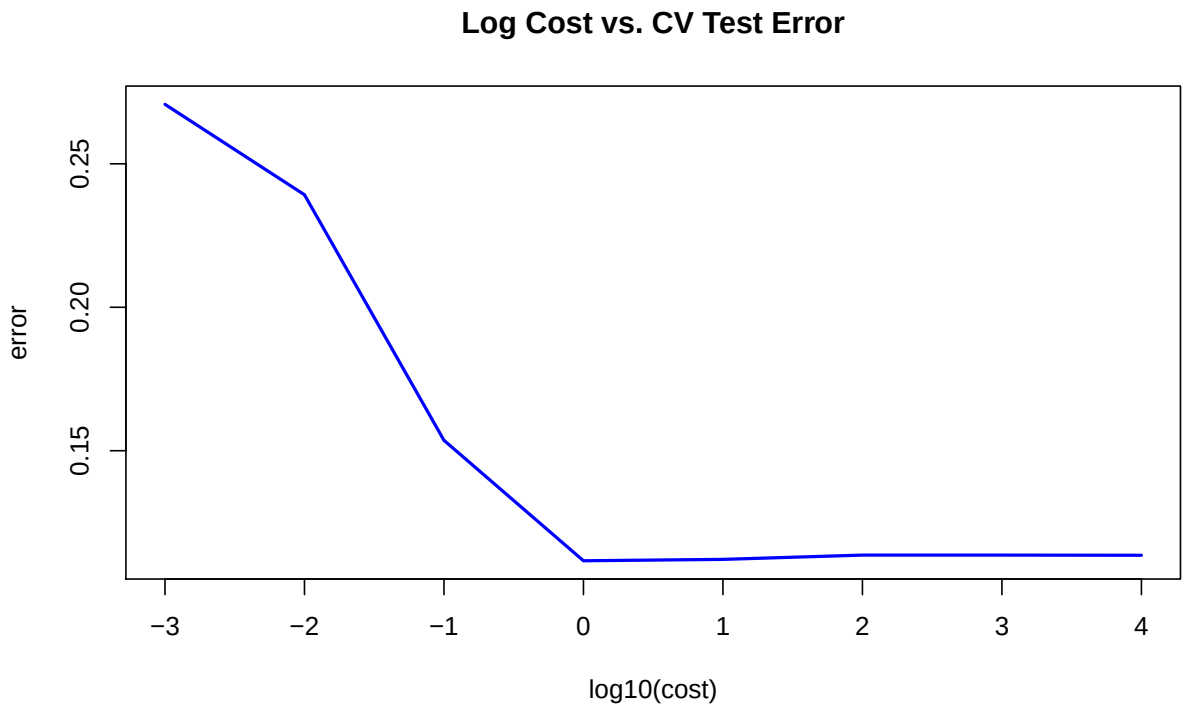


We observe that a very small cost value (i.e. 0.001 and 0.01) increases the cross-validated training error rate while slightly larger (but still relatively small) costs dramatically lower the error rate. The error rate increases as the cost value goes up. A cost of 0.1 produced the lowest error rate.

(c) Generate an appropriate test data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that yield the fewest training errors and the fewest cross-validation errors?

```
#siumulate test set of barely separable classes with p=2
set.seed(871)
x1 = rnorm(50, 0, 1)
x2 = rnorm(50, 0 ,2)
y = rep(1,50)
y[x1 + x2 < 1 ] = 2

#cv linear svm
set.seed(367)
svm.cv = tune(svm,
              y ~.,
              data = data.frame(x1 = x1,
                                x2 = x2,
                                y = y),
              kernel = "linear",
              ranges = list(cost  = 10^(-3:4)))

#plot errors against cost
cost = 10^(-3:4)
error = summary(svm.cv)$performances$error
```

```r
plot(log10(cost), error, type = "l",
     col = "blue",
     lwd = 2,
     main = "Log Cost vs. CV Test Error")
```



We observe a similar pattern as in part (b). The cost value with the lowest error rate is 1.

(d) Discuss your results.

We observe that, given a barely separable parameter space based on classes, a very small cost may produce a large error rate but as cost increases so does the error rate. This suggests that small ($c \approx [0.1,\ 10]$) but not too small cost values are optimal in this setting.

7. In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

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

Auto$mpg.bin = 0
Auto[Auto$mpg >= median(Auto$mpg),]$mpg.bin = 1
```

(b) Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with

different values of this parameter. Comment on your results.

```
set.seed(30)
svc.auto = tune(svm,
                as.factor(mpg.bin) ~.,
                data = Auto,
                kernel = "linear",
                ranges = list(cost = c(10^(-3:1))))
summary(svc.auto)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.09435897 0.05127920
## 2 1e-02 0.07391026 0.04245856
## 3 1e-01 0.04339744 0.03210259
## 4 1e+00 0.01025641 0.01324097
## 5 1e+01 0.01525641 0.01313295
```

Evaluating $c = \{0.001,\ 0.01, 0.1,\ 1, 10\}$ we find that a cost of 1 gives the lowest cross validated missclassification error rate of 0.010.

(c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```
set.seed(15)
svm.poly.auto = tune(svm,
                as.factor(mpg.bin) ~.,
                data = Auto,
                kernel = "polynomial",
                ranges = list(cost = c(10^(-3:1)),
                              gamma = c(10^(-3:1)),
                              degree = c(2,3,4)
                              )
                )

svm.radial.auto = tune(svm,
                as.factor(mpg.bin) ~.,
                data = Auto,
                kernel = "radial",
                ranges = list(cost = c(10^(-3:1)),
                              gamma = c(10^(-3:1))
                              )
                )
summary(svm.poly.auto)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma degree
##   0.1     1      3
##
## - best performance: 0.03833333
##
## - Detailed performance results:
##     cost gamma degree      error dispersion
## 1  1e-03 1e-03      2 0.56634615 0.04940035
## 2  1e-02 1e-03      2 0.56634615 0.04940035
## 3  1e-01 1e-03      2 0.56634615 0.04940035
## 4  1e+00 1e-03      2 0.56634615 0.04940035
## 5  1e+01 1e-03      2 0.56634615 0.04940035
## 6  1e-03 1e-02      2 0.56634615 0.04940035
## 7  1e-02 1e-02      2 0.56634615 0.04940035
## 8  1e-01 1e-02      2 0.56634615 0.04940035
## 9  1e+00 1e-02      2 0.53044872 0.08892856
## 10 1e+01 1e-02      2 0.30358974 0.08833872
## 11 1e-03 1e-01      2 0.56634615 0.04940035
## 12 1e-02 1e-01      2 0.53044872 0.08892856
## 13 1e-01 1e-01      2 0.30358974 0.08833872
## 14 1e+00 1e-01      2 0.22993590 0.11125732
## 15 1e+01 1e-01      2 0.14544872 0.04376165
## 16 1e-03 1e+00      2 0.30358974 0.08833872
## 17 1e-02 1e+00      2 0.22993590 0.11125732
## 18 1e-01 1e+00      2 0.14544872 0.04376165
## 19 1e+00 1e+00      2 0.17852564 0.05516401
## 20 1e+01 1e+00      2 0.18865385 0.05637953
## 21 1e-03 1e+01      2 0.14544872 0.04376165
## 22 1e-02 1e+01      2 0.17852564 0.05516401
## 23 1e-01 1e+01      2 0.18865385 0.05637953
## 24 1e+00 1e+01      2 0.18865385 0.05637953
## 25 1e+01 1e+01      2 0.18865385 0.05637953
## 26 1e-03 1e-03      3 0.56634615 0.04940035
## 27 1e-02 1e-03      3 0.56634615 0.04940035
## 28 1e-01 1e-03      3 0.56634615 0.04940035
## 29 1e+00 1e-03      3 0.56634615 0.04940035
## 30 1e+01 1e-03      3 0.56634615 0.04940035
## 31 1e-03 1e-02      3 0.56634615 0.04940035
## 32 1e-02 1e-02      3 0.56634615 0.04940035
## 33 1e-01 1e-02      3 0.56634615 0.04940035
## 34 1e+00 1e-02      3 0.53814103 0.07936833
## 35 1e+01 1e-02      3 0.27288462 0.08852967
## 36 1e-03 1e-01      3 0.53814103 0.07936833
## 37 1e-02 1e-01      3 0.27288462 0.08852967
## 38 1e-01 1e-01      3 0.23467949 0.11383273
## 39 1e+00 1e-01      3 0.08160256 0.05005480
## 40 1e+01 1e-01      3 0.05102564 0.03376772
## 41 1e-03 1e+00      3 0.08160256 0.05005480
```

```
## 42 1e-02 1e+00      3 0.05102564 0.03376772
## 43 1e-01 1e+00      3 0.03833333 0.02763243
## 44 1e+00 1e+00      3 0.04608974 0.02916091
## 45 1e+01 1e+00      3 0.04608974 0.02916091
## 46 1e-03 1e+01      3 0.04608974 0.02916091
## 47 1e-02 1e+01      3 0.04608974 0.02916091
## 48 1e-01 1e+01      3 0.04608974 0.02916091
## 49 1e+00 1e+01      3 0.04608974 0.02916091
## 50 1e+01 1e+01      3 0.04608974 0.02916091
## 51 1e-03 1e-03      4 0.56634615 0.04940035
## 52 1e-02 1e-03      4 0.56634615 0.04940035
## 53 1e-01 1e-03      4 0.56634615 0.04940035
## 54 1e+00 1e-03      4 0.56634615 0.04940035
## 55 1e+01 1e-03      4 0.56634615 0.04940035
## 56 1e-03 1e-02      4 0.56634615 0.04940035
## 57 1e-02 1e-02      4 0.56634615 0.04940035
## 58 1e-01 1e-02      4 0.56634615 0.04940035
## 59 1e+00 1e-02      4 0.56634615 0.04940035
## 60 1e+01 1e-02      4 0.54070513 0.08764605
## 61 1e-03 1e-01      4 0.54070513 0.08764605
## 62 1e-02 1e-01      4 0.41596154 0.07812712
## 63 1e-01 1e-01      4 0.31121795 0.08689267
## 64 1e+00 1e-01      4 0.19666667 0.12393671
## 65 1e+01 1e-01      4 0.18602564 0.03446045
## 66 1e-03 1e+00      4 0.18602564 0.03446045
## 67 1e-02 1e+00      4 0.17858974 0.03827018
## 68 1e-01 1e+00      4 0.19884615 0.04570932
## 69 1e+00 1e+00      4 0.18352564 0.05008945
## 70 1e+01 1e+00      4 0.18352564 0.05008945
## 71 1e-03 1e+01      4 0.18352564 0.05008945
## 72 1e-02 1e+01      4 0.18352564 0.05008945
## 73 1e-01 1e+01      4 0.18352564 0.05008945
## 74 1e+00 1e+01      4 0.18352564 0.05008945
## 75 1e+01 1e+01      4 0.18352564 0.05008945
```

```
summary(svm.radial.auto)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.02551282
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03 1e-03 0.55102564 0.04000260
## 2  1e-02 1e-03 0.55102564 0.04000260
## 3  1e-01 1e-03 0.52814103 0.05965057
## 4  1e+00 1e-03 0.08685897 0.05446434
## 5  1e+01 1e-03 0.07673077 0.05552436
```

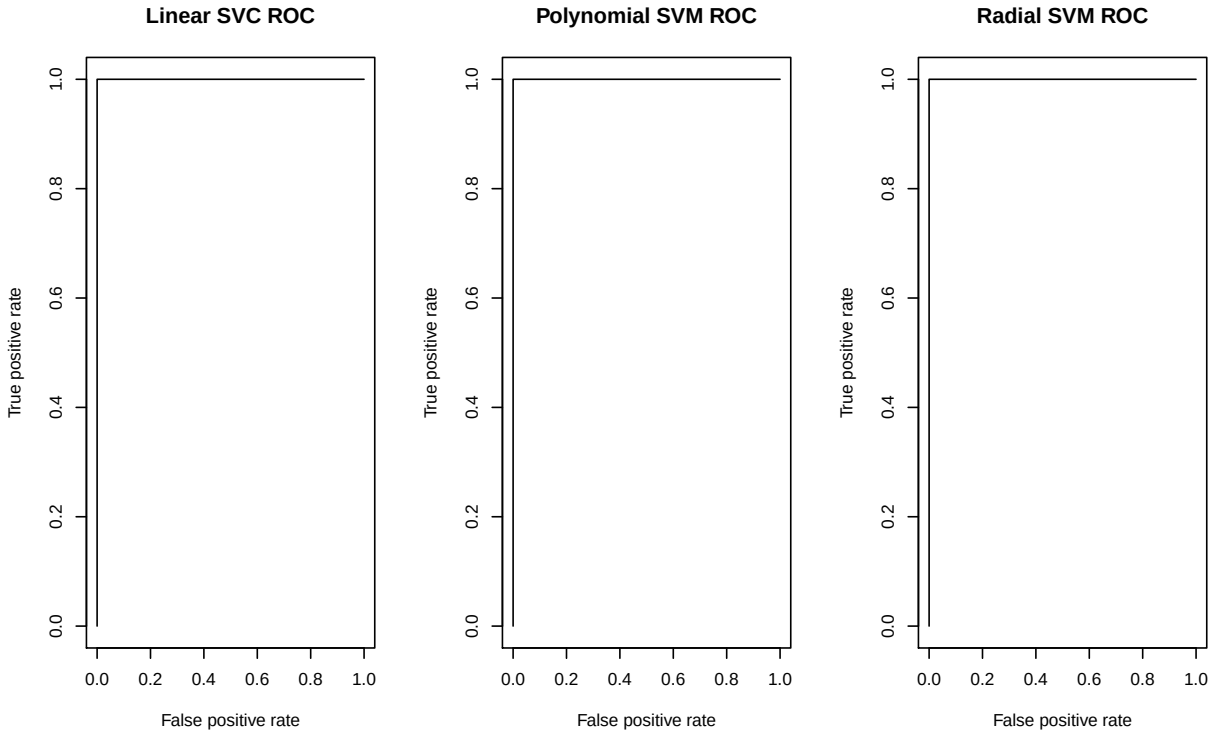
```
## 6  1e-03 1e-02 0.55102564 0.04000260
## 7  1e-02 1e-02 0.55102564 0.04000260
## 8  1e-01 1e-02 0.08685897 0.05446434
## 9  1e+00 1e-02 0.07673077 0.05552436
## 10 1e+01 1e-02 0.03070513 0.02357884
## 11 1e-03 1e-01 0.55102564 0.04000260
## 12 1e-02 1e-01 0.18615385 0.07412233
## 13 1e-01 1e-01 0.07673077 0.05552436
## 14 1e+00 1e-01 0.05871795 0.03642670
## 15 1e+01 1e-01 0.02551282 0.02417610
## 16 1e-03 1e+00 0.55102564 0.04000260
## 17 1e-02 1e+00 0.55102564 0.04000260
## 18 1e-01 1e+00 0.55102564 0.04000260
## 19 1e+00 1e+00 0.06634615 0.03244101
## 20 1e+01 1e+00 0.06384615 0.03879626
## 21 1e-03 1e+01 0.55102564 0.04000260
## 22 1e-02 1e+01 0.55102564 0.04000260
## 23 1e-01 1e+01 0.55102564 0.04000260
## 24 1e+00 1e+01 0.52294872 0.06031079
## 25 1e+01 1e+01 0.51032051 0.05494774
```

We cross-validate the polynomial and radial svm models using $c, \gamma = \{0.001,\ 0.01, 0.1,\ 1, 10\}$ and $d = \{2,\ 3,\ 4\}$. The best polynomial parameters achieved a cross-validated missclassification error rate of 0.038 ($c = 0.01,\ \gamma = 1,\ d = 3$). The best radial model parameters achieved a cross-validated missclassification error rate of 0.025 ($c = 10,\ \gamma = 0.1$). This suggests that a linear svc is satisfactory and the polynomial and radial svm are not an improvement.

(d) Make some plots to back up your assertions in (b) and (c).

The graphs below show the equivalent performance of the optimal tuned linear, polynomial, and radial svm models.

```
fitted.linear = -1*attributes(predict(svc.auto$best.model,Auto, decision.values = TRUE))$decision.values

fitted.poly = -1*attributes(predict(svm.poly.auto$best.model,Auto, decision.values = TRUE))$decision.va

fitted.radial = -1*attributes(predict(svm.radial.auto$best.model,Auto, decision.values = TRUE))$decisio

par(mfrow = c(1,3))
rocplot(fitted.linear, as.factor(Auto[,"mpg.bin"]),
        main= "Linear SVC ROC")

rocplot(fitted.poly, as.factor(Auto[,"mpg.bin"]),
        main = "Polynomial SVM ROC")

rocplot(fitted.radial, as.factor(Auto[,"mpg.bin"]),
        main = "Radial SVM ROC")
```



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.

```
library(ISLR)

#create test and training set
set.seed(1)
train = sample(1:nrow(OJ), size = 800, replace = FALSE)
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.

```
svc.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "linear",
             cost = 0.01)
summary(svc.OJ)
```

```
##
## Call:
## svm(formula = as.factor(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
```

The support vector classifier (i.e. linear kernel) has 443 support vectors ($ > 50\%$ of data) with near even balance of the support vectors among the classes.

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

```
preds.train = predict(svc.OJ, newdata = OJ.train)
preds.test = predict(svc.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.166"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.181"
```

The training error rate is 0.166 and the test error rate is 0.17.

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

```
set.seed(4)
svc.OJ = tune(svm,
              as.factor(Purchase) ~ .,
              data = OJ.train,
              kernel = "linear",
              ranges = list(cost = c(10^(-2:1))))
summary(svc.OJ)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
```

```
## - best performance: 0.1675
##
## - Detailed performance results:
##    cost  error dispersion
## 1  0.01 0.1700 0.04417453
## 2  0.10 0.1675 0.04937104
## 3  1.00 0.1700 0.04647281
## 4 10.00 0.1750 0.04859127
```

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

```
svc.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "linear",
             cost = 1)
preds.train = predict(svc.OJ, newdata = OJ.train)
preds.test = predict(svc.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.159"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.193"
```

The training error rate is 0.161 and the test error rate is 0.178. This is better training performance and worse test performance compared to a cost of 0.01. This suggests a cost of 1 overfits the data.

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

```
svm.radial.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "radial",
             cost = 0.01)
summary(svm.radial.OJ)
```

```
##
## Call:
## svm(formula = as.factor(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:  617
##
##  ( 306 311 )
##
```

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

The support vector machine with radial kernel has 661 support vectors ($ > 75%$ of data) with near even balance of the support vectors among the classes.

```
preds.train = predict(svm.radial.OJ, newdata = OJ.train)
preds.test = predict(svm.radial.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.382"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.411"
```

The training error rate of the radial kernel svm is 0.412 and the test error is 0.322.

```
set.seed(4)
svm.radial.OJ = tune(svm,
              as.factor(Purchase) ~ .,
              data = OJ.train,
              kernel = "radial",
              ranges = list(cost = c(10^(-2:1))))
summary(svm.radial.OJ)
```

```
##
## 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 0.38250 0.03291403
## 2  0.10 0.17250 0.04887626
## 3  1.00 0.16625 0.03866254
## 4 10.00 0.17750 0.03476109
```

```
svm.radial.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "radial",
             cost = 1)
preds.train = predict(svm.radial.OJ, newdata = OJ.train)
preds.test = predict(svm.radial.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.145"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.17"
```

The training error rate of the radial kernel svm is 0.151 and the test error is 0.159 with optimal cost of 1. This is better than the radial svm with cost value 0.01 and also the optimal support vector classifier with linear kernel.

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

```
svm.poly.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "polynomial",
             cost = 0.01,
             degree = 2)
summary(svm.poly.OJ)
```

```
##
## Call:
## svm(formula = as.factor(Purchase) ~ ., data = OJ.train, kernel = "polynomial",
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  620
##
##  ( 306 314 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The support vector machine with polynomial kernel has 663 support vectors ($ > 75\%$ of data) with near even balance of the support vectors among the classes.

```
preds.train = predict(svm.poly.OJ, newdata = OJ.train)
preds.test = predict(svm.poly.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.382"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.411"
```

The training error rate of the radial kernel svm is 0.411 and the test error is 0.322.

```
set.seed(4)
svm.poly.OJ = tune(svm,
              as.factor(Purchase) ~ .,
              data = OJ.train,
              kernel = "polynomial",
              ranges = list(cost = c(10^(-2:1))))
summary(svm.poly.OJ)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1775
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.36750 0.03872983
## 2  0.10 0.30125 0.04656611
## 3  1.00 0.18875 0.03458584
## 4 10.00 0.17750 0.03763863
```

```
svm.poly.OJ = svm(as.factor(Purchase) ~ .,
             data = OJ.train,
             kernel = "polynomial",
             cost = 1)
preds.train = predict(svm.poly.OJ, newdata = OJ.train)
preds.test = predict(svm.poly.OJ, newdata = OJ.test)
error.train = round(1- sum(preds.train == OJ.train$Purchase)/nrow(OJ.train),3)
error.test = round(1 - sum(preds.test == OJ.test$Purchase)/nrow(OJ.test),3)
print(paste0("The training error rate is ", error.train))
```

```
## [1] "The training error rate is 0.154"
```

```
print(paste0("The test error rate is ", error.test))
```

```
## [1] "The test error rate is 0.2"
```

The training error rate of the radial kernel svm is 0.164 and the test error is 0.163 with optimal cost of 1. This slightly worse than the optimal radial svm with cost value 1 but better than the optimal support vector classifier with linear kernel.

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

The highest performing model in terms of training and test error is the radial kernel svm with cost value 1.