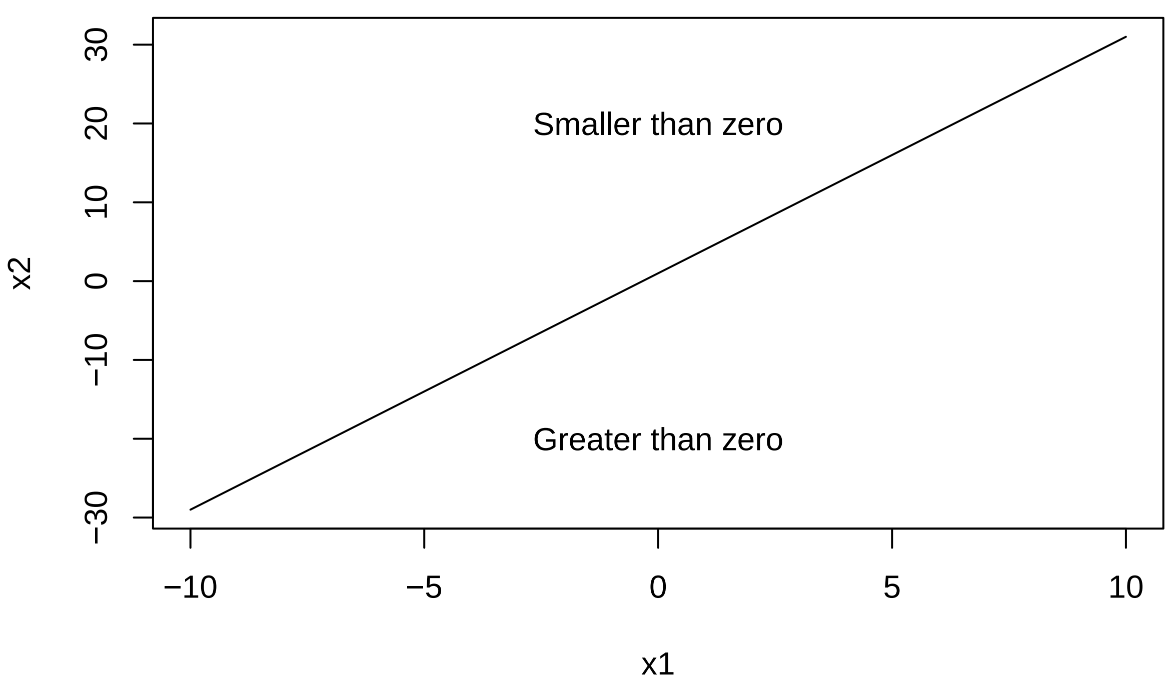
Chapter 9 Support Vector Machines

```
library(ggplot2)
library(plotrix)
library(e1071)
library(ISLR2)
```

Exercise 1

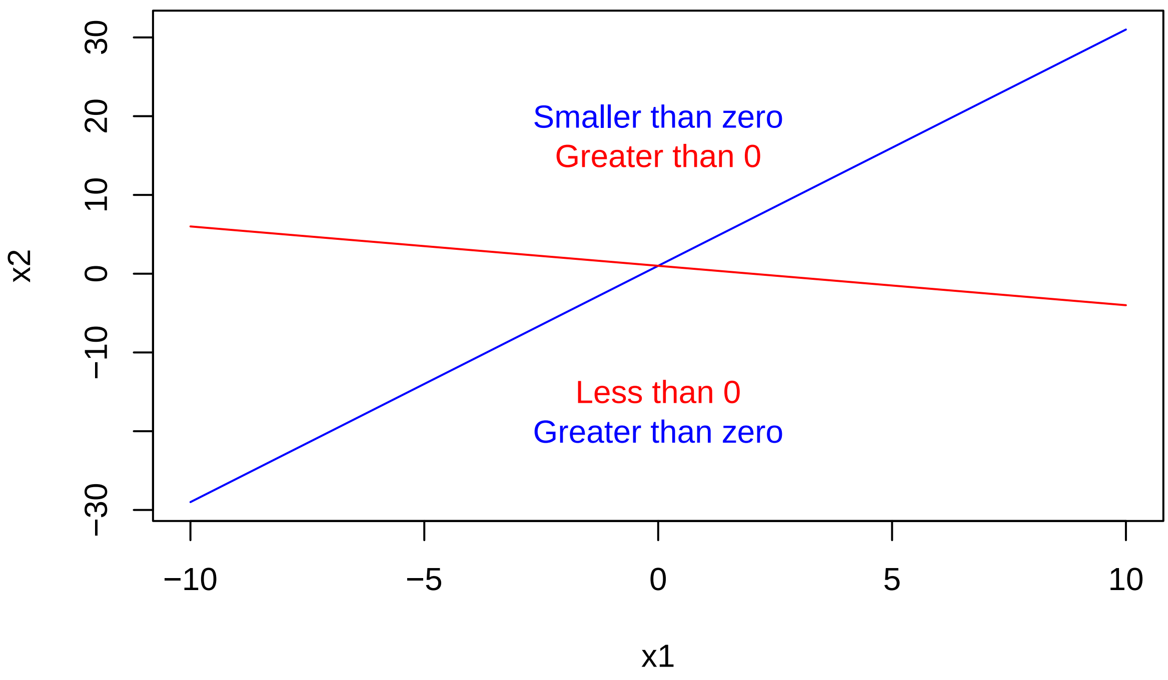
a

```
x1 = c(-10:10)
x2 = 1 + 3*x1
plot(x1, x2, type = 'l')
text(c(0), c(-20), 'Greater than zero')
text(c(0), c(20), 'Smaller than zero')
```



b

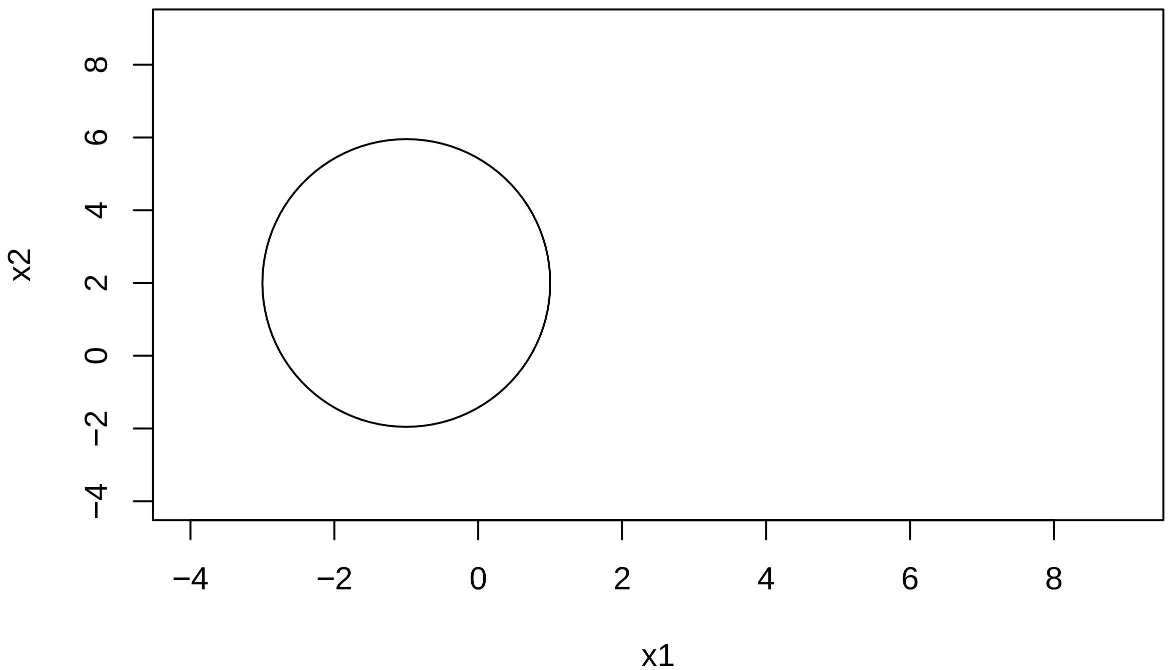
```
x1 = c(-10:10)
x2 = 1 + 3*x1

plot(x1, x2, type = 'l', col = 'blue')
text(c(0), c(-20), 'Greater than zero', col = 'blue')
text(c(0), c(20), 'Smaller than zero', col = 'blue')

lines(x1, 1 - x1/2, col = 'red')
text(c(0), c(-15), 'Less than 0', col = 'red')
text(c(0), c(15), 'Greater than 0', col = 'red')
```



Exercise 2

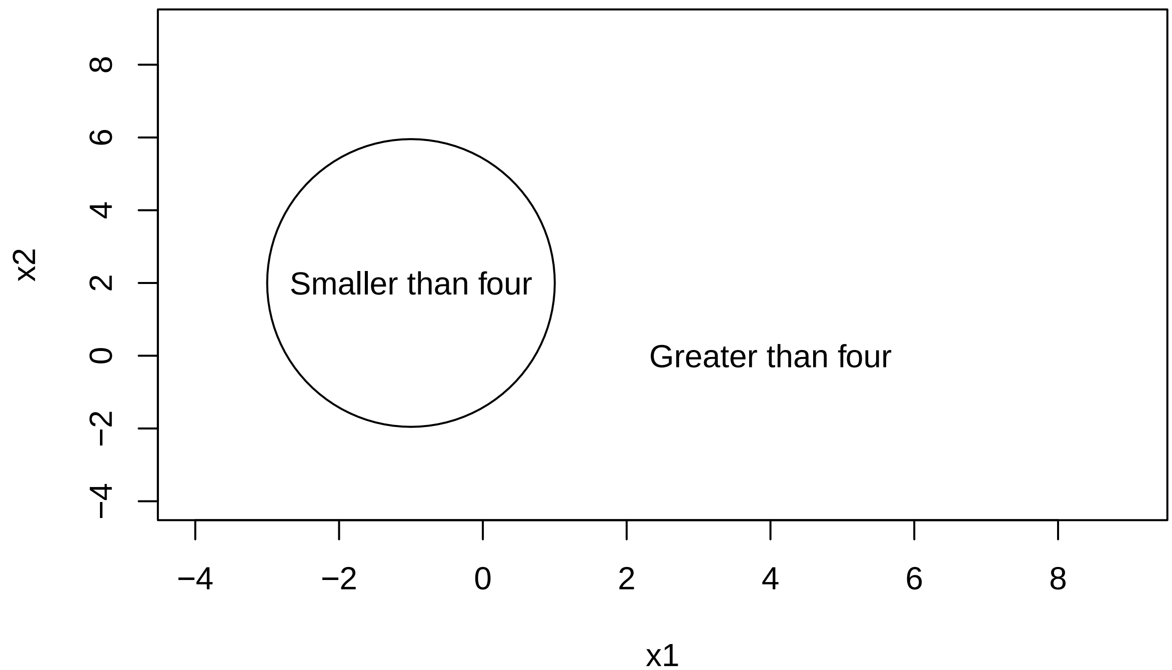
a

```
plot(-4:9, -4:9, type='n', xlab = 'x1', ylab = 'x2')
draw.circle(-1, 2, 2)
```



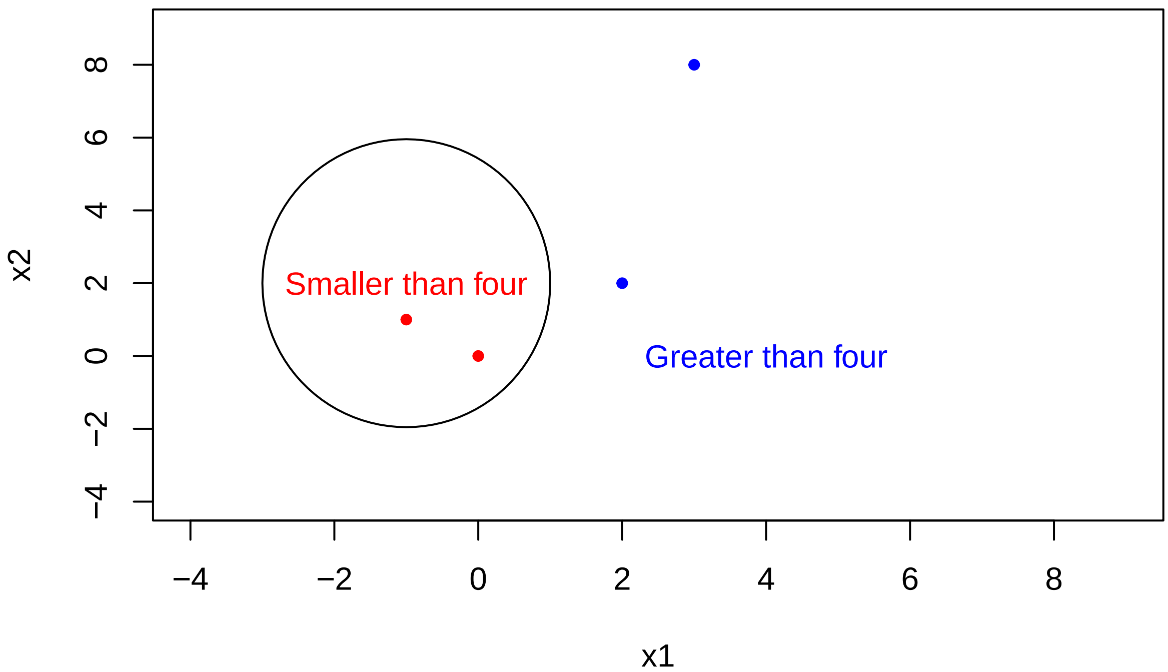
b

```
plot(-4:9, -4:9, type='n', xlab = 'x1', ylab = 'x2')
draw.circle(-1, 2, 2)

text(c(4), c(0), 'Greater than four')
text(c(-1), c(2), 'Smaller than four')
```



c

```
plot(-4:9, -4:9, type='n', xlab = 'x1', ylab = 'x2')
draw.circle(-1, 2, 2)

text(c(-1), c(2), 'Smaller than four', col = 'red')
text(c(4), c(0), 'Greater than four', col = 'blue')

points(0, 0, pch = 20, col = 'red')
points(-1, 1, pch = 20, col = 'red')
points(2, 2, pch = 20, col = 'blue')
points(3, 8, pch = 20, col = 'blue')
```



d

We can rewrite the function as

$$X_1^2 + 2X_1 + X_2^2 - 4X_2 + 1 = 0$$

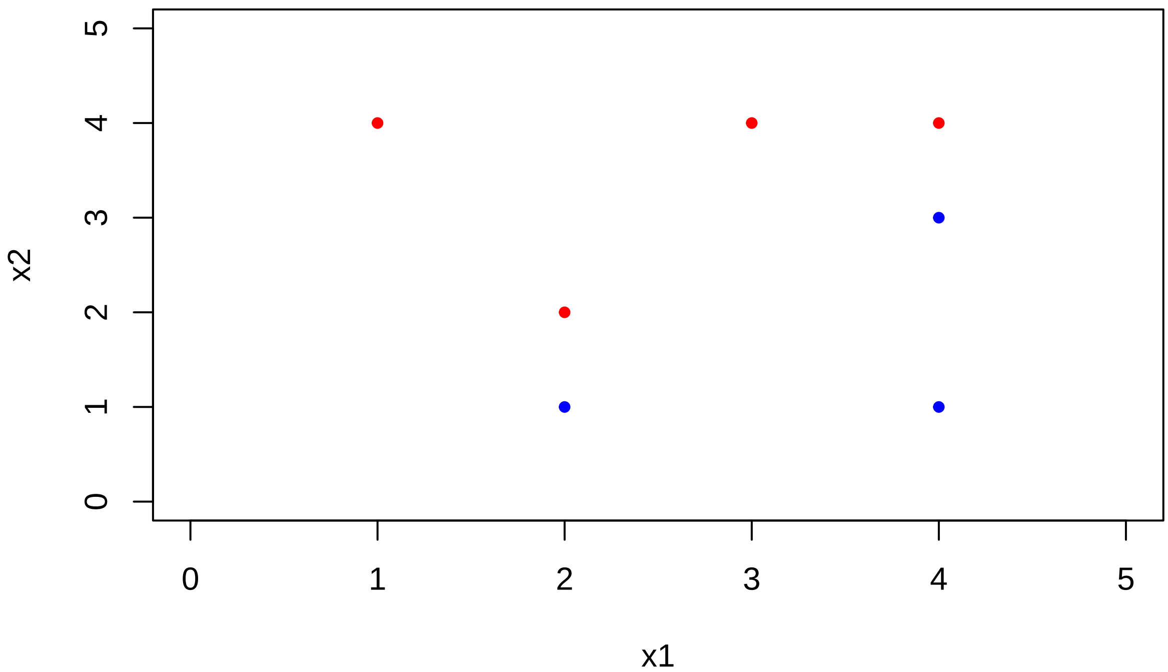
Here, the decision boundary is linear in terms of

$$X_1, X_1^2, X_2, X_2^2$$

Exercise 3

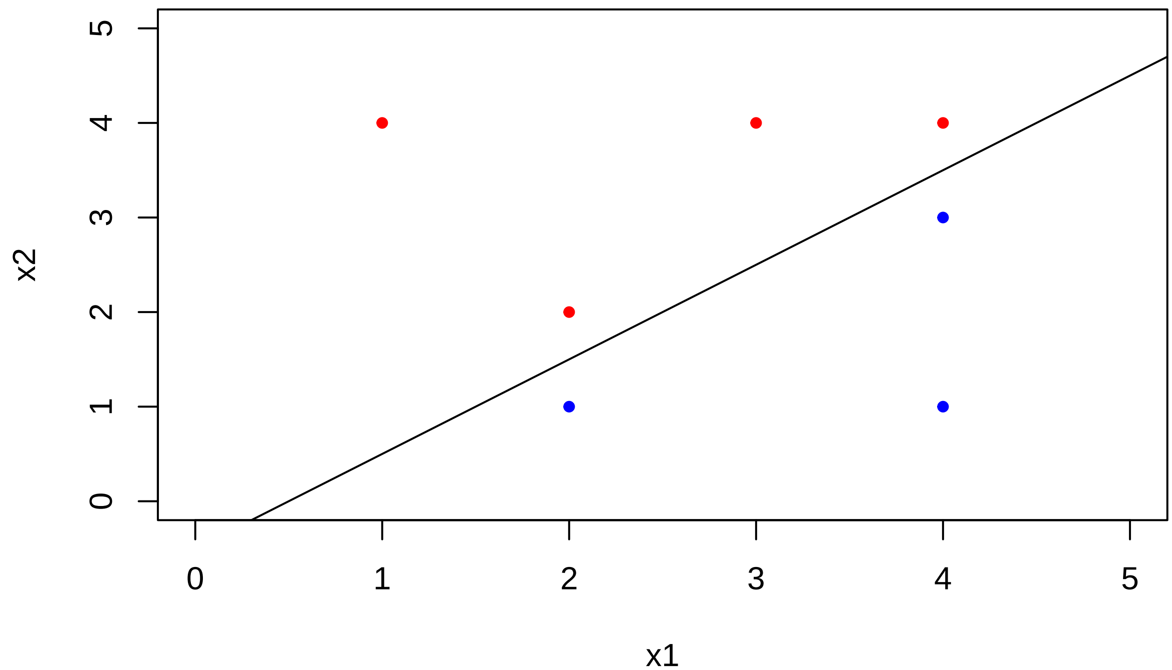
a

```r
plot(0:5, 0:5, type = 'n', xlab = 'x1', ylab = 'x2')

points(3, 4, pch = 20, col = 'red')
points(2, 2, pch = 20, col = 'red')
points(4, 4, pch = 20, col = 'red')
points(1, 4, pch = 20, col = 'red')
points(2, 1, pch = 20, col = 'blue')
points(4, 3, pch = 20, col = 'blue')
points(4, 1, pch = 20, col = 'blue')
```



b

```
plot(0:5, 0:5, type = 'n', xlab = 'x1', ylab = 'x2')

points(3, 4, pch = 20, col = 'red')
points(2, 2, pch = 20, col = 'red')
points(4, 4, pch = 20, col = 'red')
points(1, 4, pch = 20, col = 'red')
points(2, 1, pch = 20, col = 'blue')
points(4, 3, pch = 20, col = 'blue')
points(4, 1, pch = 20, col = 'blue')

# X1 - X2 - 0.5 = 0
x1 = c(0, 6)
x2 = x1 - 0.5
abline(-0.5, 1)
```



c

The classification rule for the maximal margin classifier takes the form

$$f(X) = X_1 - X_2 - 0.5$$

For a particular value of

$$X$$

,

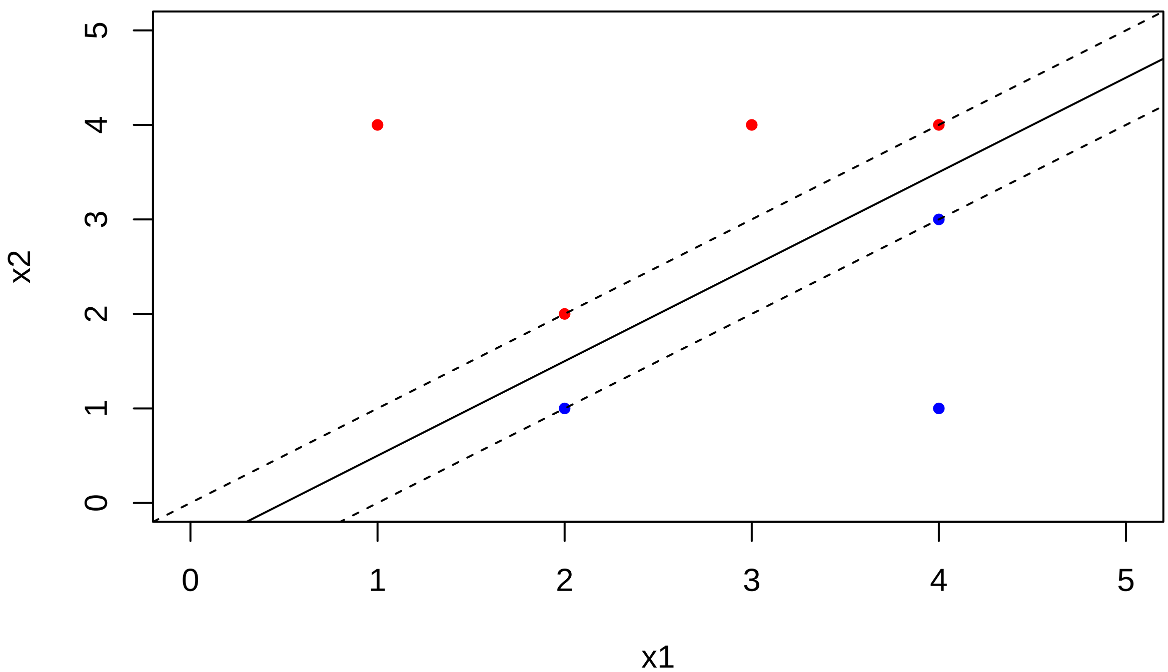
$$\text{If } f(X) < 0\text{, classify as Red and if } f(X) > 0\text{, classify as Blue.}$$

$$\beta_0, \beta_2 \text{ and } \beta_3 \text{ are } 1, -1, -0.5 \text{ respectively.}$$

d

```
plot(0:5, 0:5, type = 'n', xlab = 'x1', ylab = 'x2')

points(3, 4, pch = 20, col = 'red')
points(2, 2, pch = 20, col = 'red')
points(4, 4, pch = 20, col = 'red')
```

```
points(1, 4, pch = 20, col = 'red')
points(2, 1, pch = 20, col = 'blue')
points(4, 3, pch = 20, col = 'blue')
points(4, 1, pch = 20, col = 'blue')

# X1 - X2 - 0.5 = 0
x1 = c(0, 6)
x2 = x1 - 0.5
abline(-0.5, 1, lty = 1)
abline(0, 1, lty = 2)
abline(-1, 1, lty = 2)
```



e

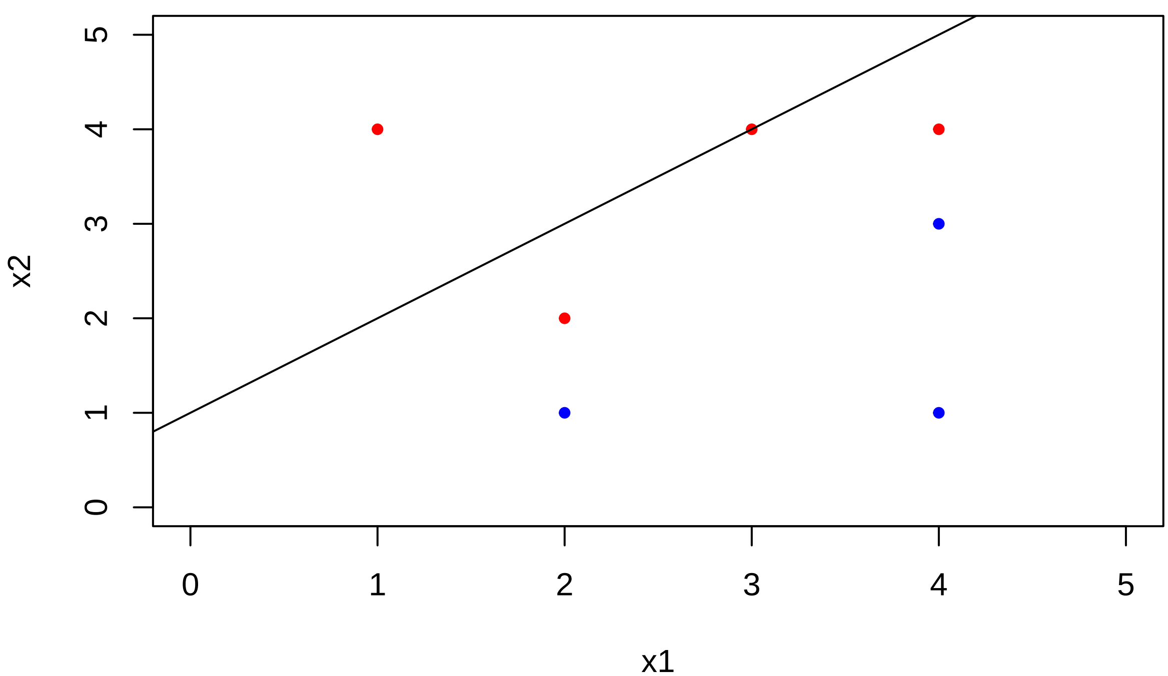
The support vectors for the maximal margin classifier are vectors (2, 2), (4, 4), (2, 1) and (4, 1)

f

The 7th observation is not the support vector of the maximal margin classifier so a slight movement from it does not affect the maximal margin hyperplane.

g

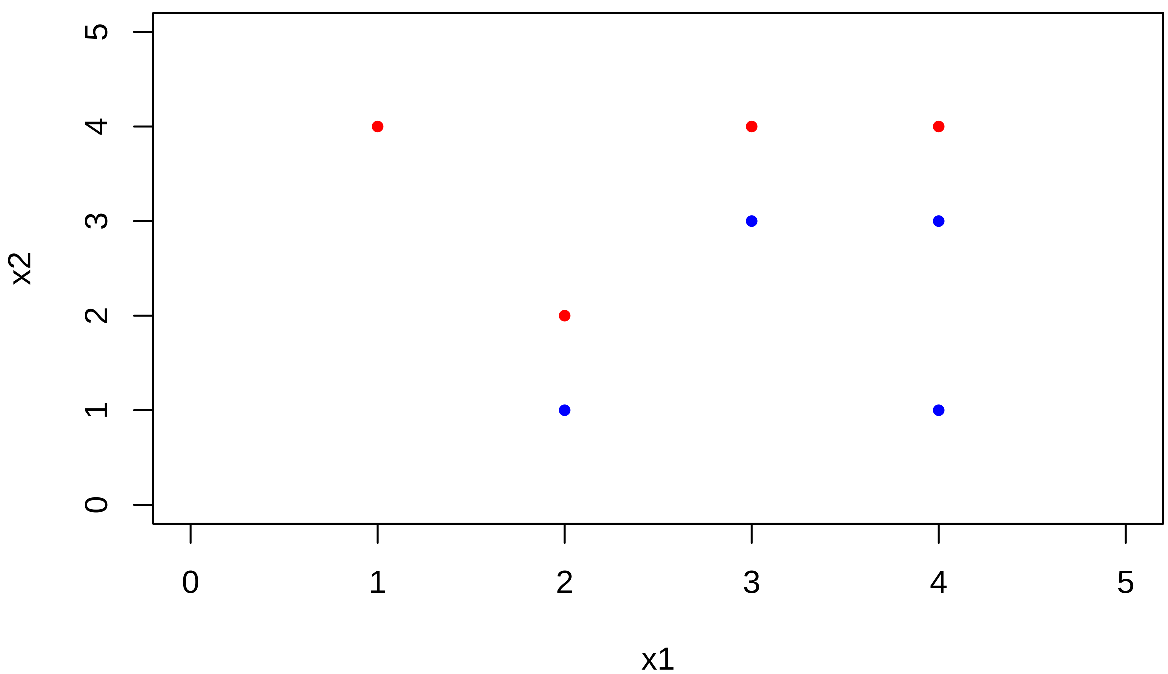
```
plot(0:5, 0:5, type = 'n', xlab = 'x1', ylab = 'x2')

points(3, 4, pch = 20, col = 'red')
points(2, 2, pch = 20, col = 'red')
points(4, 4, pch = 20, col = 'red')
points(1, 4, pch = 20, col = 'red')
points(2, 1, pch = 20, col = 'blue')
points(4, 3, pch = 20, col = 'blue')
points(4, 1, pch = 20, col = 'blue')

# X1 - X2 - 0.5 = 0
x1 = c(0, 6)
x2 = x1 - 0.5
abline(1, 1, lty = 1)
```



h

```
plot(0:5, 0:5, type = 'n', xlab = 'x1', ylab = 'x2')

points(3, 4, pch = 20, col = 'red')
```

```
points(2, 2, pch = 20, col = 'red')
points(4, 4, pch = 20, col = 'red')
points(1, 4, pch = 20, col = 'red')
points(2, 1, pch = 20, col = 'blue')
points(4, 3, pch = 20, col = 'blue')
points(4, 1, pch = 20, col = 'blue')

# X1 - X2 - 0.5 = 0
x1 = c(0, 6)
x2 = x1 - 0.5
#abline(-0.5, 1, lty = 1)

points(3, 3, pch = 20, col = 'blue')
```



Exercise 4

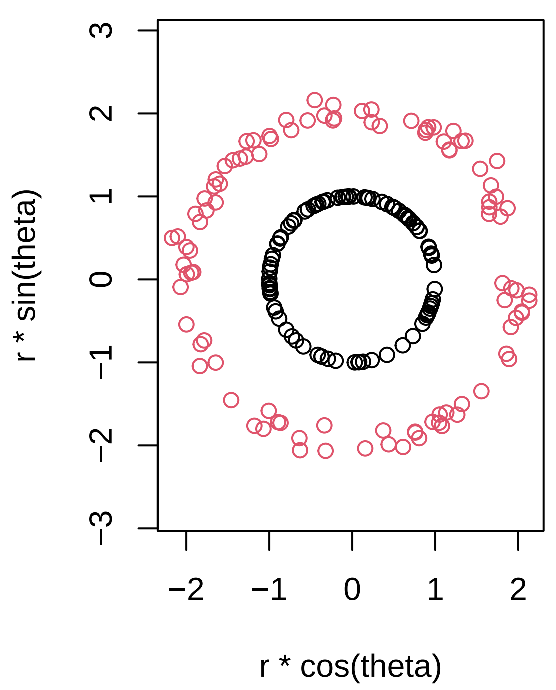
A radial kernel is likely to outperform a support vector classifier because we don't have many features and it's obvious that splitting the observations with a curve (a polynomial kernel) is more appropriate than a straight line (a linear kernel)

```
circles = function(n, mu, sigma) {
    lr = Map(rlnorm, n = n, meanlog = mu, sdlog = sigma)
    N = length(lr)
```

```r
    n = lengths(lr, FALSE)
    data.frame(group = rep.int(gl(N, 1L), n),
               r = unlist(lr, FALSE, FALSE),
               theta = runif(sum(n), 0, 2 * pi))
}

set.seed(1)
d = circles(n = c(100, 100), mu = log(c(1, 2)), sigma = c(0, 0.05))
str(d)
```

```
## 'data.frame':    200 obs. of  3 variables:
##  $ group: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ r    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ theta: num  1.68 1.37 3.25 1.69 1.14 ...
```

```r
par(mfrow = c(1, 2))
with(d, {
    plot(r * cos(theta), r * sin(theta), asp = 1, col = group)
})
```

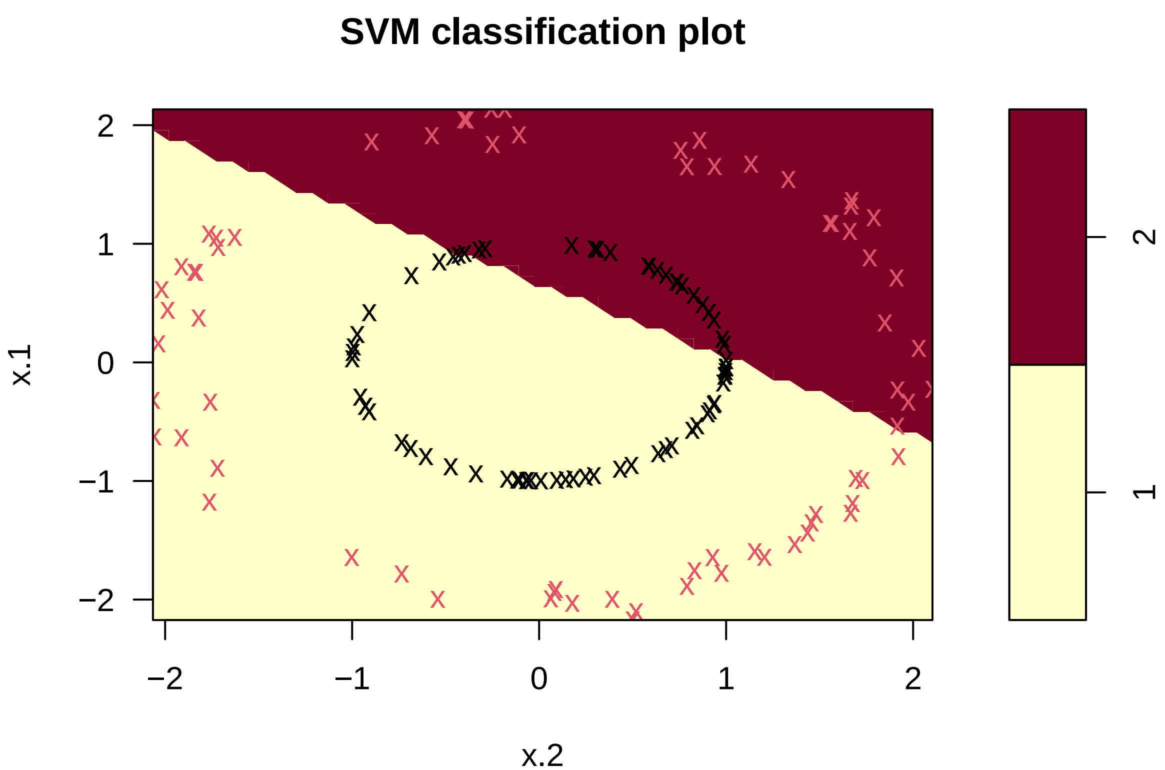



```r
set.seed(42)
train_indices = sample(200, 200*0.7)
x_train = matrix(c(cos(d$theta[train_indices]) * d$r[train_indices],
                   sin(d$theta[train_indices]) * d$r[train_indices]), ncol = 2)
```

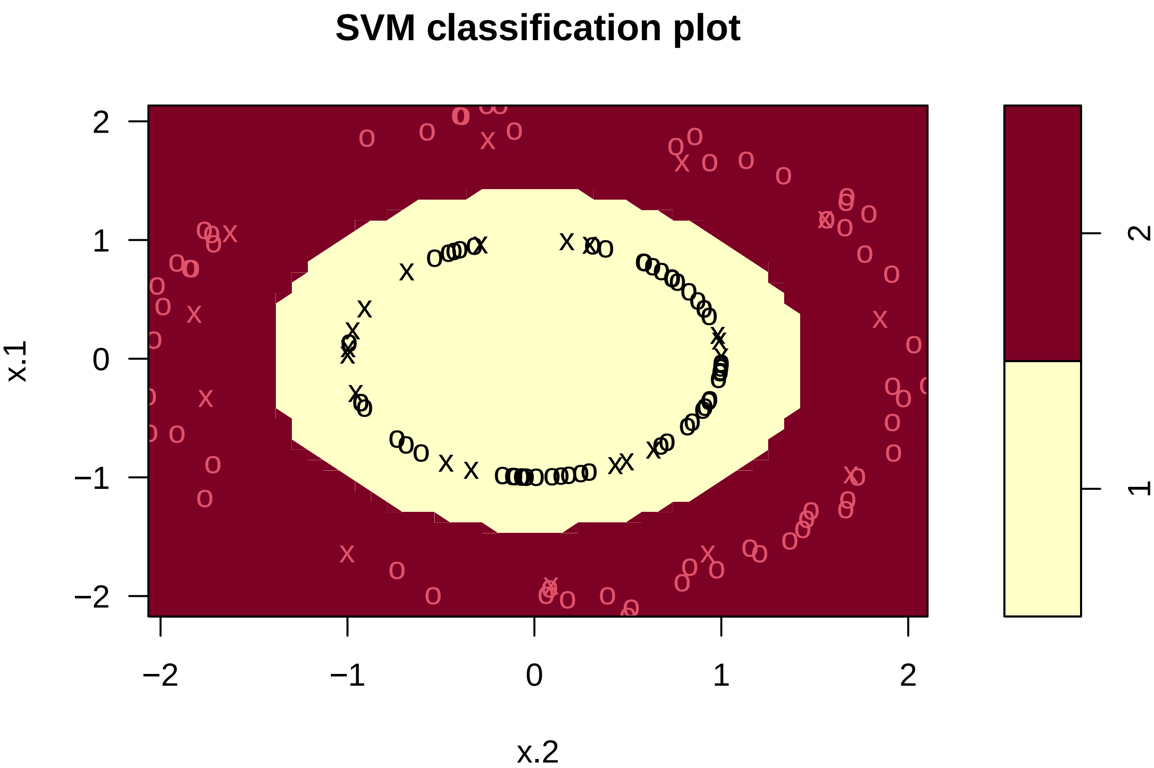
```
y_train = d$group[train_indices]

x_test = matrix(c(cos(d$theta[-train_indices]) * d$r[-train_indices],
                   sin(d$theta[-train_indices]) * d$r[-train_indices]), ncol = 2)
y_test = d$group[-train_indices]

train_set = data.frame(x = x_train, y = y_train)
test_set = data.frame(x = x_test, y = y_test)
```

```
svc_model = svm(y ~ ., data = train_set, kernel = 'linear', cost = 1)
```

```
plot(svc_model, train_set)
```



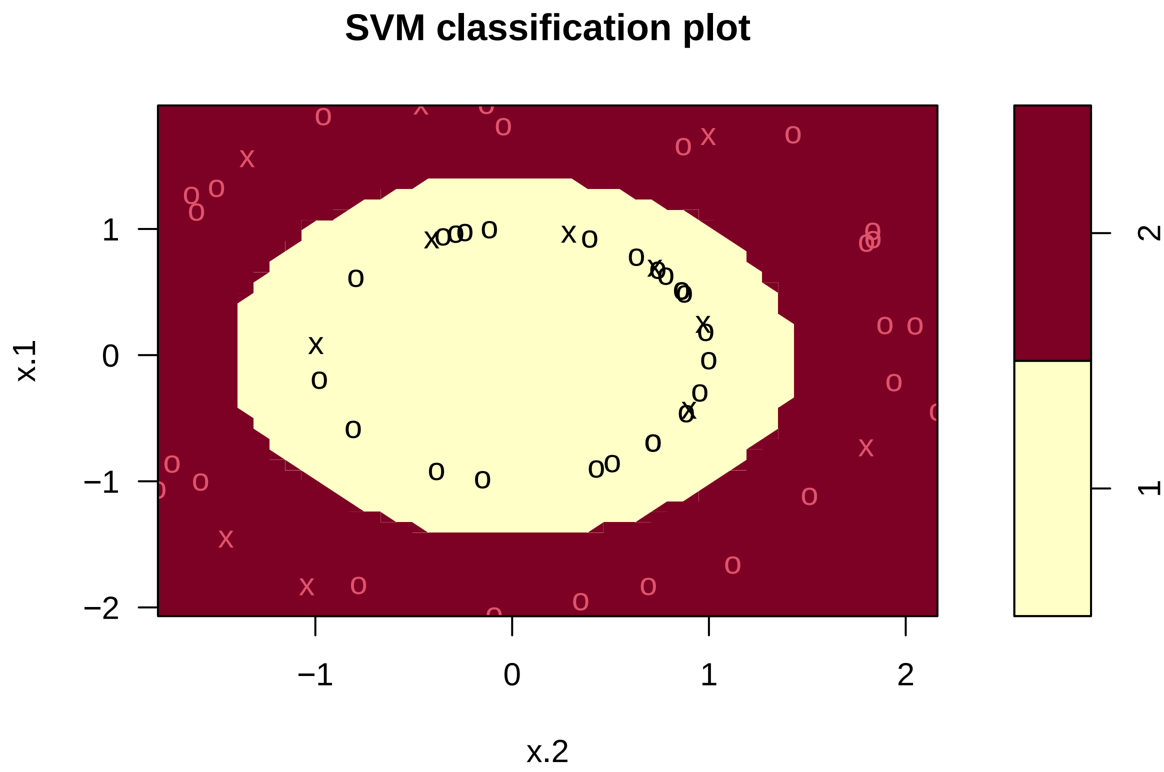

```
svm_model = svm(y ~ ., data = train_set, kernel = 'radial',
                cost = 1, gamma = 1)
plot(svm_model, train_set)
```




```
table(svm_model$fitted, y_train)
```

```
##    y_train
##      1  2
##   1 70  0
##   2  0 70
```

```
plot(svm_model, test_set)
```




```
test_pred = predict(svm_model, test_set)
table(test_pred, y_test)
```

```
##          y_test
## test_pred  1  2
##         1 30  0
##         2  0 30
```

Exercise 5

a

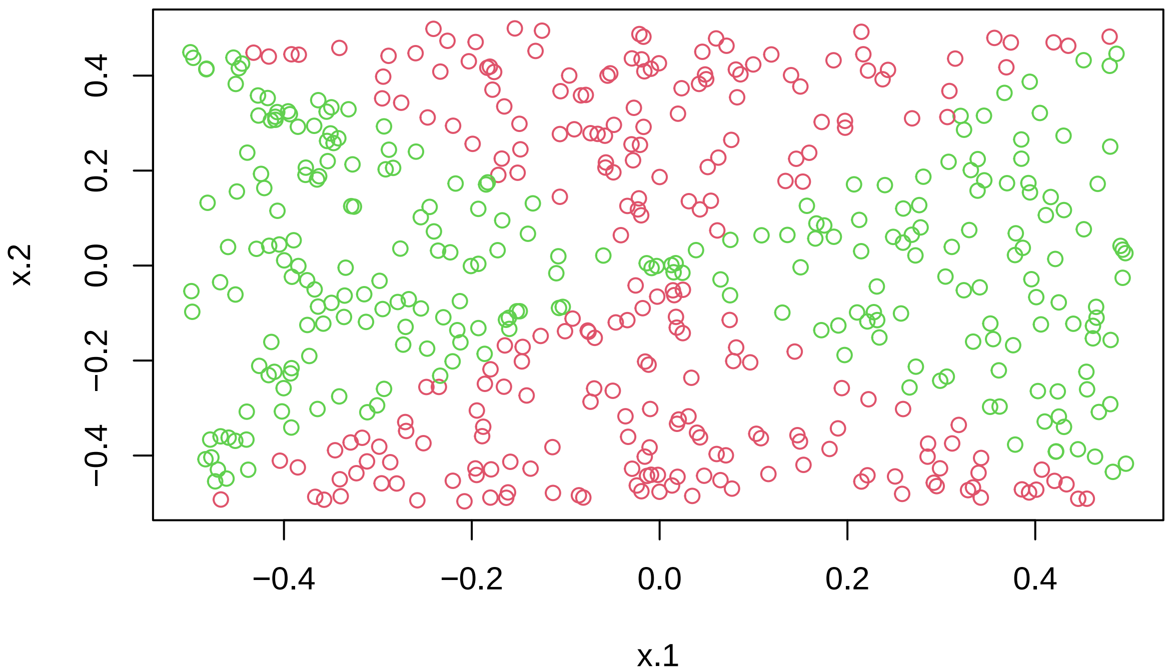
```
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = (x1**2 - x2**2 > 0) * 1
dat = data.frame(x = matrix(c(x1, x2), ncol = 2), y = as.factor(y))
head(dat)
```

```
##          x.1         x.2 y
## 1  0.24318772  0.41202998 0
## 2  0.23131548 -0.04402148 1
## 3  0.38511769  0.26552048 1
```

```
## 4  0.01711106  0.00504458 1
## 5  0.35193098 -0.29745152 1
## 6 -0.05720373  0.21713872 0
```

b

```
plot(dat[, 1:2], col = y + 2)
```



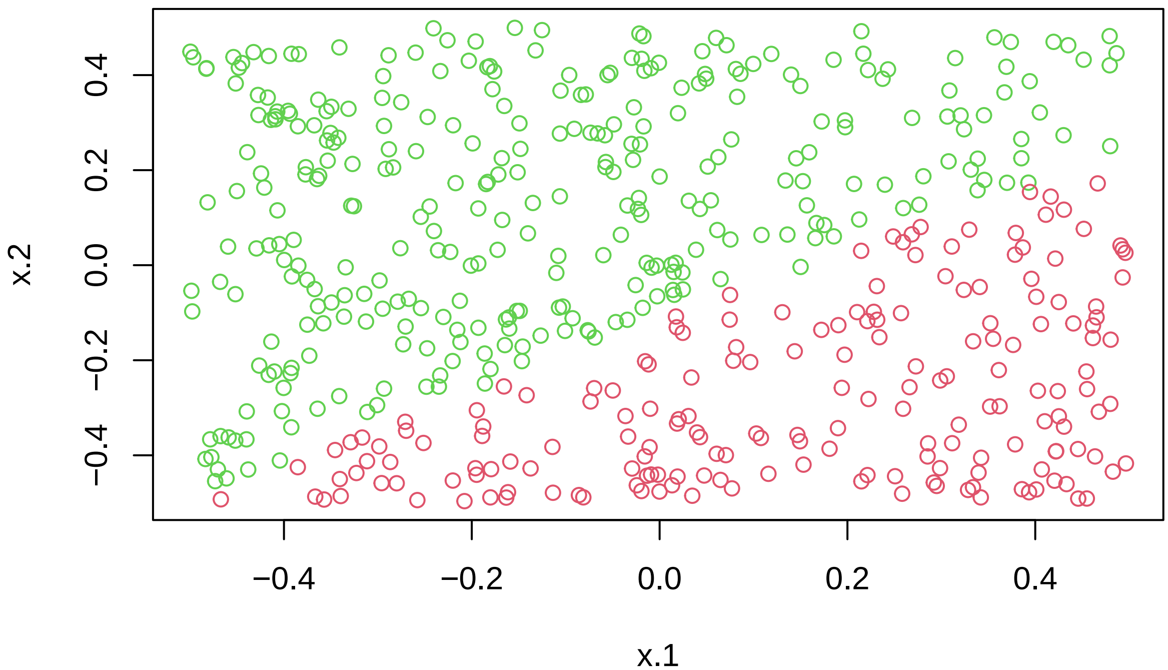
c

```
LR_model = glm(y ~ ., data = dat, family = binomial)
train_pred = (LR_model$fitted.values > 0.5) * 1
table(train_pred, dat$y)
```

```
##
## train_pred   0   1
##          0 112  74
##          1 132 182
```

d

```
plot(dat[, 1:2], col = train_pred + 2)
```



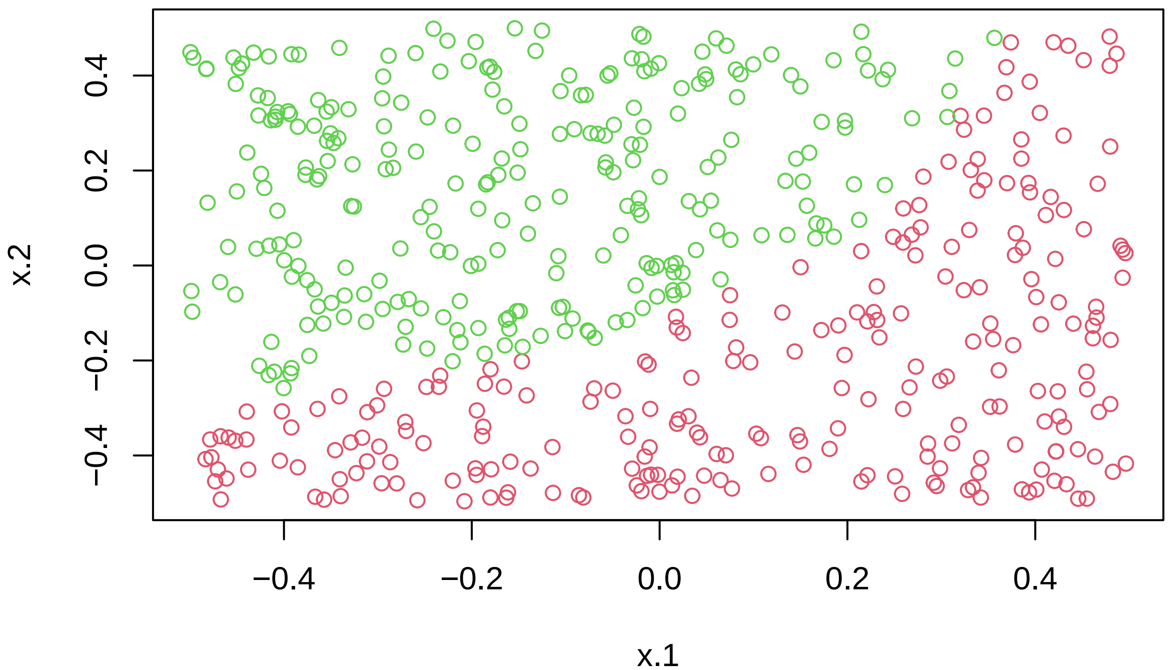
e

```
poly_LR_model = glm(y ~ x1 + x2 + x1^2 + x2^2 + x1*x2 + x1^3 + x2^3,
                    data = dat, family = 'binomial')
train_pred = (poly_LR_model$fitted.values > 0.5) * 1
table(train_pred, dat$y)
```

```
##
## train_pred   0   1
##          0 123 118
##          1 121 138
```

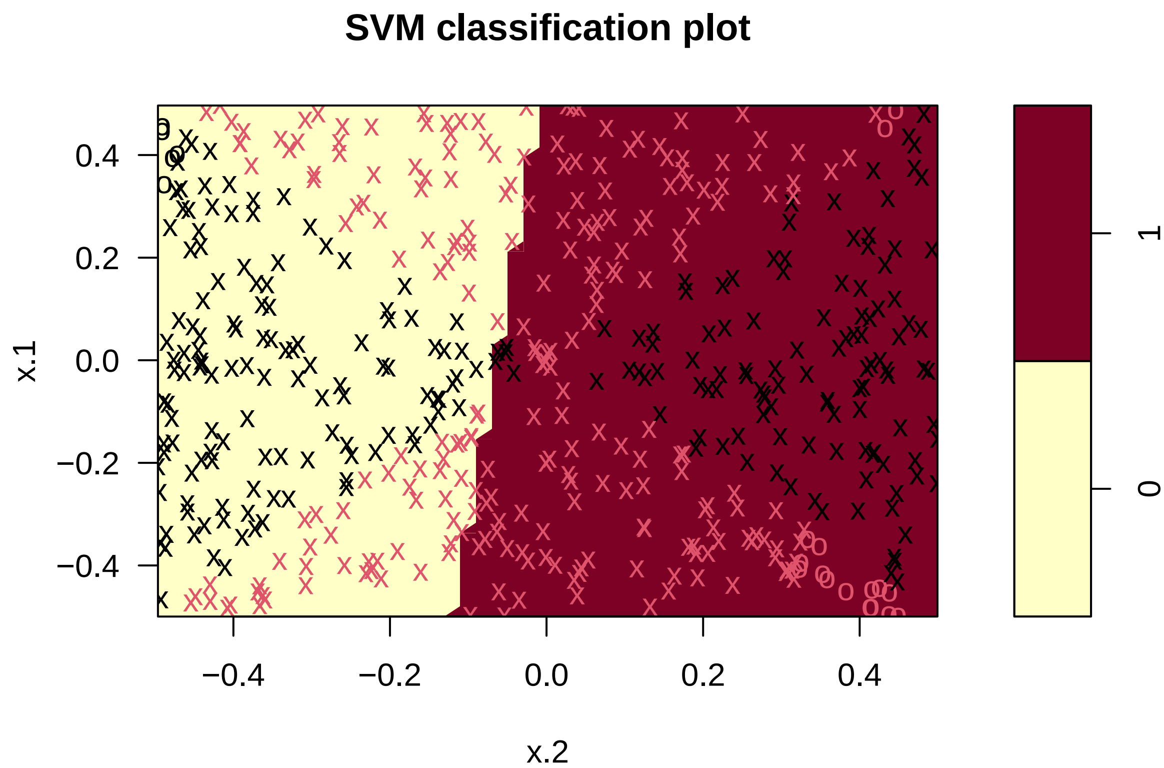
f

```
plot(dat[, 1:2], col = train_pred + 2)
```



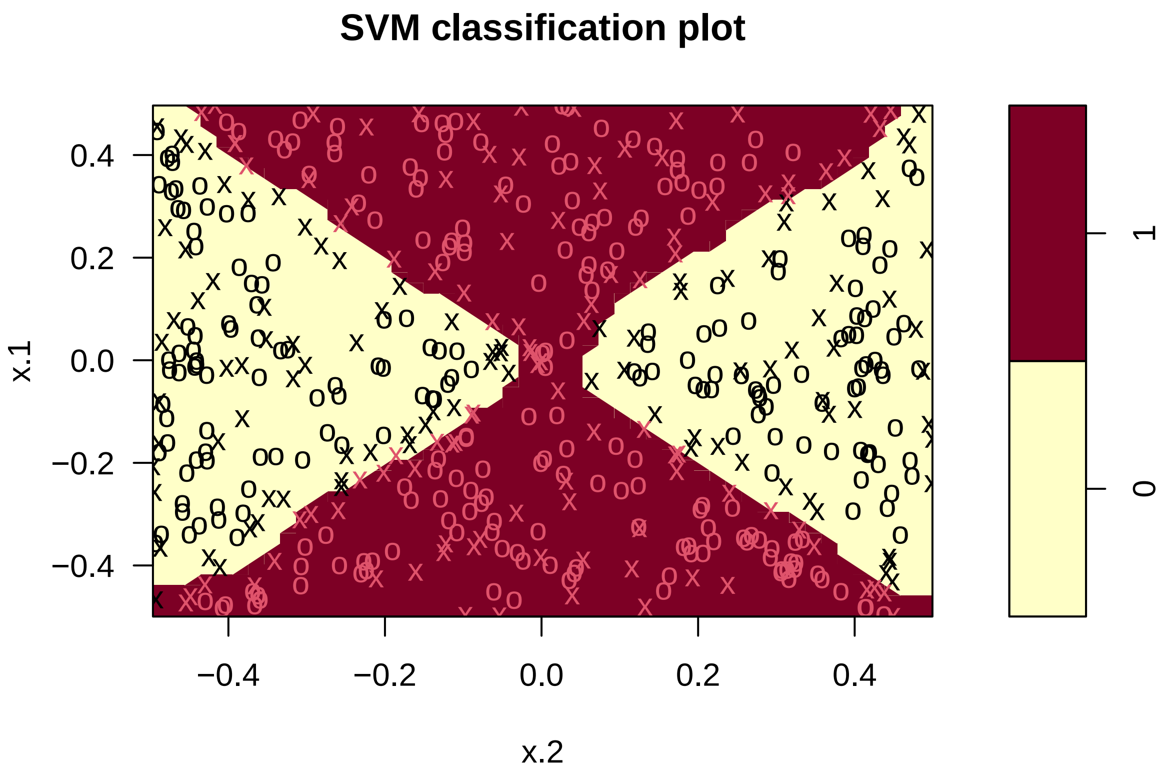
g

```r
svc_model = svm(y ~ ., data = dat, kernel = 'linear', cost = 1)

#table(svc_model$fitted, dat$y)
plot(svc_model, dat)
```



h

```
SVM_model = svm(y ~ ., data = dat, kernel = 'radial',
                cost = 1, gamma = 10)
plot(SVM_model, dat)
```



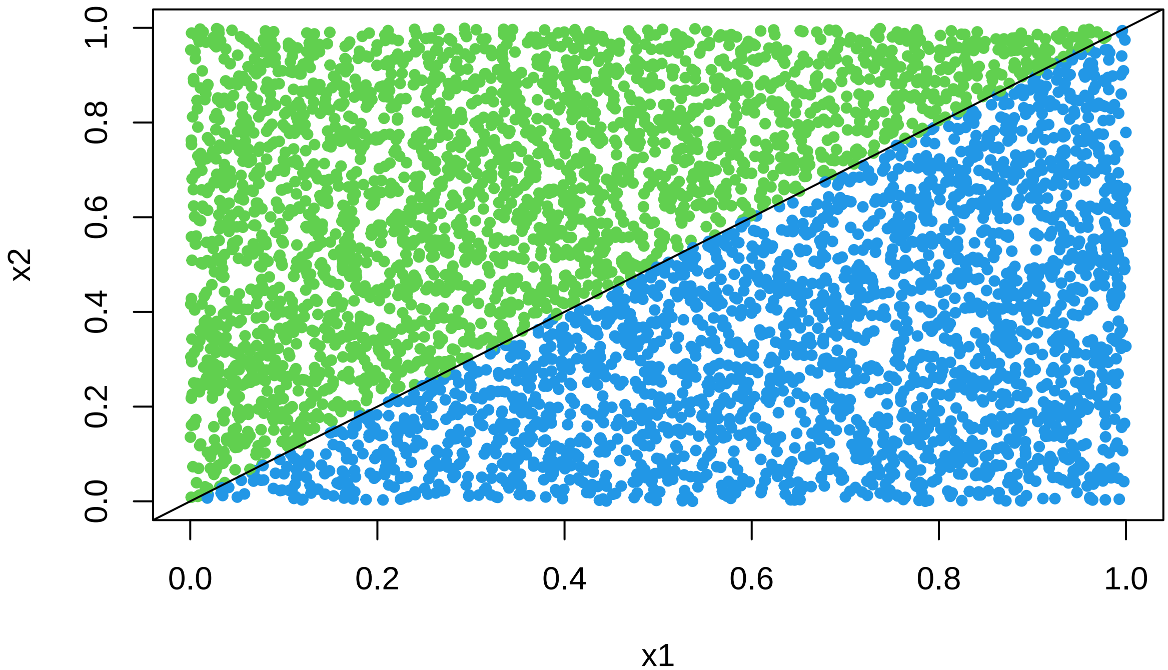
i

SVM gives the best performance for this task, then Logistic Regression and SVC. The speed is almost the same. We can also trying randomly figuring out the underlying predictors (try manipulating the predictors) for the LR model but that might be time-consuming.

Exercise 6

a

```
set.seed(1)
x1 = runif(5000, 0, 1)
#x1_confusing = runif(100, 0, 1)
x2 = runif(5000, 0, 1)
#x2_confusing = x1_confusing

y = x1 - x2 > 0
plot(x1, x2, col = y + 3, pch = 20)
abline(0, 1, col = 1)
```




```r
# X = c(x1, x1_confusing, x2, x2_confusing)

train_set = data.frame(x1, x2, y = as.factor(y))
```

b

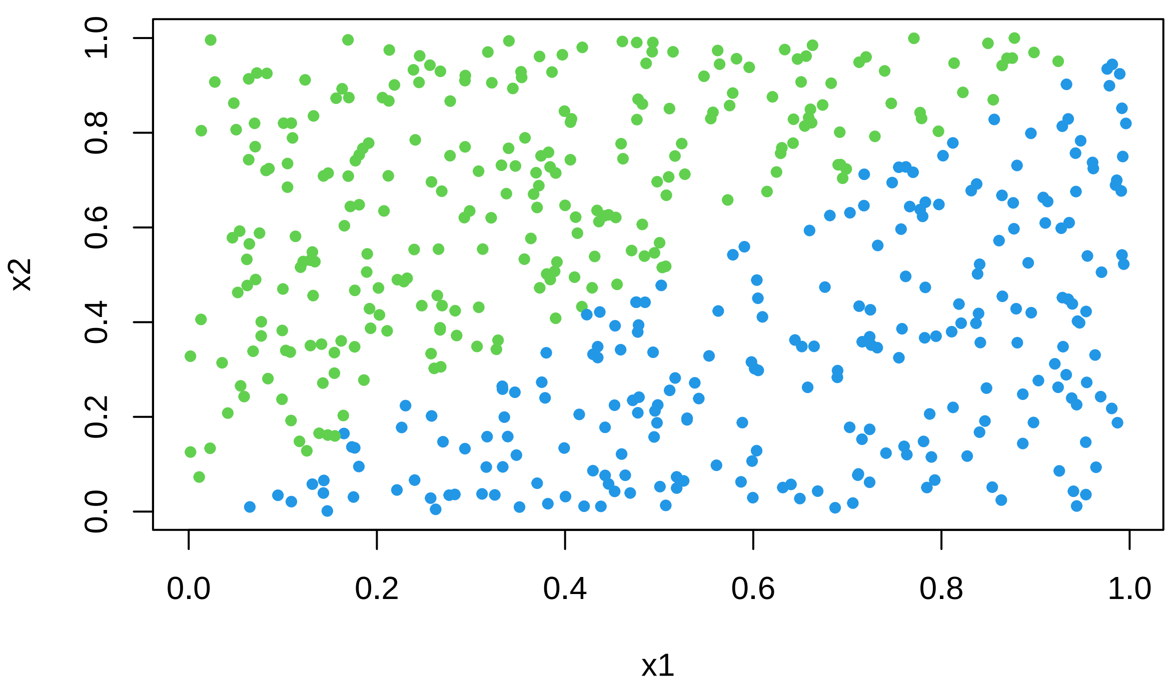
```r
cost_values = c(0.01, 0.1, 1, 10, 100, 1e3, 1e4)
cost_tune = tune(svm, y ~ ., data = train_set, kernel = 'linear',
                 ranges = list(cost = cost_values))
summary(cost_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 2e-04
##
## - Detailed performance results:
```

```
##    cost  error   dispersion
## 1 1e-02 0.0012 0.0016865481
## 2 1e-01 0.0012 0.0013984118
## 3 1e+00 0.0002 0.0006324555
## 4 1e+01 0.0020 0.0023094011
## 5 1e+02 0.0008 0.0010327956
## 6 1e+03 0.0006 0.0009660918
## 7 1e+04 0.0008 0.0010327956
```

c

```r
set.seed(1)
x1 = runif(500, 0, 1)
x2 = runif(500, 0, 1)

y = x1 - x2 > 0
plot(x1, x2, col = y + 3, pch = 20)
```




```r
test_set = data.frame(x1, x2, y = as.factor(y))
```

```r
error_rates = rep(NA, length(cost_values))

for (i in 1:length(cost_values)){
```

```
  cost_value = cost_values[i]
  an_svm_model = svm(y ~ ., data = train_set, kernel = 'linear', cost = cost_value)
  test_pred = predict(an_svm_model, test_set)
  error_rates[i] = sum(test_pred != test_set$y) / nrow(test_set)
}

data.frame(cost = cost_values, error_rates = error_rates)
```

```
##    cost error_rates
## 1 1e-02       0.002
## 2 1e-01       0.002
## 3 1e+00       0.000
## 4 1e+01       0.000
## 5 1e+02       0.000
## 6 1e+03       0.002
## 7 1e+04       0.002
```

d

The selected model using cross-validation might not be perfect. 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.

Exercise 7

a

```
auto = Auto
auto$gas_mileage = (Auto['mpg'] > median(Auto$mpg)) * 1
auto$gas_mileage = as.factor(auto$gas_mileage)
auto = auto[, 2:10]
#View(auto)
```

b

```
cost_values = c(0.1, 1, 10, 100, 1e3, 1e4)
tune_cost = tune(svm, gas_mileage ~ ., data = auto, kernel = 'linear',
                 ranges = list(cost = cost_values))
summary(tune_cost)
```

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

```
##  cost
##   0.1
##
## - best performance: 0.08942308
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-01 0.08942308 0.04889872
## 2 1e+00 0.10198718 0.05097706
## 3 1e+01 0.10705128 0.06351583
## 4 1e+02 0.10961538 0.05105094
## 5 1e+03 0.11217949 0.06494039
## 6 1e+04 0.11474359 0.06273760
```

c

```r
cost_values = c(0.1, 1, 10, 100, 1e3, 1e4)
gamma_values = c(1e-2, 1e-1, 1, 1e1, 1e2)
tune_rad_gam = tune(svm, gas_mileage ~ ., data = auto, kernel = 'radial',
                    ranges = list(cost = cost_values,
                                  gamma = gamma_values))
summary(tune_rad_gam)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     1
##
## - best performance: 0.07391026
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01 1e-02 0.11724359 0.04962645
## 2  1e+00 1e-02 0.08673077 0.04387585
## 3  1e+01 1e-02 0.08923077 0.04214637
## 4  1e+02 1e-02 0.09698718 0.02660930
## 5  1e+03 1e-02 0.10974359 0.02989743
## 6  1e+04 1e-02 0.12000000 0.04035894
## 7  1e-01 1e-01 0.08923077 0.04540163
## 8  1e+00 1e-01 0.08416667 0.04188694
## 9  1e+01 1e-01 0.07397436 0.01450224
## 10 1e+02 1e-01 0.10448718 0.03469180
## 11 1e+03 1e-01 0.10711538 0.03360677
## 12 1e+04 1e-01 0.10711538 0.03360677
## 13 1e-01 1e+00 0.54070513 0.03076812
## 14 1e+00 1e+00 0.07391026 0.04228832
## 15 1e+01 1e+00 0.07391026 0.03693807
## 16 1e+02 1e+00 0.07391026 0.03693807
```

```
## 17 1e+03 1e+00 0.07391026 0.03693807
## 18 1e+04 1e+00 0.07391026 0.03693807
## 19 1e-01 1e+01 0.54070513 0.03076812
## 20 1e+00 1e+01 0.49987179 0.05092898
## 21 1e+01 1e+01 0.49730769 0.05499122
## 22 1e+02 1e+01 0.49730769 0.05499122
## 23 1e+03 1e+01 0.49730769 0.05499122
## 24 1e+04 1e+01 0.49730769 0.05499122
## 25 1e-01 1e+02 0.54070513 0.03076812
## 26 1e+00 1e+02 0.54070513 0.03076812
## 27 1e+01 1e+02 0.54070513 0.03076812
## 28 1e+02 1e+02 0.54070513 0.03076812
## 29 1e+03 1e+02 0.54070513 0.03076812
## 30 1e+04 1e+02 0.54070513 0.03076812
```

```
cost_values = c(0.1, 1, 10, 100, 1e3, 1e4)
degree_values = c(1, 2, 3, 4, 5)
tune_pol_deg = tune(svm, gas_mileage ~ ., data = auto, kernel = 'polynomial',
                    ranges = list(cost = cost_values,
                                  degree = degree_values))
summary(tune_pol_deg)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      1
##
## - best performance: 0.08705128
##
## - Detailed performance results:
##     cost degree      error dispersion
## 1  1e-01      1 0.28576923 0.11269167
## 2  1e+00      1 0.10730769 0.04832576
## 3  1e+01      1 0.08961538 0.06983016
## 4  1e+02      1 0.08705128 0.06317910
## 5  1e+03      1 0.09730769 0.06502527
## 6  1e+04      1 0.12012821 0.05172106
## 7  1e-01      2 0.56910256 0.03113623
## 8  1e+00      2 0.56910256 0.03113623
## 9  1e+01      2 0.55897436 0.03735346
## 10 1e+02      2 0.31397436 0.10024160
## 11 1e+03      2 0.28583333 0.04556491
## 12 1e+04      2 0.16608974 0.08144499
## 13 1e-01      3 0.56910256 0.03113623
## 14 1e+00      3 0.56910256 0.03113623
## 15 1e+01      3 0.56910256 0.03113623
## 16 1e+02      3 0.40294872 0.11585855
## 17 1e+03      3 0.25769231 0.07302630
## 18 1e+04      3 0.16326923 0.09599895
## 19 1e-01      4 0.56910256 0.03113623
```

```
## 20 1e+00      4 0.56910256 0.03113623
## 21 1e+01      4 0.56910256 0.03113623
## 22 1e+02      4 0.56910256 0.03113623
## 23 1e+03      4 0.56910256 0.03113623
## 24 1e+04      4 0.42858974 0.08693205
## 25 1e-01      5 0.56910256 0.03113623
## 26 1e+00      5 0.56910256 0.03113623
## 27 1e+01      5 0.56910256 0.03113623
## 28 1e+02      5 0.56910256 0.03113623
## 29 1e+03      5 0.56910256 0.03113623
## 30 1e+04      5 0.56910256 0.03113623
```

d

```
SVC_linear_model = svm(gas_mileage ~ ., data = auto,
                       kernel = 'linear', cost = 0.1)
SVM_radial_model = svm(gas_mileage ~ ., data = auto,
                       kernel = 'radial', cost = 1, gamma = 1)
SVM_polynomial_model = svm(gas_mileage ~ ., data = auto,
                           kernel = 'polynomial', cost = 100, degree = 1)
```

```
table(SVC_linear_model$fitted, auto$gas_mileage)
```

```
##
##       0   1
##   0 172   7
##   1  24 189
```

```
table(SVM_radial_model$fitted, auto$gas_mileage)
```

```
##
##       0   1
##   0 195   2
##   1   1 194
```

```
table(SVM_polynomial_model$fitted, auto$gas_mileage)
```

```
##
##       0   1
##   0 179   5
##   1  17 191
```

Exercise 8

a

```r
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

```r
set.seed(42)
train_indices = sample(1070, 800)
train_set = OJ[train_indices, ]
test_set = OJ[-train_indices, ]
```

b

```r
SVC_model = svm(Purchase ~ ., data = train_set, kernel = 'linear', cost = 0.01)
summary(SVC_model)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_set, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
```

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

c

```r
test_pred = predict(SVC_model, test_set)
table(SVC_model$fitted, train_set$Purchase)
```

```
##
##       CH  MM
##   CH 432  77
##   MM  60 231
```

```r
table(test_pred, test_set$Purchase)
```

```
##
## test_pred  CH  MM
##         CH 142  25
##         MM  19  84
```

```r
sum(SVC_model$fitted == train_set$Purchase) / nrow(train_set)
```

```
## [1] 0.82875
```

```r
sum(test_pred == test_set$Purchase) / nrow(test_set)
```

```
## [1] 0.837037
```

d

```r
cost_values = c(0.01, 0.1, 0.5, 1, 5, 10)
cost_tune = tune(svm, Purchase ~ ., data = train_set, kernel = 'linear',
                 ranges = list(cost = cost_values))
summary(cost_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1775
```

```
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.18250 0.04133199
## 2  0.10 0.18000 0.04901814
## 3  0.50 0.18125 0.04050463
## 4  1.00 0.17750 0.04031129
## 5  5.00 0.17875 0.03821086
## 6 10.00 0.18375 0.03438447
```

e

```r
SVC_linear_model = svm(Purchase ~ ., data = train_set, kernel = 'linear', cost = 1)

test_pred = predict(SVC_linear_model, test_set)
table(SVC_linear_model$fitted, train_set$Purchase)
```

```
##
##       CH  MM
##   CH 434  76
##   MM  58 232
```

```r
table(test_pred, test_set$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 140  23
##        MM  21  86
```

```r
sum(SVC_linear_model$fitted == train_set$Purchase) / nrow(train_set)
```

```
## [1] 0.8325
```

```r
sum(test_pred == test_set$Purchase) / nrow(test_set)
```

```
## [1] 0.837037
```

f

```r
gamma_values = c(1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3)
gamma_tune = tune(svm, Purchase ~ ., data = train_set, kernel = 'radial',
                  ranges = list(gamma = gamma_values))
summary(gamma_tune)
```

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

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##   0.01
##
## - best performance: 0.1725
##
## - Detailed performance results:
##   gamma   error dispersion
## 1 1e-03 0.19250 0.05143766
## 2 1e-02 0.17250 0.05263871
## 3 1e-01 0.18375 0.04084609
## 4 1e+00 0.20750 0.04571956
## 5 1e+01 0.24500 0.05041494
## 6 1e+02 0.29500 0.05109903
## 7 1e+03 0.35625 0.06325928
```

```
SVM_radial_model = svm(Purchase ~ ., data = train_set, kernel = 'radial',
                       gamma = gamma_tune$best.parameters$gamma)

test_pred = predict(SVM_radial_model, test_set)
table(SVM_radial_model$fitted, train_set$Purchase)
```

```
##
##       CH  MM
##   CH 441  78
##   MM  51 230
```

```
table(test_pred, test_set$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 142  23
##        MM  19  86
```

```
sum(SVM_radial_model$fitted == train_set$Purchase) / nrow(train_set)
```

```
## [1] 0.83875
```

```
sum(test_pred == test_set$Purchase) / nrow(test_set)
```

```
## [1] 0.8444444
```

g

```
degree_values = c(0.5, 1, 2, 3, 4, 5)
cost_values = c(0.01, 0.1, 0.5, 1, 5, 10)
degree_tune = tune(svm, Purchase ~ ., data = train_set, kernel = 'polynomial',
```

```r
                  ranges = list(degree = degree_values,
                                cost = cost_values))
summary(degree_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree cost
##       1  0.1
##
## - best performance: 0.175
##
## - Detailed performance results:
##    degree  cost   error dispersion
## 1     0.5  0.01 0.38500 0.06146363
## 2     1.0  0.01 0.38500 0.06146363
## 3     2.0  0.01 0.38500 0.06146363
## 4     3.0  0.01 0.36625 0.05591723
## 5     4.0  0.01 0.36625 0.05591723
## 6     5.0  0.01 0.36750 0.05627314
## 7     0.5  0.10 0.38500 0.06146363
## 8     1.0  0.10 0.17500 0.05803495
## 9     2.0  0.10 0.31125 0.05447030
## 10    3.0  0.10 0.29375 0.05750906
## 11    4.0  0.10 0.31000 0.05458174
## 12    5.0  0.10 0.31000 0.05197489
## 13    0.5  0.50 0.38500 0.06146363
## 14    1.0  0.50 0.17750 0.05263871
## 15    2.0  0.50 0.20375 0.04450733
## 16    3.0  0.50 0.19750 0.05263871
## 17    4.0  0.50 0.24375 0.05566829
## 18    5.0  0.50 0.27000 0.05374838
## 19    0.5  1.00 0.38500 0.06146363
## 20    1.0  1.00 0.18125 0.05628857
## 21    2.0  1.00 0.19875 0.05152197
## 22    3.0  1.00 0.19500 0.05502525
## 23    4.0  1.00 0.21875 0.04759858
## 24    5.0  1.00 0.24125 0.04788949
## 25    0.5  5.00 0.38500 0.06146363
## 26    1.0  5.00 0.18125 0.05245699
## 27    2.0  5.00 0.18500 0.04362084
## 28    3.0  5.00 0.18625 0.06520534
## 29    4.0  5.00 0.20625 0.04649149
## 30    5.0  5.00 0.21125 0.05573063
## 31    0.5 10.00 0.38500 0.06146363
## 32    1.0 10.00 0.18375 0.05466120
## 33    2.0 10.00 0.18625 0.04226652
## 34    3.0 10.00 0.18875 0.06520534
## 35    4.0 10.00 0.20250 0.04993051
## 36    5.0 10.00 0.21000 0.04706674
```

```r
SVM_polynomial_model = svm(Purchase ~ ., data = train_set, kernel = 'polynomial',
                       degree = degree_tune$best.parameters$degree,
                       cost = degree_tune$best.parameters$cost)

test_pred = predict(SVM_polynomial_model, test_set)
table(SVM_polynomial_model$fitted, train_set$Purchase)
```

```
##
##       CH  MM
##   CH 434  80
##   MM  58 228
```

```r
table(test_pred, test_set$Purchase)
```

```
##
## test_pred  CH  MM
##        CH 142  26
##        MM  19  83
```

```r
sum(SVM_polynomial_model$fitted == train_set$Purchase) / nrow(train_set)
```

```
## [1] 0.8275
```

```r
sum(test_pred == test_set$Purchase) / nrow(test_set)
```

```
## [1] 0.8333333
```

h

Overall, for this data set, the SVM with a radial performed the best (although the results among those three approaches are not significantly different).