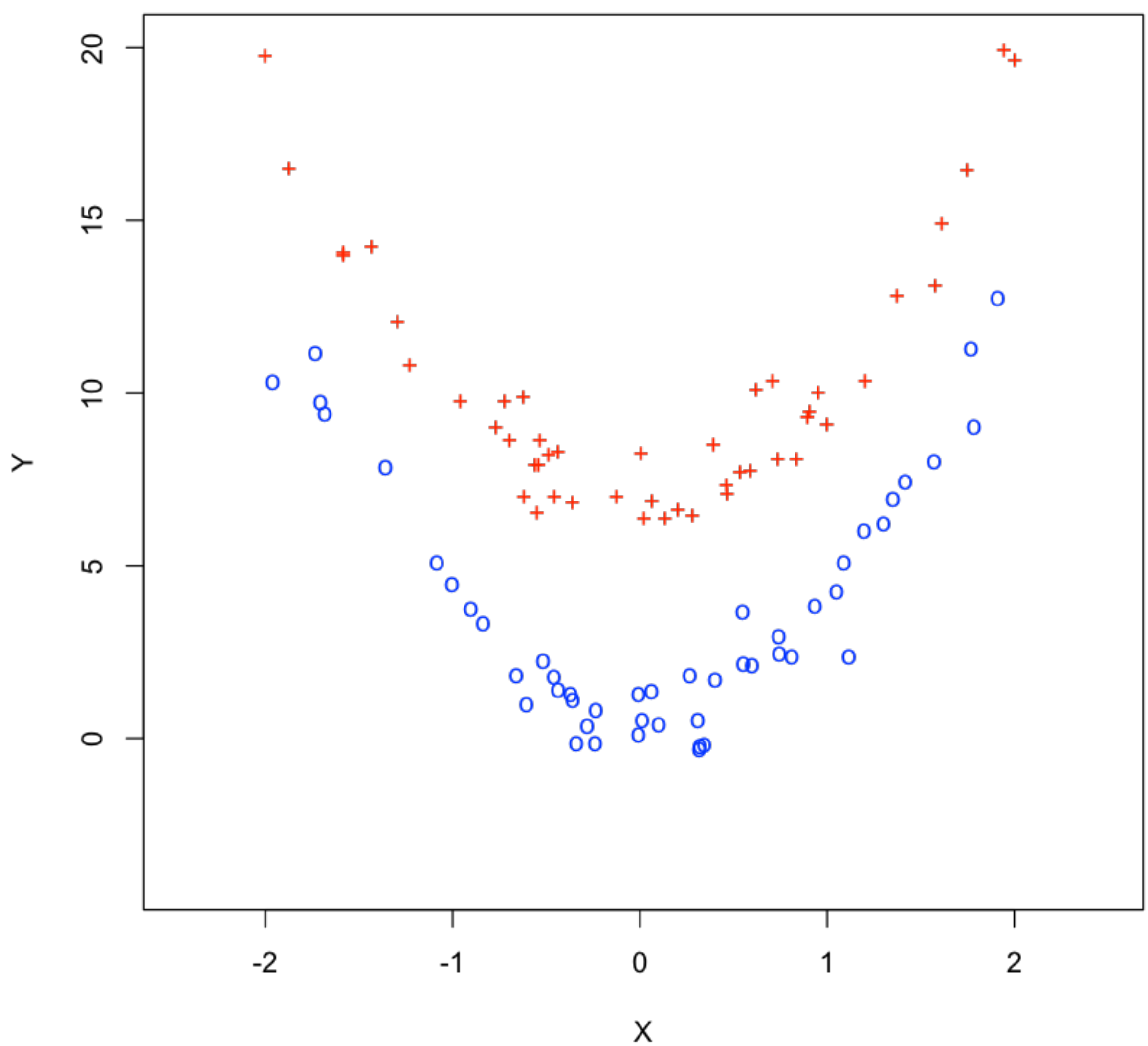
#4

In [1]:

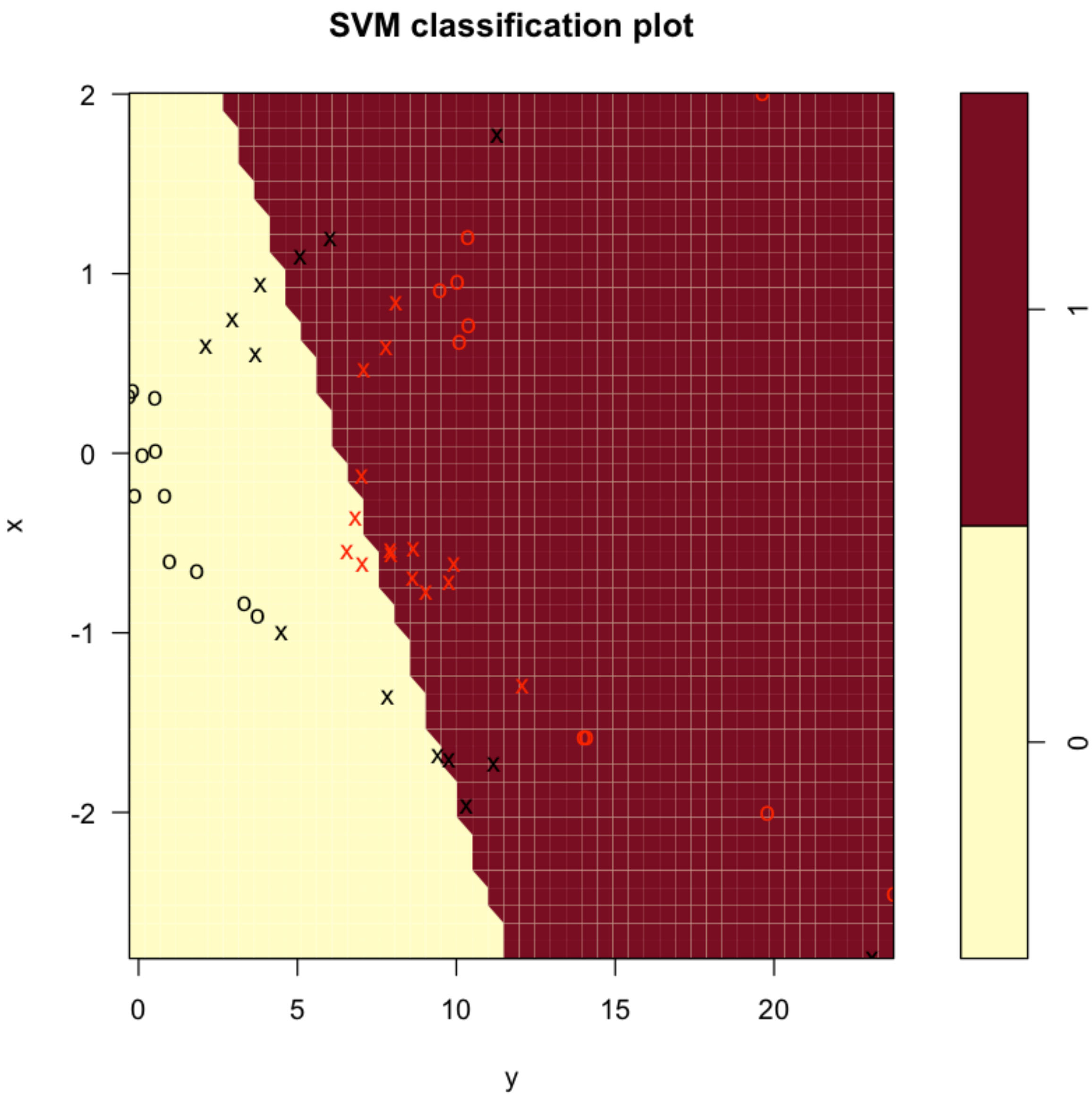
```
set.seed(113)
x = rnorm(100)
y = 3 * x^2 + 4 + rnorm(100)
train = sample(100, 50)
y[train] = y[train] + 3
y[-train] = y[-train] - 3

plot(x[train], y[train], pch="+", lwd=4, col="red",
     ylim=c(-4, 20), xlab="X", ylab="Y")
points(x[-train], y[-train], pch="o", lwd=4, col="blue")
```



In [2]:

```
#plot is non-linearly seperable,
#creating train and test dfs by taking half of +ve and -ve classes
#to create a new vector

library(e1071)
set.seed(113)
z = rep(0, 100)
z[train] = 1
# Take 25 observations each from train and -train
final.train = c(sample(train, 25), sample(setdiff(1:100, train), 2
data.train = data.frame(x=x[final.train], y=y[final.train],
                        z=as.factor(z[final.train]))
data.test = data.frame(x=x[-final.train], y=y[-final.train],
                       z=as.factor(z[-final.train]))
```

In [3]:
```
svm.linear = svm(z~., data=data.train, kernel="linear", cost=10)
plot(svm.linear, data.train)
```

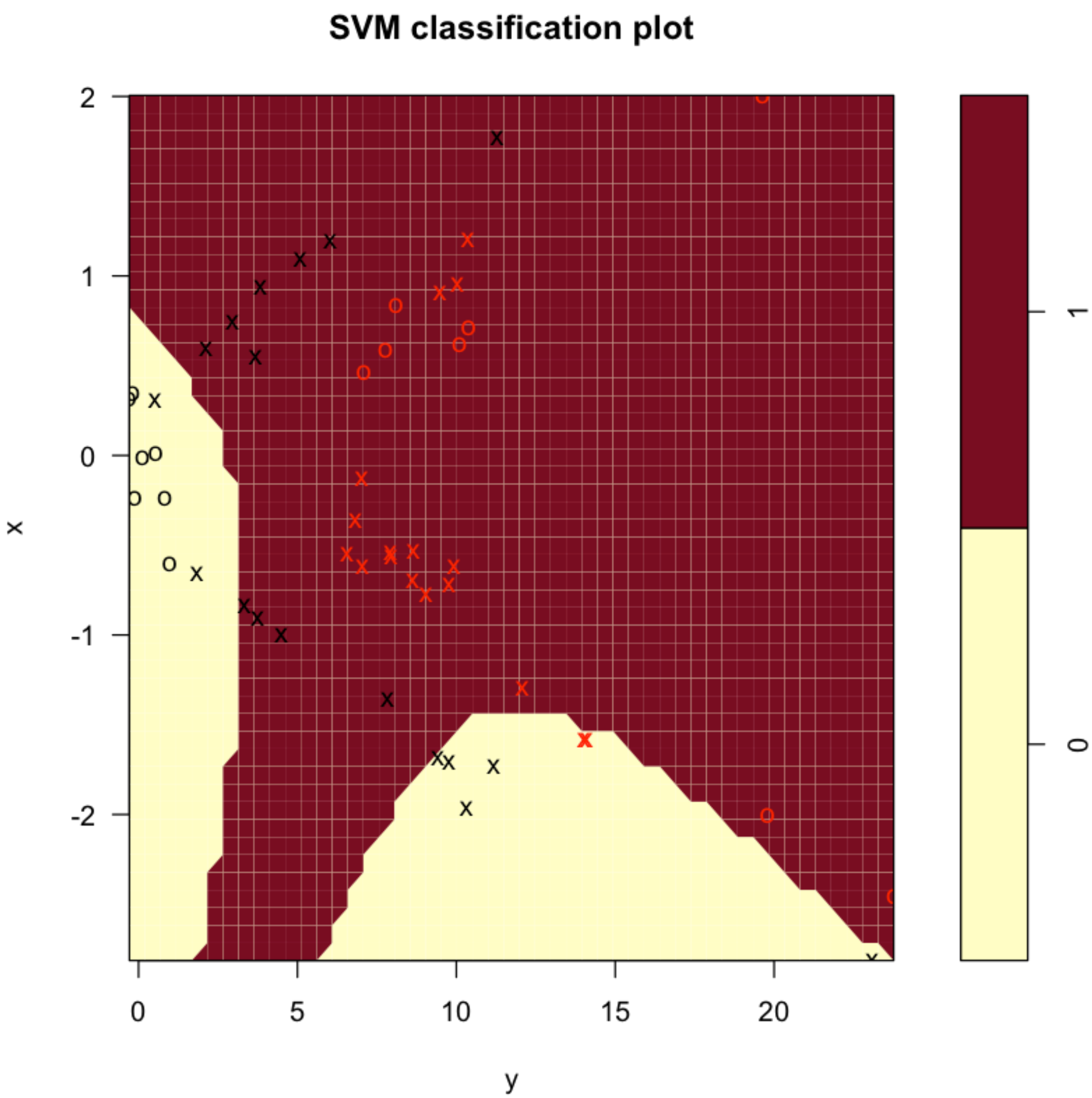


In [4]:
```
#plot shows linear boundry
```

In [5]:
```
table(z[final.train], predict(svm.linear, data.train))
```

```
     0  1
  0 17  8
  1  2 23
```

In [6]:
```
set.seed(32325)
svm.poly = svm(z~., data=data.train, kernel="polynomial", cost=10)
plot(svm.poly, data.train)
```

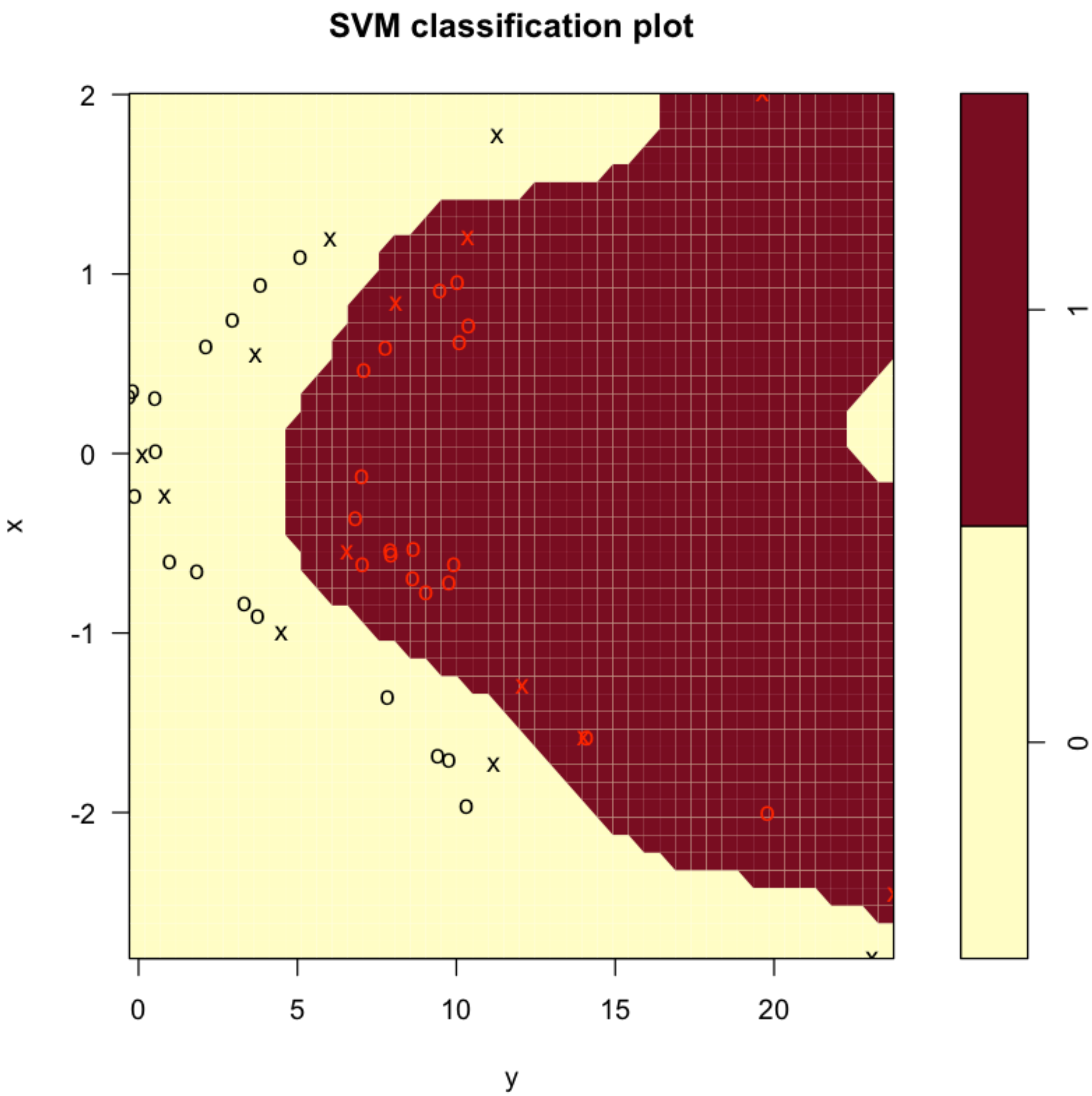


In [7]:
```
table(z[final.train], predict(svm.poly, data.train))
```

```
    0  1
0  13 12
1   2 23
```

In [8]:
```
#This is a default polynomial kernel with degree 3.
```

In [10]:
```
set.seed(353)
svm.radial = svm(z~., data=data.train, kernel="radial", gamma=1,
                 cost=10)
plot(svm.radial, data.train)
```

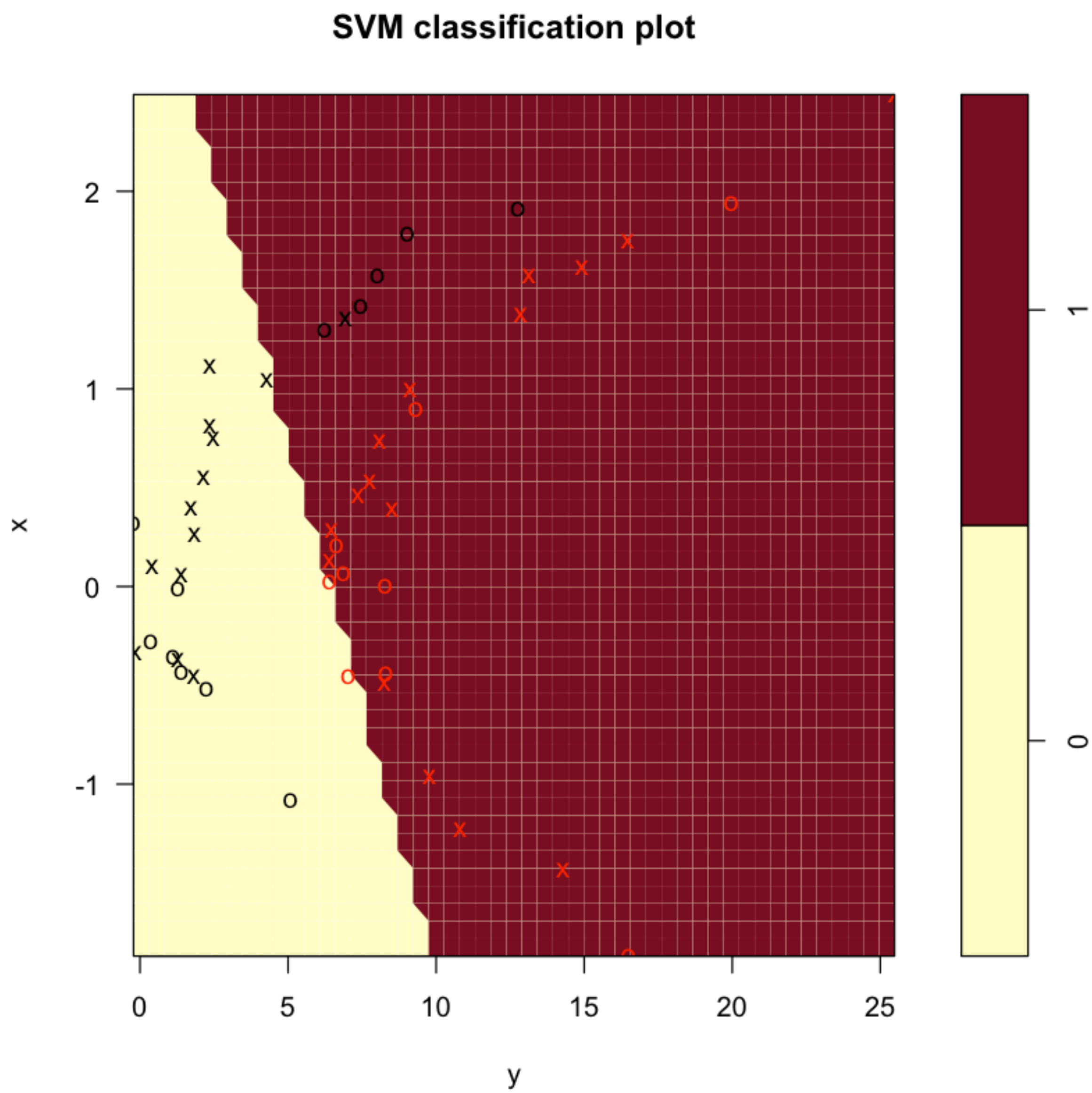


In [11]:
```
table(z[final.train], predict(svm.radial, data.train))
```

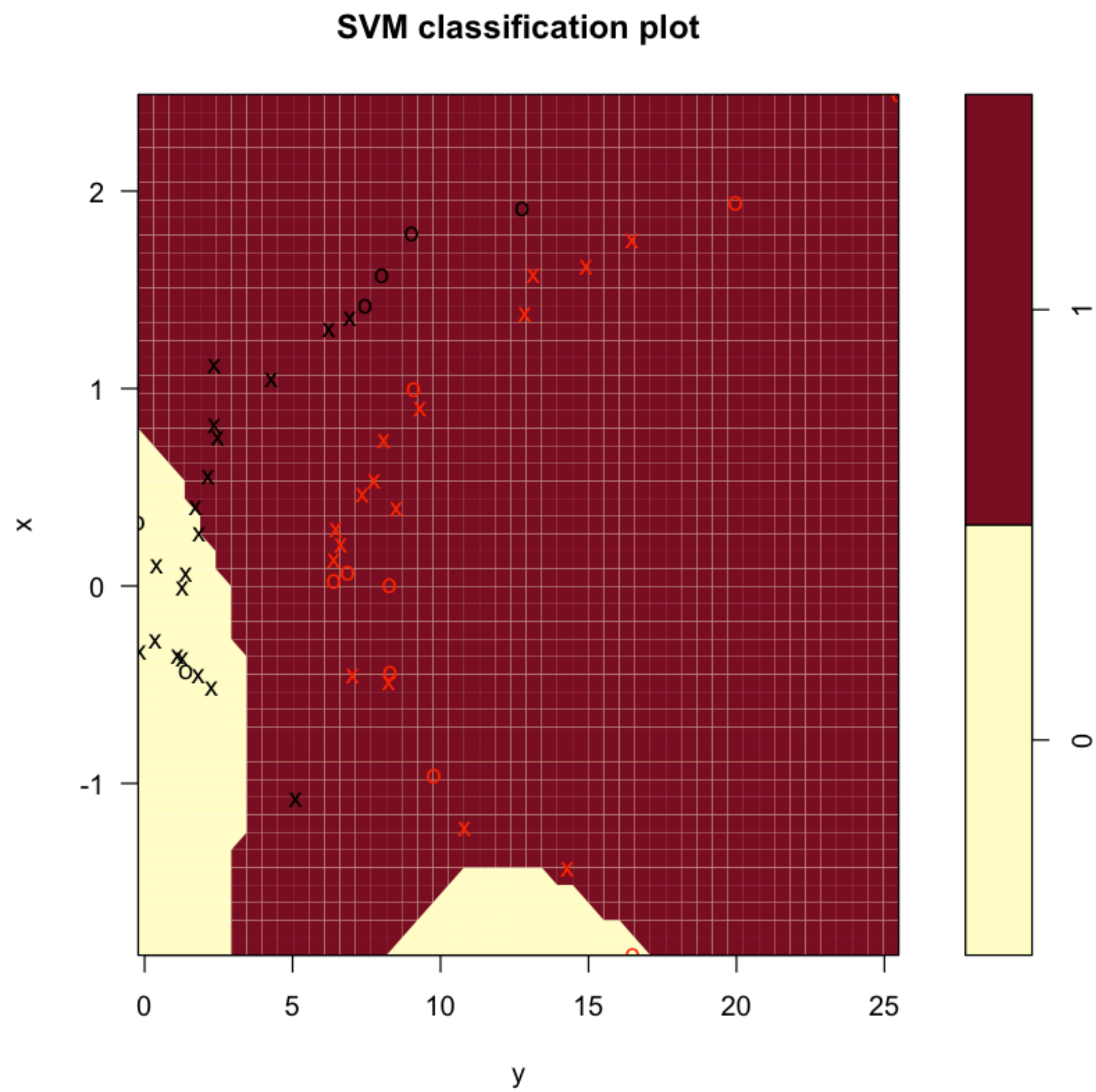
```
   0  1
0 25  0
1  0 25
```

In [12]:
```
#this classifier perfectly classifies train data.
```

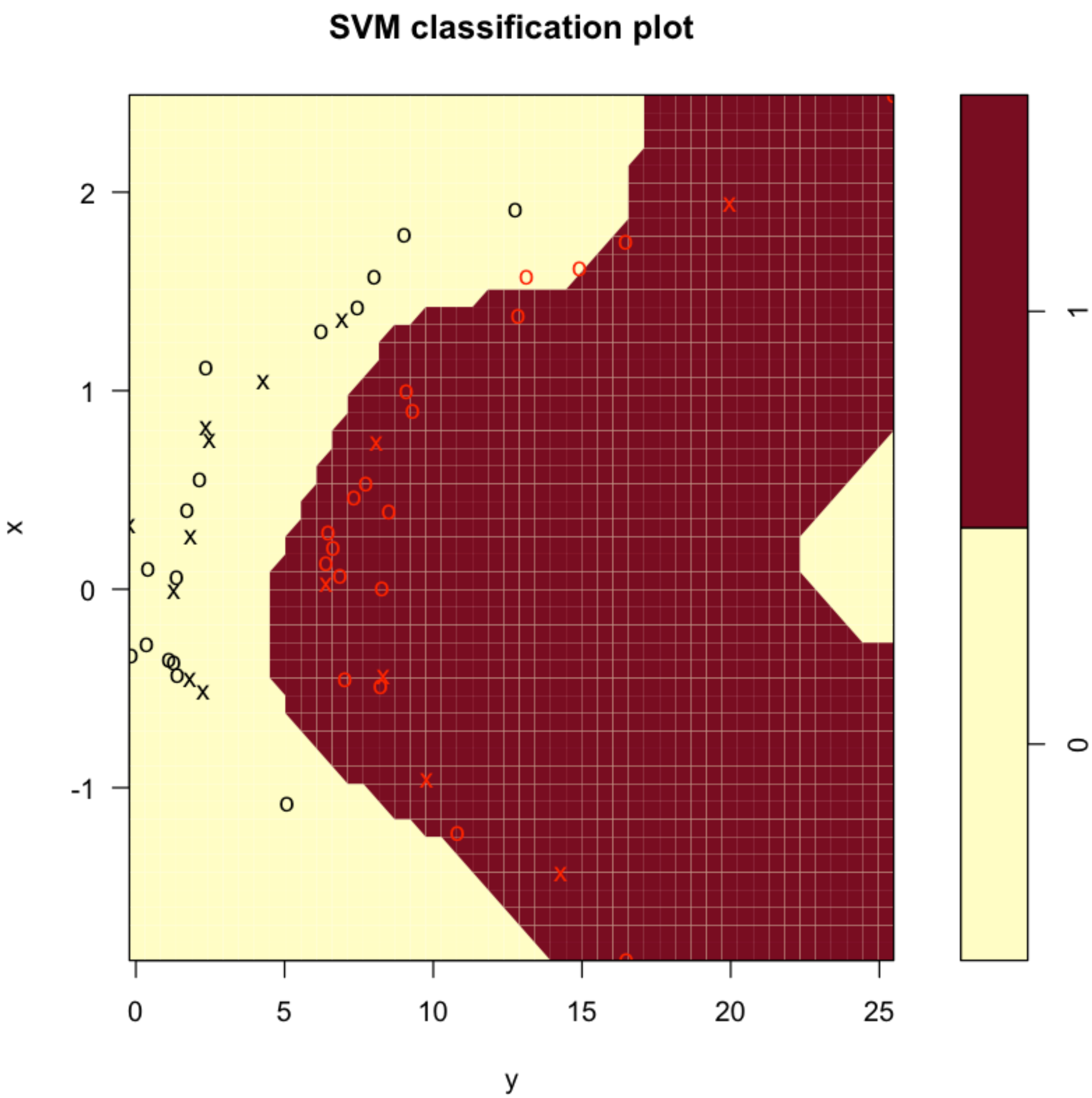
In [13]:
```
plot(svm.linear, data.test)
```



In [14]:
```
plot(svm.poly, data.test)
```



In [15]:
```
plot(svm.radial, data.test)
```



In [16]:
```
table(z[-final.train], predict(svm.linear, data.test))
```

```
    0  1
0 18  7
1  0 25
```

In [17]:
```
table(z[-final.train], predict(svm.poly, data.test))
```

```
    0  1
0 11 14
1  1 24
```

In [18]:
```
table(z[-final.train], predict(svm.radial, data.test))
```

```
    0  1
0 25  0
1  1 24
```

In [19]:
```
#linear, polynomial and radial basis kernels classify
#7, 15, and 1 test points incorrectly respectively.
#Radial kernel is the best and has less test misclassification err
#than linear and polynomial.
```

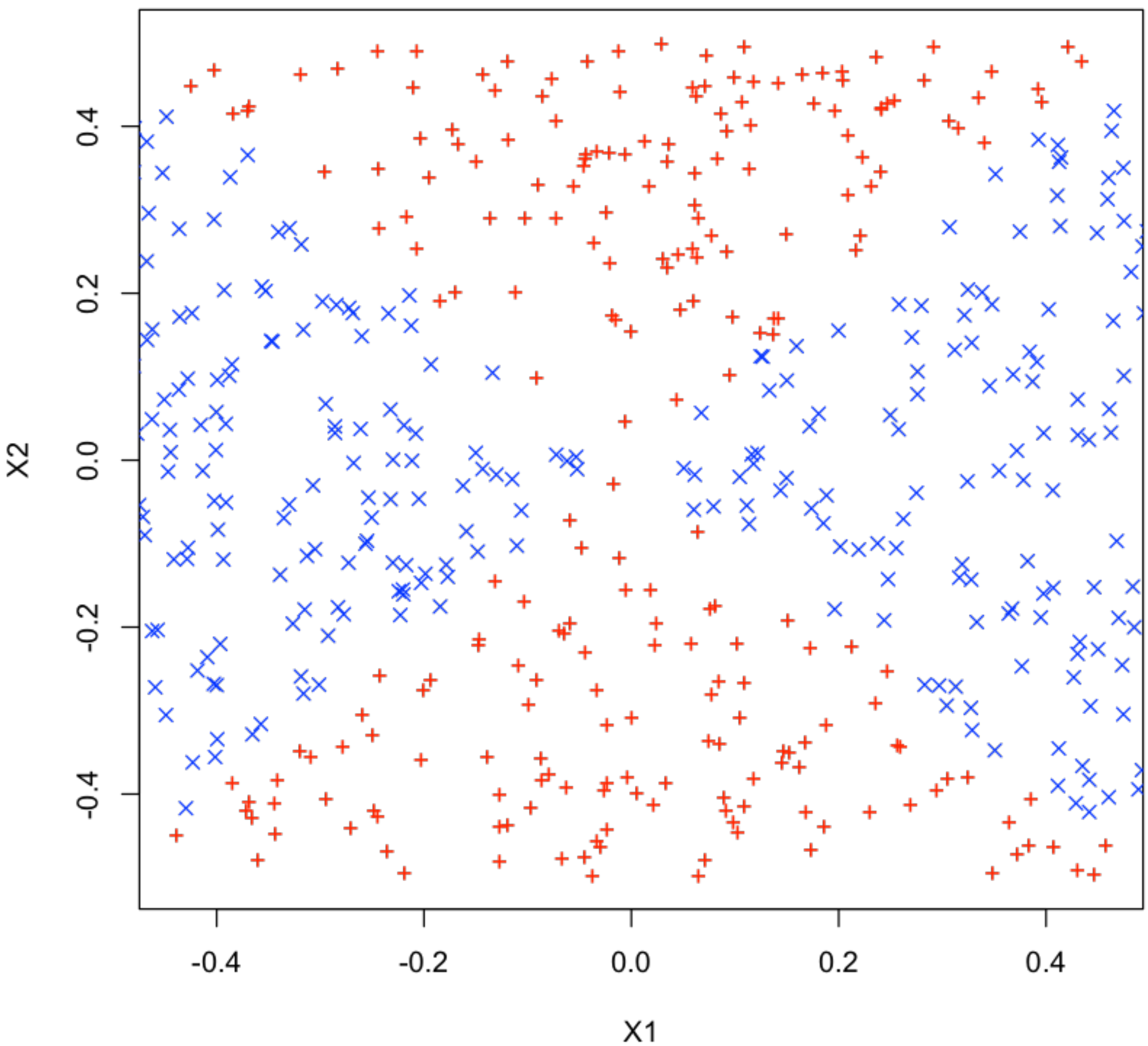
#5

#a

In [20]:
```
set.seed(333)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
```

#b

In [21]:
```
plot(x1[y == 0], x2[y == 0], col = "red", xlab = "X1",
     ylab = "X2", pch = "+")
points(x1[y == 1], x2[y == 1], col = "blue", pch = 4)
```



In [22]:
```
#plot show non-linear decision boundry
```

#c

In [23]:

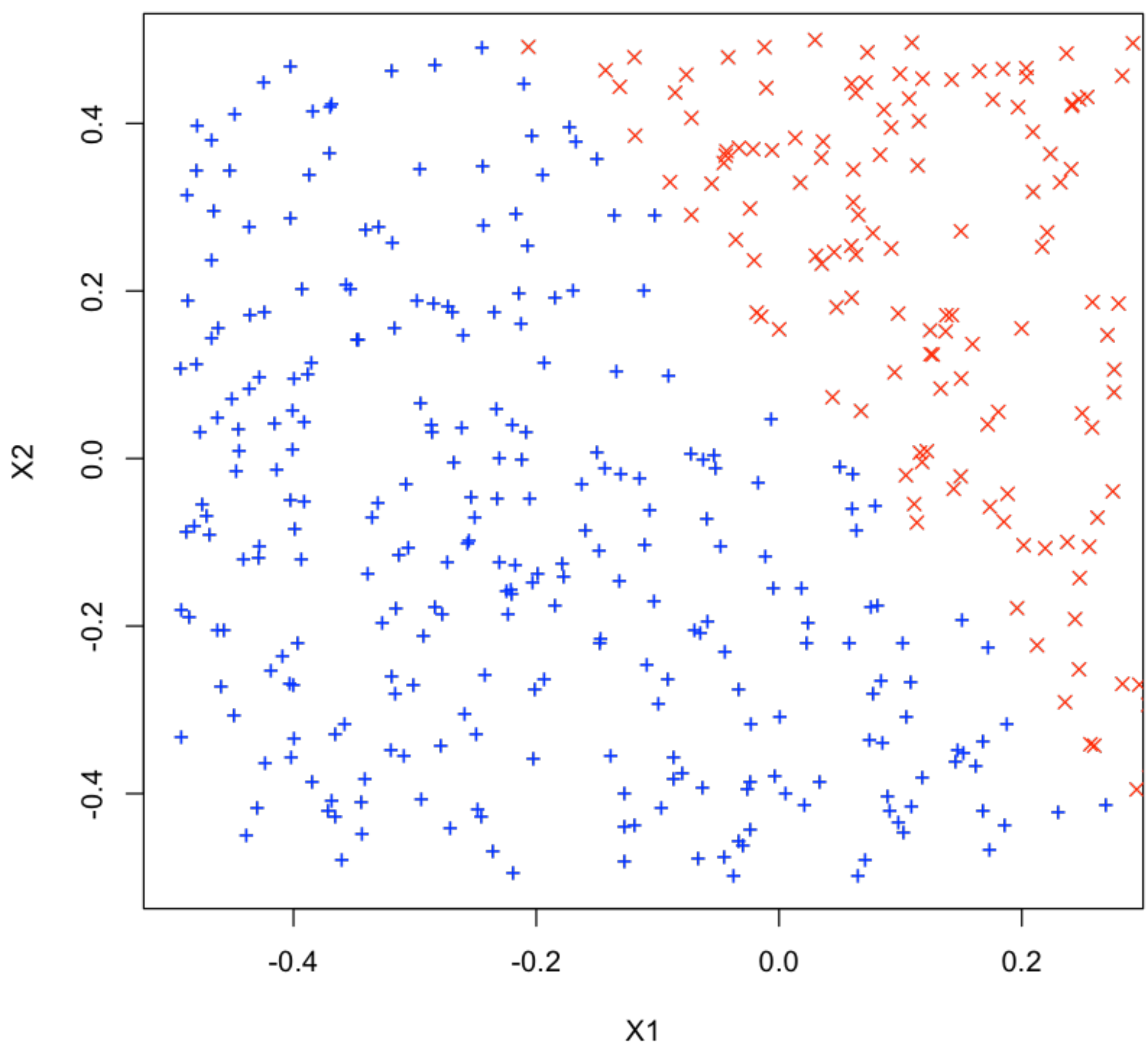
```
lm.fit = glm(y~x1 + x2, family = binomial)
summary(lm.fit)
```

```
Call:
glm(formula = y ~ x1 + x2, family = binomial)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.341  -1.212   1.044   1.124   1.253

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.10321    0.08976   1.150    0.250
x1          -0.40262    0.30981  -1.300    0.194
x2          -0.21758    0.30929  -0.703    0.482

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 691.79  on 499  degrees of freedom
Residual deviance: 689.57  on 497  degrees of freedom
AIC: 695.57

Number of Fisher Scoring iterations: 3
```

#d

In [24]:

```
data = data.frame(x1 = x1, x2 = x2, y = y)
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.52, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1",
     ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```



In [25]:

```
#no decision boundry is shown in the plot. all points are are clas
```

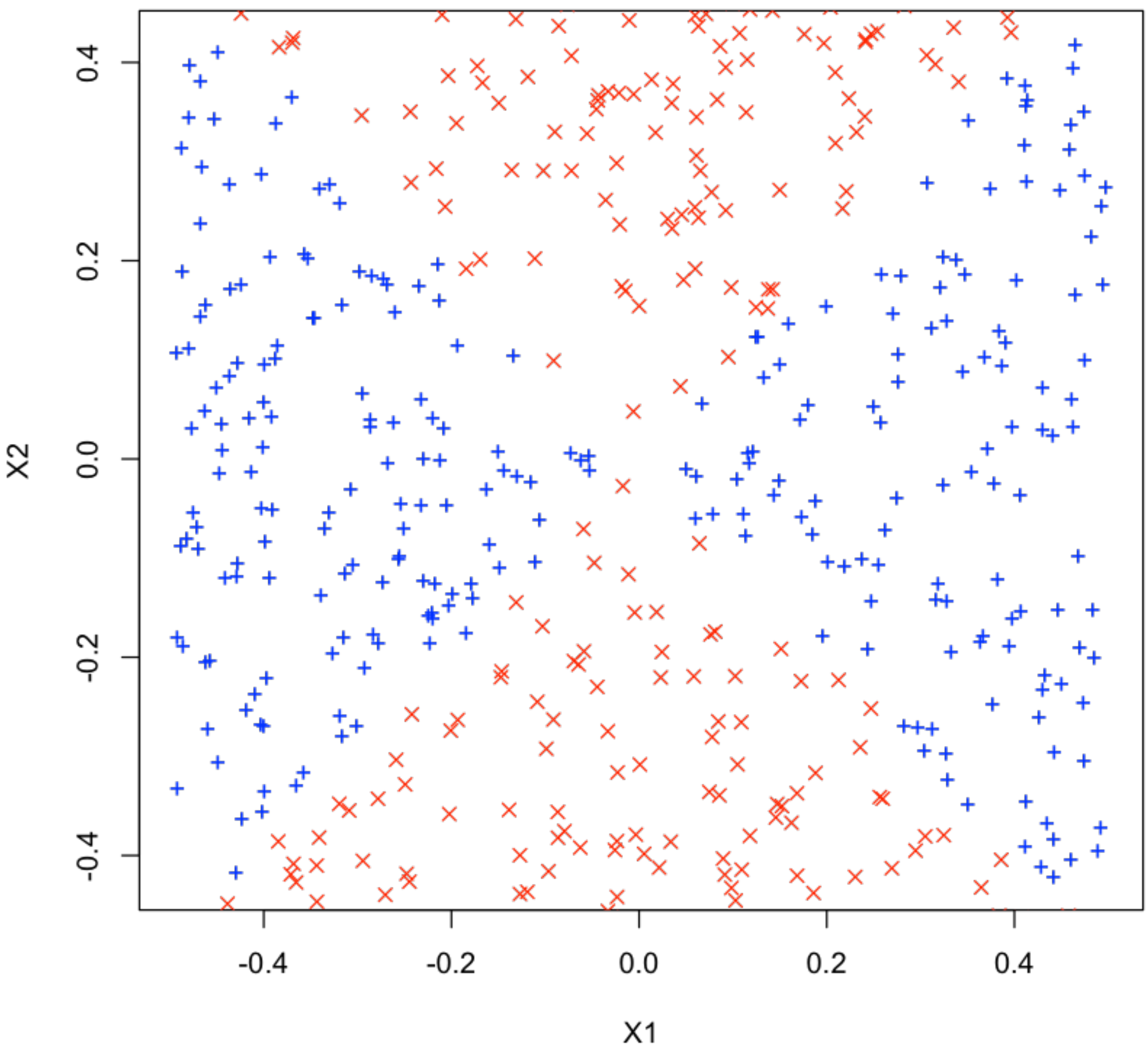
#e

In [26]:
```
lm.fit = glm(y~poly(x1, 2) + poly(x2, 2) + I(x1 * x2),
                data = data, family = binomial)
```

```
Warning message:
“glm.fit: algorithm did not converge”Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
```

#f

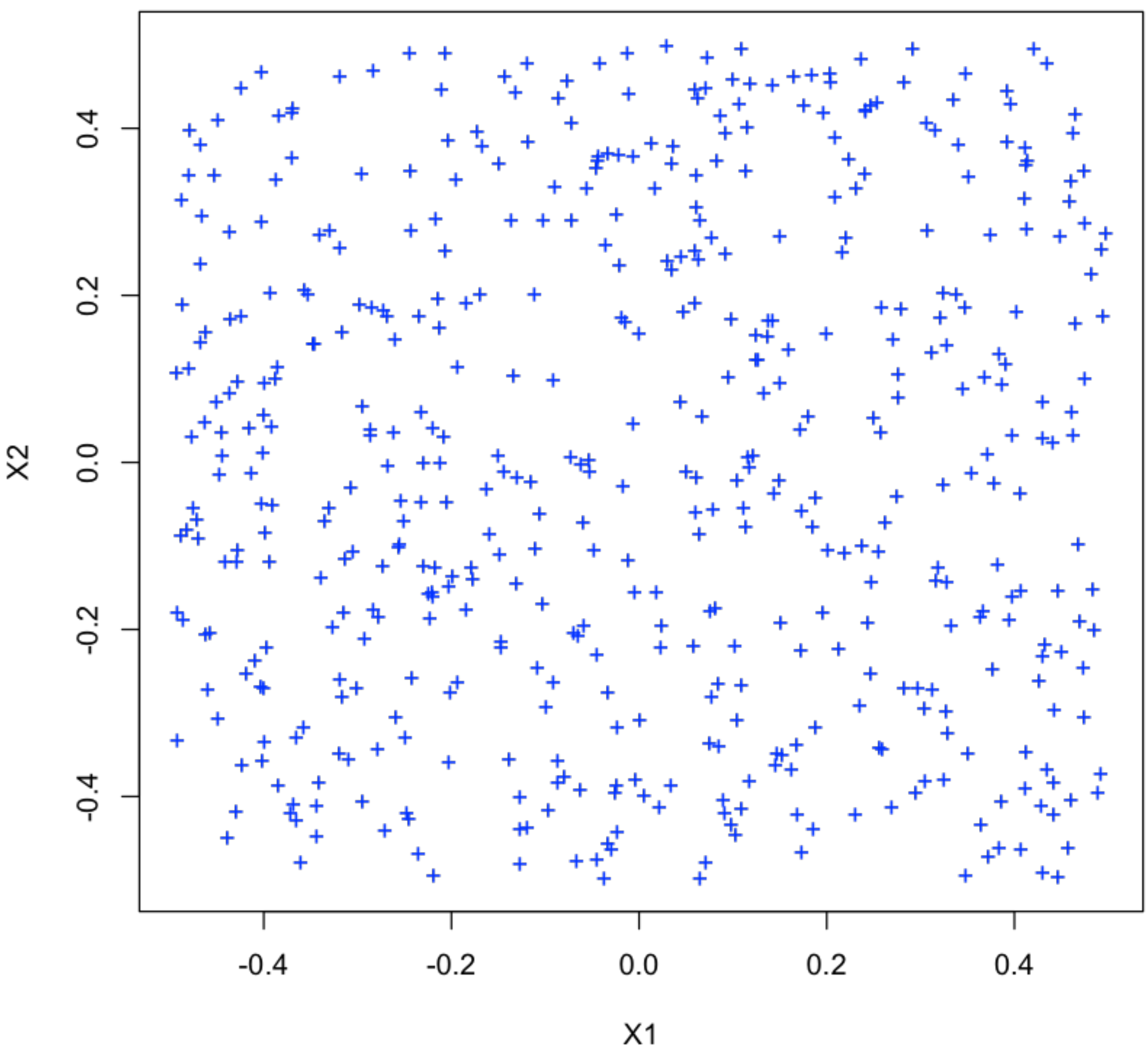
In [27]:
```
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.5, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1",
     ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```



In [28]:
```
#non-linear decision boundary very similar to true decision bounda
```

#g

In [29]:
```
svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear",
              cost = 0.1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1",
     ylab = "X2",
     pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```

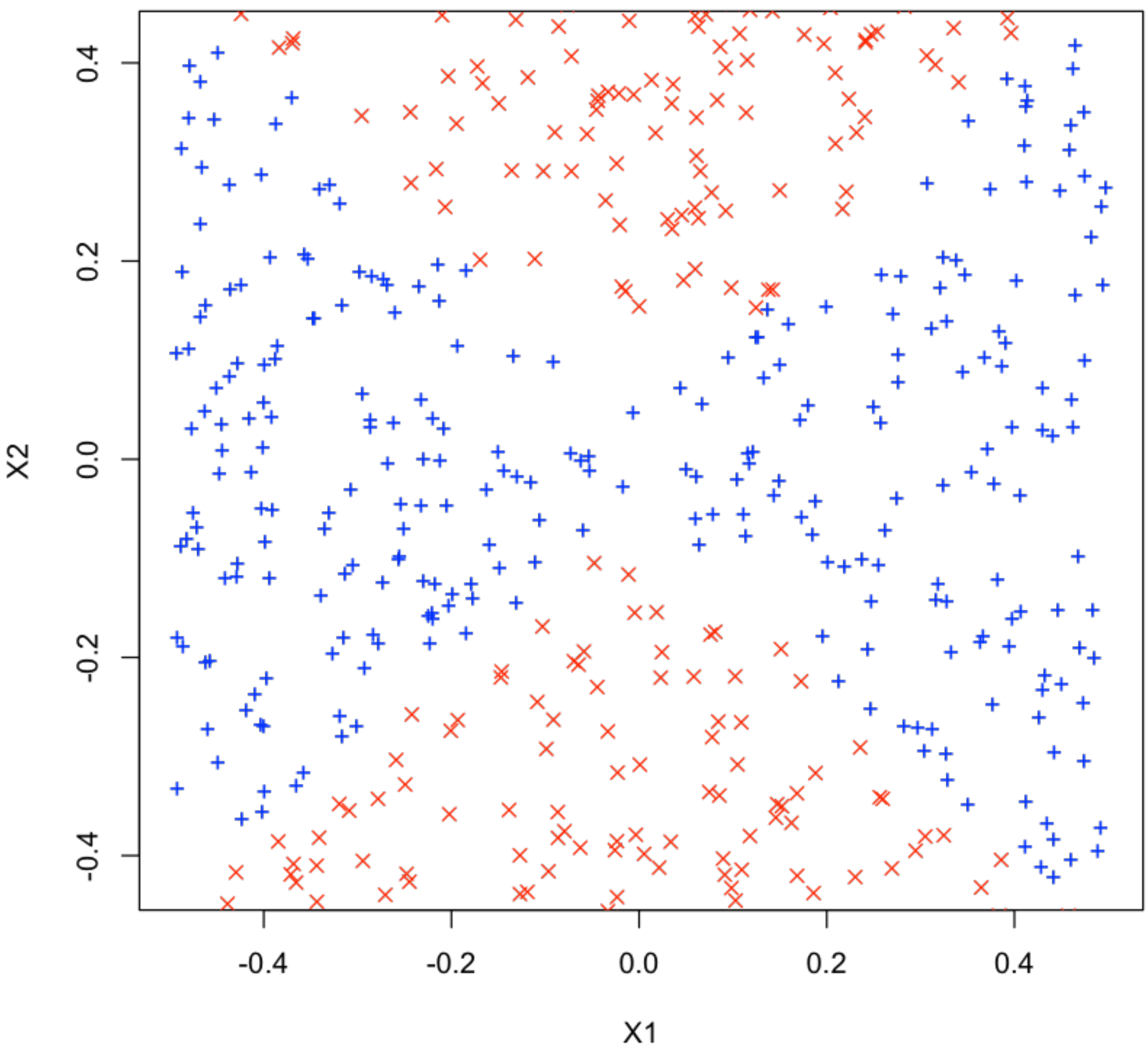


In [30]:
```
# linear kernel, even with low cost fails to find
#non-linear decision boundary
#and classifies all points to a single class.
```

#h

In [31]:

```
svm.fit = svm(as.factor(y)~x1 + x2, data, gamma = 1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1",
     ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```



In [32]:

```
#non-linear decision boundary on predicted labels similar
#true decision boundary.
```

In [33]:
```
#from above problem, SVMs of non-linear kernel are extremely power
#in finding non-linear boundary.logistic regression with non-inter
#and SVMs with linear kernels fail to find the decision boundary.
#Adding interaction terms to logistic regression gives same power
#radial-basis kernels. However, there is some manual efforts and
#tuning involved in picking right interaction terms. This effort c
#prohibitive with large number of features. Radial basis kernels,
#on the other hand, only require tuning of one parameter - gamma -
#which can be easily done using cross-validation.
```

#7

In [34]:
```
library(ISLR)
gas.med = median(Auto$mpg)
new.var = ifelse(Auto$mpg > gas.med, 1, 0)
Auto$mpglevel = as.factor(new.var)
```

In [35]:
```
library(e1071)
```

In [36]:
```
set.seed(3255)
tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear",
                ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.01269231

- Detailed performance results:
   cost      error dispersion
1 1e-02 0.07397436 0.06863413
2 1e-01 0.05102564 0.06923024
3 1e+00 0.01269231 0.02154160
4 5e+00 0.01519231 0.01760469
5 1e+01 0.02025641 0.02303772
6 1e+02 0.03294872 0.02898463
```

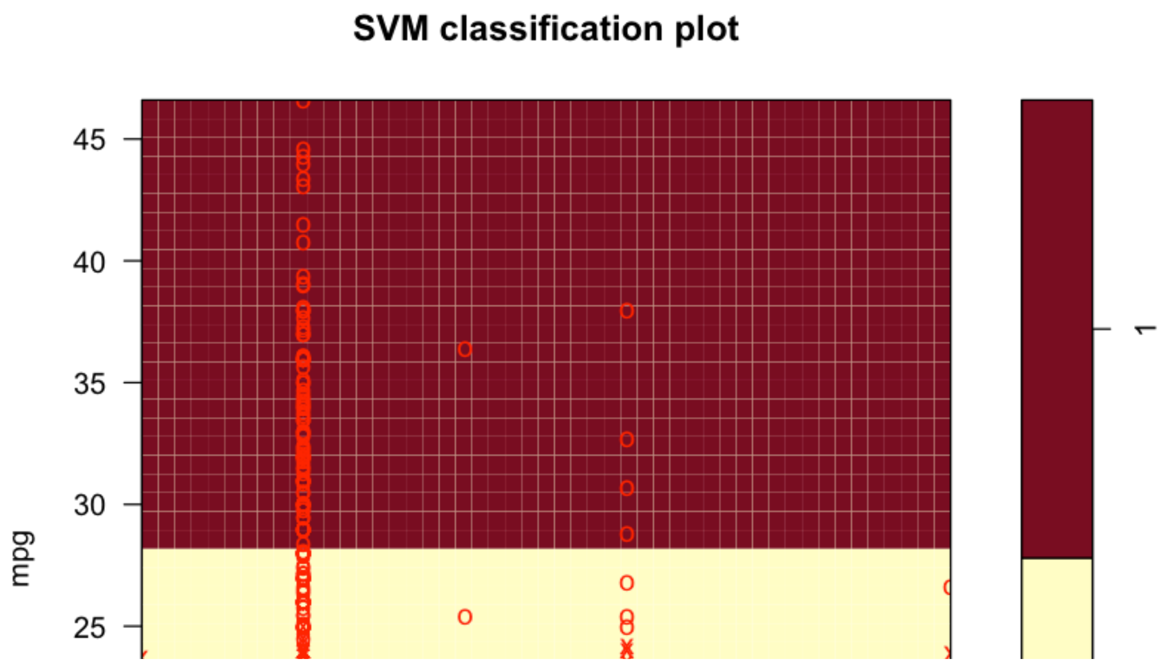
In [37]:
```
set.seed(21)
tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomi
                ranges = list(cost = c(0.1,
    1, 5, 10), degree = c(2, 3, 4)))
summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      2

- best performance: 0.5435897

- Detailed performance results:
   cost degree     error dispersion
1   0.1      2 0.5587821 0.04538579
2   1.0      2 0.5587821 0.04538579
3   5.0      2 0.5587821 0.04538579
4  10.0      2 0.5435897 0.05611162
5   0.1      3 0.5587821 0.04538579
6   1.0      3 0.5587821 0.04538579
7   5.0      3 0.5587821 0.04538579
8  10.0      3 0.5587821 0.04538579
9   0.1      4 0.5587821 0.04538579
10  1.0      4 0.5587821 0.04538579
11  5.0      4 0.5587821 0.04538579
12 10.0      4 0.5587821 0.04538579
```

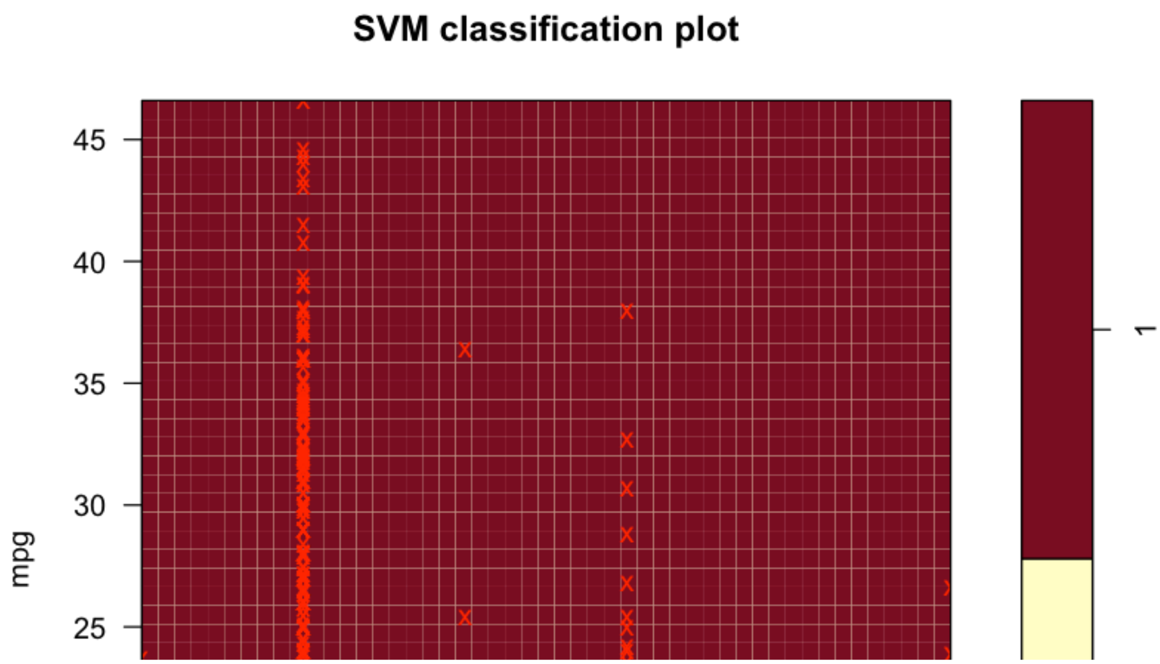
In [38]:
```
set.seed(333)
tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial",
                ranges = list(cost = c(0.1,
    1, 5, 10), gamma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10  0.01

- best performance: 0.02301282

- Detailed performance results:
   cost gamma      error dispersion
1   0.1 1e-02 0.08647436 0.05202669
2   1.0 1e-02 0.07397436 0.03896185
3   5.0 1e-02 0.04346154 0.03829236
4  10.0 1e-02 0.02301282 0.03298698
5   0.1 1e-01 0.07647436 0.04153447
6   1.0 1e-01 0.04858974 0.03714976
7   5.0 1e-01 0.02820513 0.04090081
8  10.0 1e-01 0.02564103 0.02702801
9   0.1 1e+00 0.55628205 0.04963230
10  1.0 1e+00 0.06628205 0.03615496
11  5.0 1e+00 0.05608974 0.03755733
12 10.0 1e+00 0.05608974 0.03755733
13  0.1 5e+00 0.55628205 0.04963230
14  1.0 5e+00 0.51025641 0.05558924
15  5.0 5e+00 0.50512821 0.06013429
16 10.0 5e+00 0.50512821 0.06013429
17  0.1 1e+01 0.55628205 0.04963230
18  1.0 1e+01 0.53076923 0.06373112
19  5.0 1e+01 0.52564103 0.06043648
20 10.0 1e+01 0.52564103 0.06043648
21  0.1 1e+02 0.55628205 0.04963230
22  1.0 1e+02 0.55628205 0.04963230
23  5.0 1e+02 0.55628205 0.04963230
24 10.0 1e+02 0.55628205 0.04963230
```

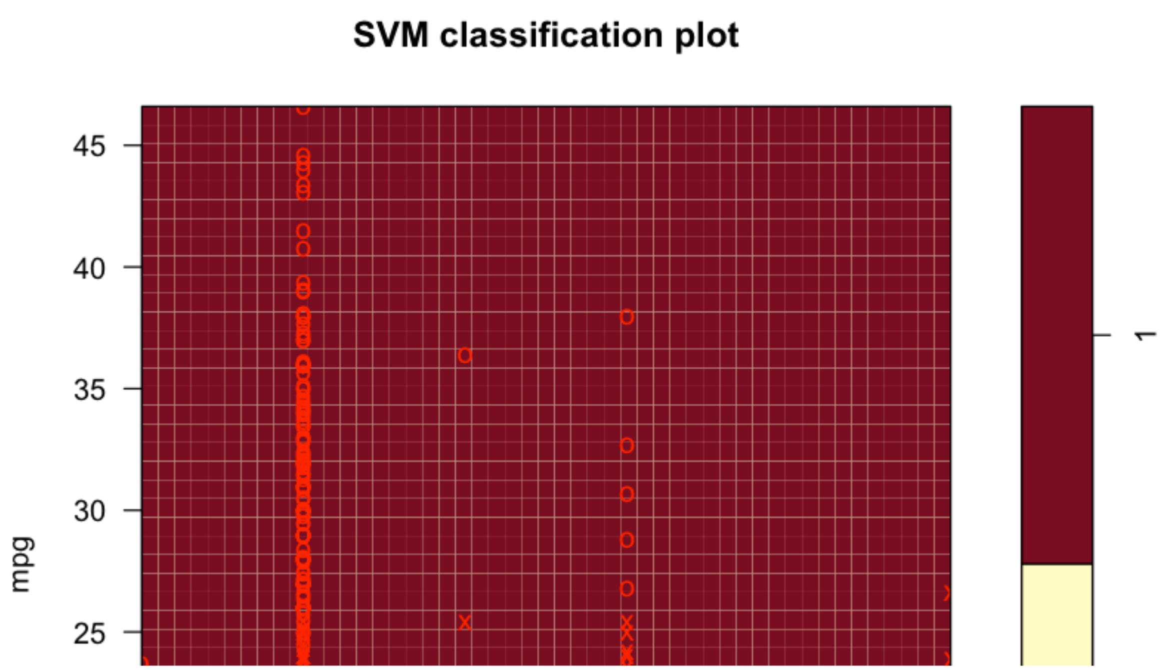
In [39]:
```
svm.linear = svm(mpglevel~., data = Auto, kernel = "linear",
                 cost = 1)
svm.poly = svm(mpglevel~., data = Auto, kernel = "polynomial",
               cost = 10,
    degree = 2)
svm.radial = svm(mpglevel~., data = Auto, kernel = "radial",
                 cost = 10, gamma = 0.01)
plotpairs = function(fit) {
    for (name in names(Auto)[!(names(Auto)
                              %in% c("mpg", "mpglevel", "name"))]
        plot(fit, Auto, as.formula(paste("mpg~", name, sep = "")))
    }
}
plotpairs(svm.linear)
```



In [40]:
```
plotpairs(svm.poly)
```



In [41]:
```
plotpairs(svm.radial)
```



In [ ]: