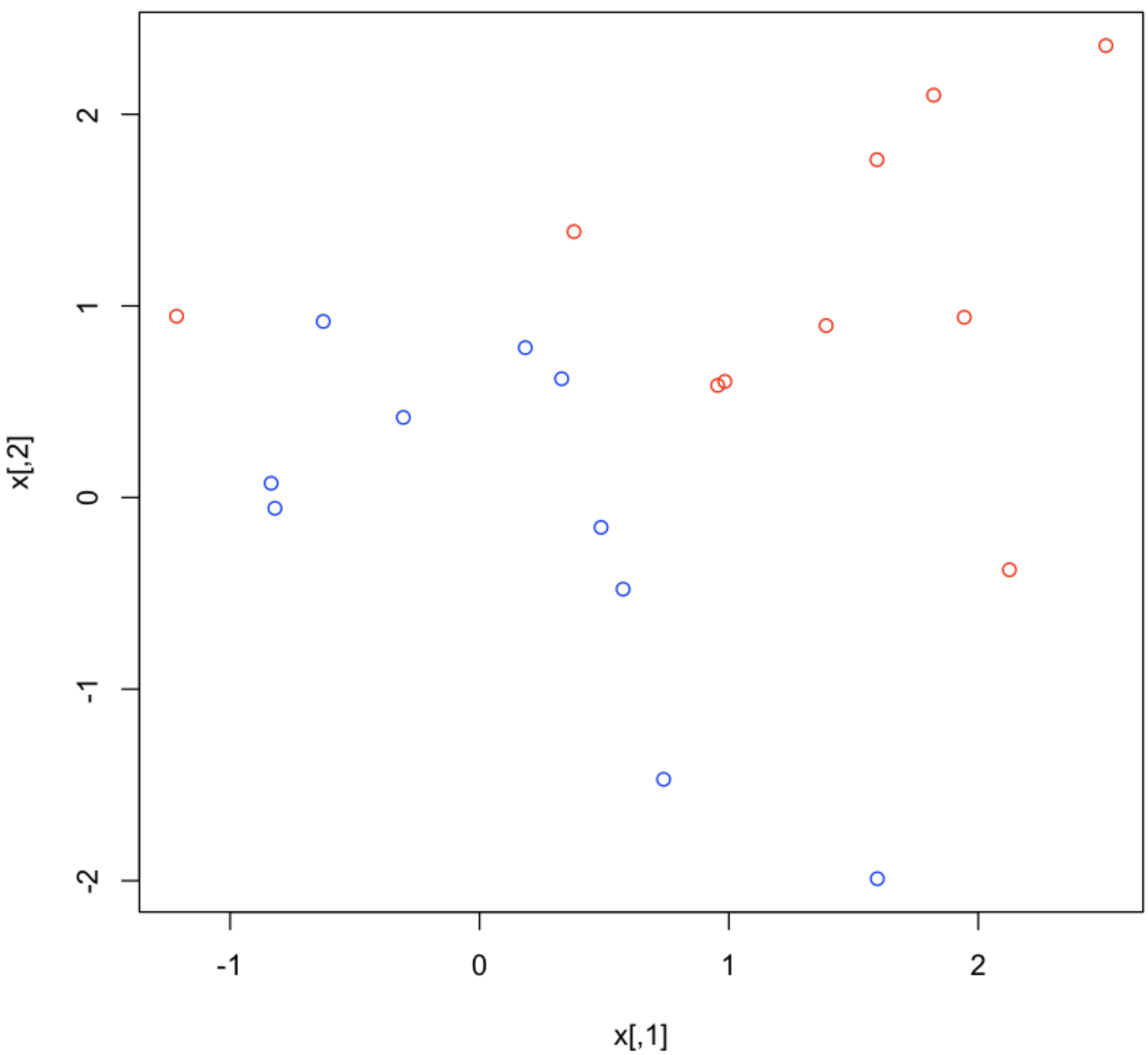
#Lab: Support Vector Machines

In [1]:
```
library(e1071)
```

In [2]:
```
set.seed(1)
x=matrix(rnorm(20*2),ncol=2)
y=c(rep(-1,10),rep(1,10))
x[y==1,]=x[y==1,]+1
plot(x,col=(3-y))
```

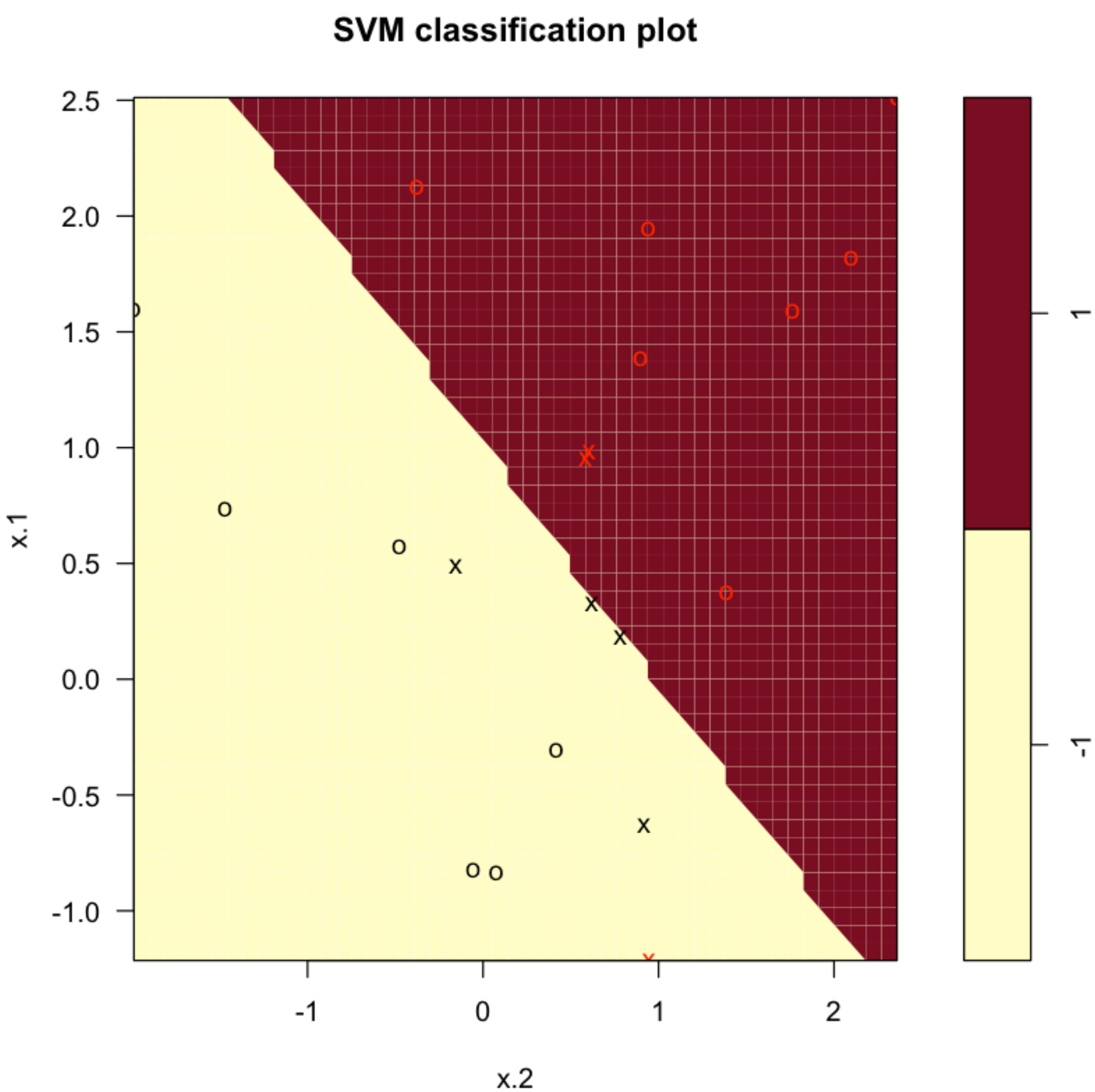


In [3]:

```
#svm() function can be used to fit a support vector classifier
#when the argument kernel="linear" is used.
dat=data.frame(x=x,y=as.factor(y)) #resp as factor variable
library(e1071)
svmfit=svm(y~., data=dat, kernel="linear", cost=10,
scale=FALSE) #svm the support vector classifier for a given value

#False--not to scale each feature to have mean zero or standard de
```

In [4]:

```
plot(svmfit , dat) #plot support vector classifier
```



In [5]:

```
#-1 class is shown in yellow, and +1 class is shown in red,
#disc boundry is "linear"
```

In [6]:
```
svmfit$index

#The support vectors are plotted as crosses and the remaining
#observations are plotted as circles
#even support vectors are below-
```

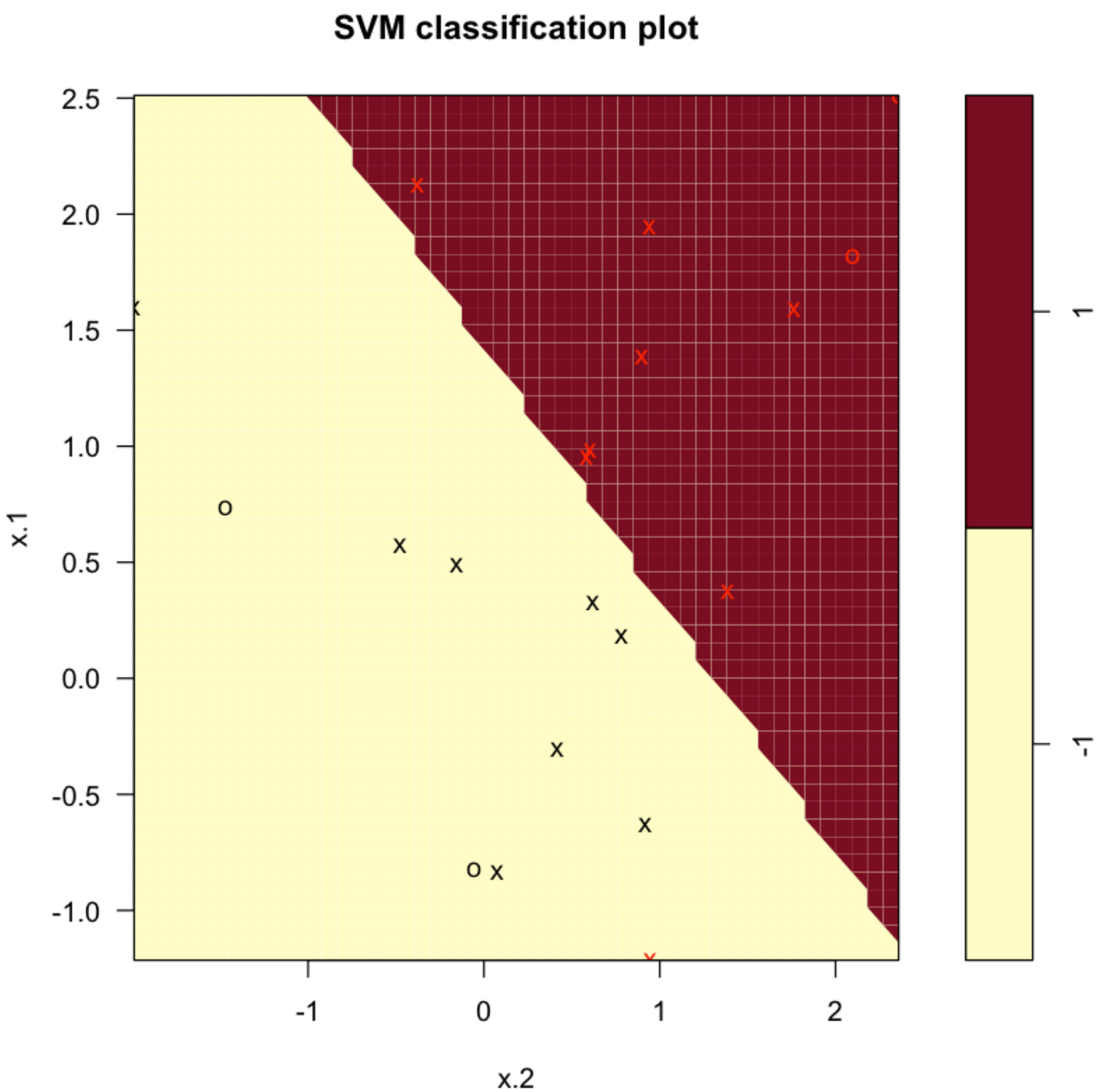
1 2 5 7 14 16 17

In [7]:
```
summary(svmfit)
```

```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale
= FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

In [8]:
```
#linear kernel was used with cost=10, and that there were seven
#support vectors, four in one class and three in the other.
```

In [9]:
```
#if we use small value of cost
svmfit=svm(y~., data=dat, kernel="linear", cost=0.1, scale=FALSE)
plot(svmfit , dat)
```



In [10]:
```
svmfit$inde

#smaller cost,larger number of support vectors, coz the margin is
```

1 2 3 4 5 7 9 10 12 13 14 15 16 17 18 20

In [11]:
```
set.seed (1)
tune.out=tune(svm,y~.,data=dat,kernel="linear",
ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
##tune() performs ten-fold cross-validation on a set
summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.05

- Detailed performance results:
   cost error dispersion
1 1e-03  0.55  0.4377975
2 1e-02  0.55  0.4377975
3 1e-01  0.05  0.1581139
4 1e+00  0.15  0.2415229
5 5e+00  0.15  0.2415229
6 1e+01  0.15  0.2415229
7 1e+02  0.15  0.2415229
```

In [12]:
```
#cost=0.1 results in the lowest cross-validation error rate.

```

In [13]:

```
bestmod=tune.out$best.model #tune()stores the best model obtained,
summary(bestmod)
```

```
Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(co
st = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

In [14]:

```
xtest=matrix(rnorm(20*2), ncol=2)
ytest=sample(c(-1,1), 20, rep=TRUE)
xtest[ytest==1,]=xtest[ytest==1,] + 1
testdat=data.frame(x=xtest, y=as.factor(ytest))

#predict() can be used to predict the class label on a
#set of test observations, at any given value of the cost paramete

```

In [15]:

```
#predicting the class labels of these test observations.
ypred=predict(bestmod,testdat)
table(predict=ypred, truth=testdat$y)
```

```
       truth
predict -1 1
     -1  9 1
     1   2 8
```

In [16]:

```
svmfit=svm(y~.,data=dat, kernel="linear", cost=.01, scale=FALSE)
ypred=predict(svmfit,testdat)
table(predict=ypred,truth=testdat$y)
```

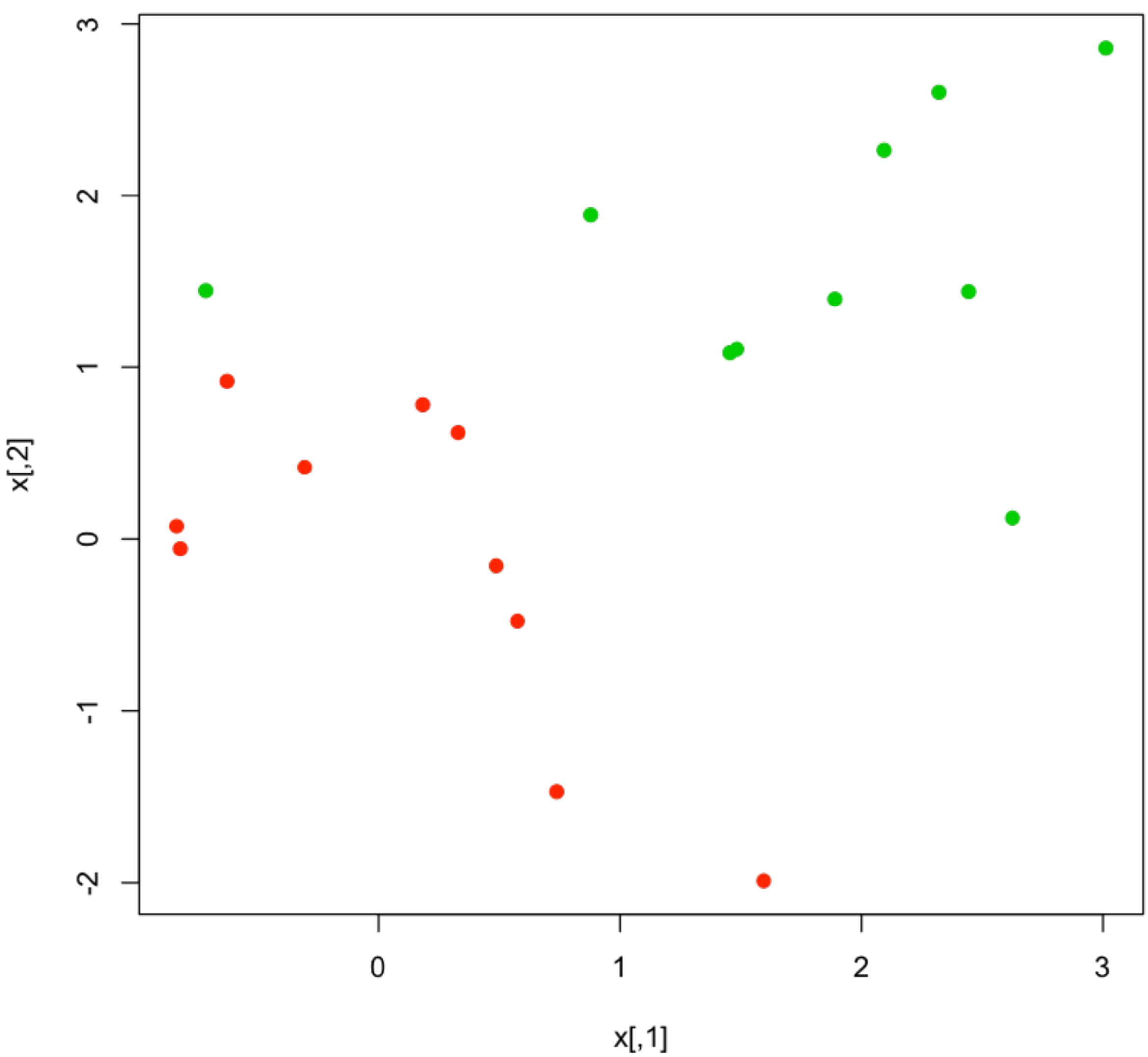
```
       truth
predict -1  1
     -1 11  6
     1   0  3
```

In [17]:

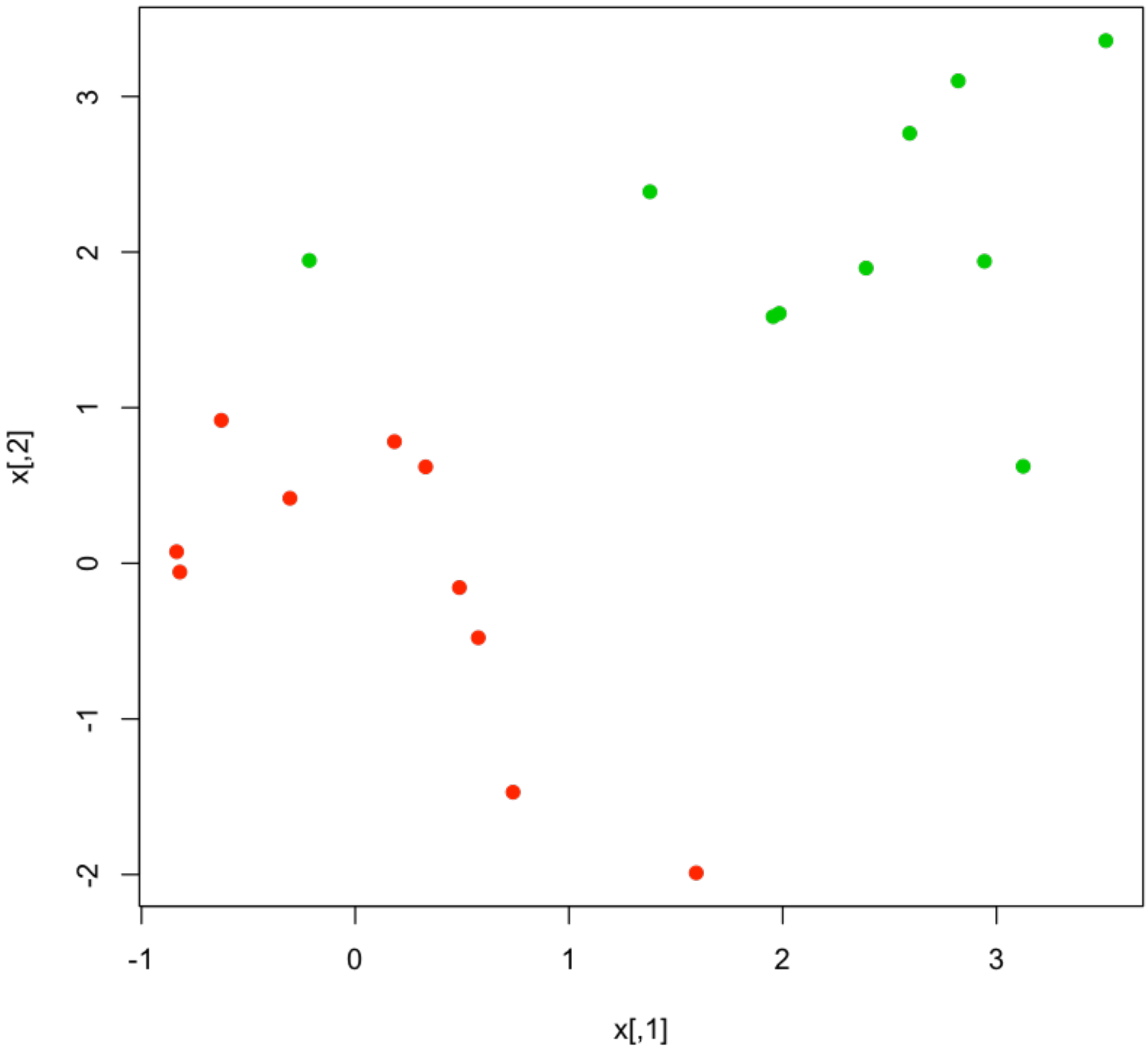
```
# three additional observation is misclassified.
```

In [18]:

```
#In case two classes are linearly separable.
x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)
```



In [19]:
```
x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)
```



In [20]:
```
#observations are just barely linearly separable. We fit the SV ca
#and plot the resulting hyperplane, using a very large value of co
#so that no observations are misclassified
```

In [21]:

```
dat=data.frame(x=x,y=as.factor(y))
svmfit=svm(y~., data=dat, kernel="linear",cost=1e5)
summary(svmfit)

#No training errors were made and only three support vectors were
```

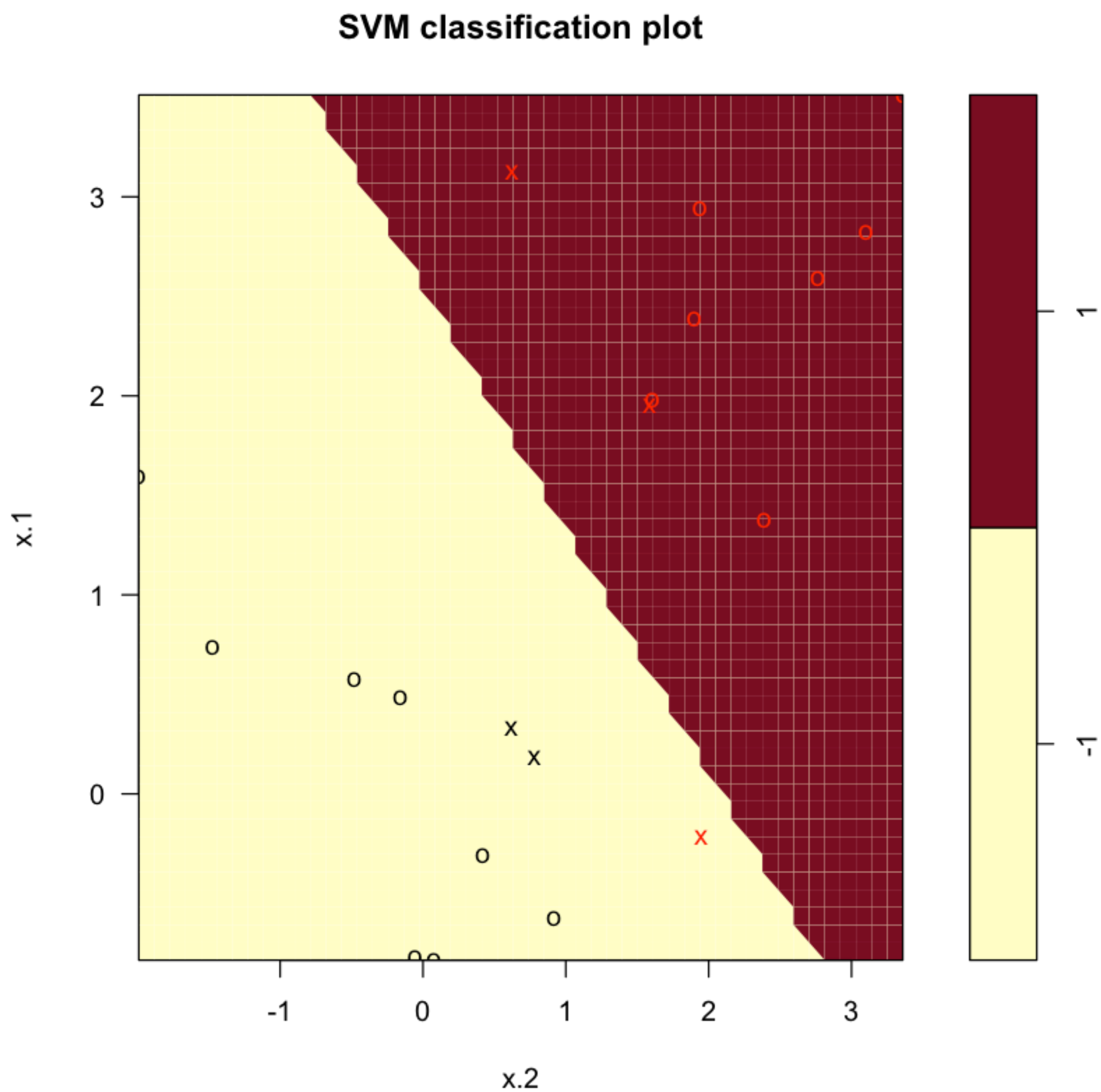
```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+05
      gamma:  0.5

Number of Support Vectors:  3

 ( 1 2 )


Number of Classes:  2

Levels:
 -1 1
```

In [22]:

```
svmfit=svm(y~., data=dat, kernel="linear", cost=1)
summary(svmfit)
plot(svmfit ,dat)
```

```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  5

 ( 2 3 )
```

```
Number of Classes:  2

Levels:
 -1 1
```



In [56]:

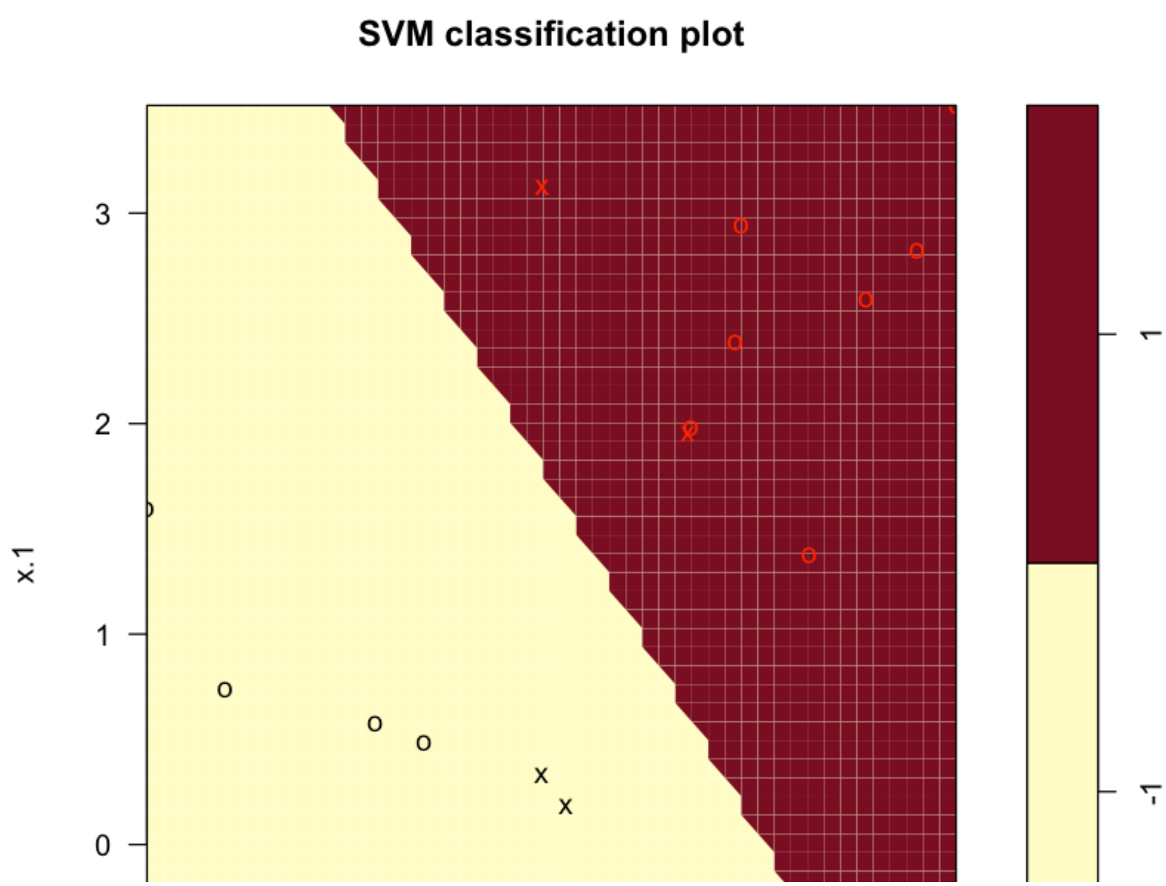
```
#from fig, margin is very narrow as the observations that are not
#indicated as circles, are very close to the decision boundary.
#It seems likely that this model will perform poorly on test data.
 #close to the decision boundary. It seems likely that this model
```

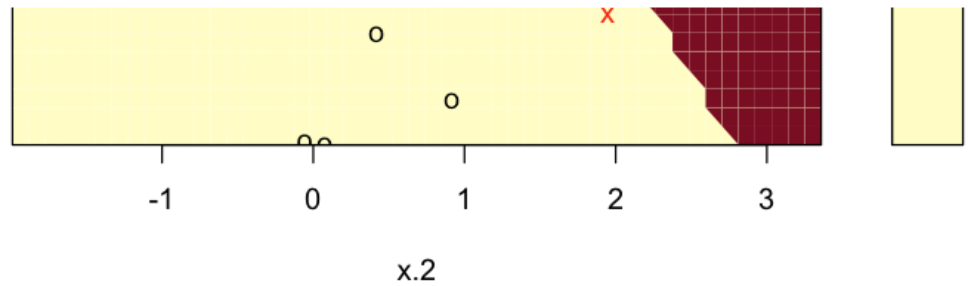
In [24]:

```
# trying with smaller value oc cost
svmfit=svm(y~., data=dat, kernel="linear", cost=1)
summary(svmfit)
```

```
summary(svmfit)
plot(svmfit ,dat)
```

```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  5

 ( 2 3 )


Number of Classes:  2

Levels:
 -1 1
```



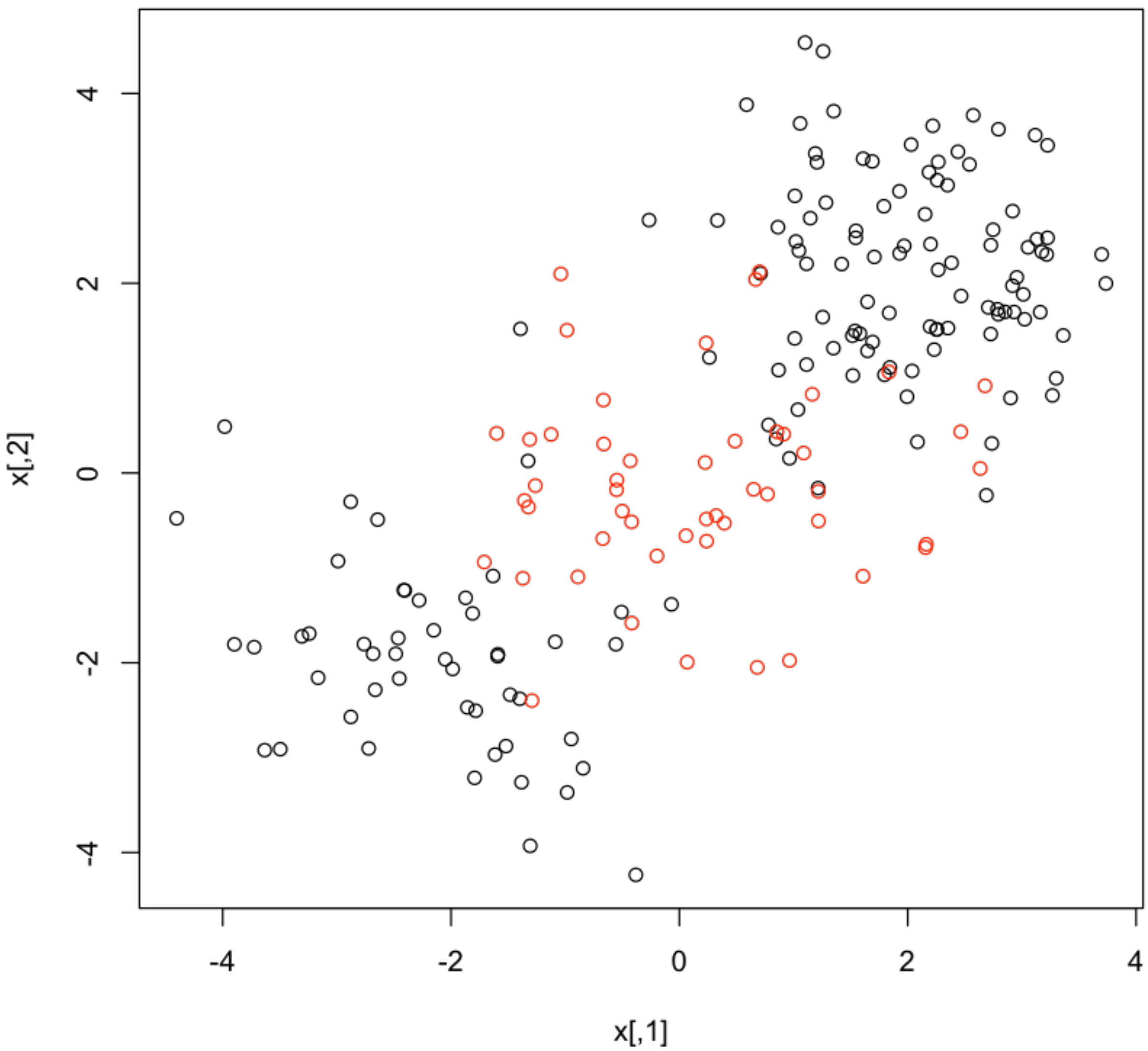


In [25]:
```
#with cost=1, we misclassify a training observation, but we obtair
#much wider margin and make use of sevensupport vectors.
# this model will perform better on test data than the model with
```

#Support Vector Machine

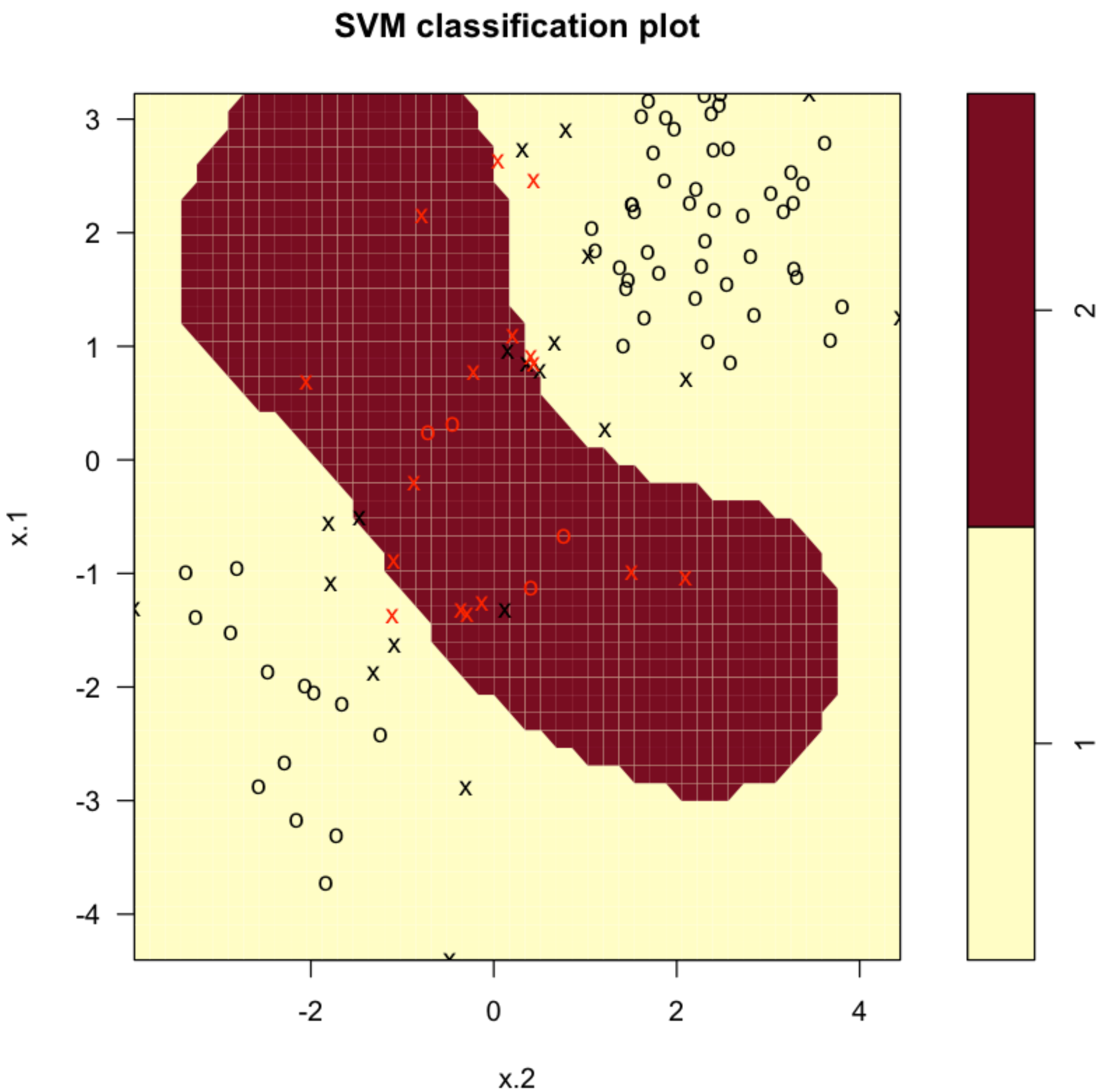
In [26]:
```
set.seed(3)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
```

In [27]:
```
plot(x, col=y)
```



In [28]:
```
#data randomly split into training and testing groups.
#fitting the training data using the svm()with a radial kernel and
```

In [29]:
```
train=sample(200,100)
svmfit=svm(y~., data=dat[train,], kernel="radial", gamma=1,
cost =1)
plot(svmfit,dat[train,])
```

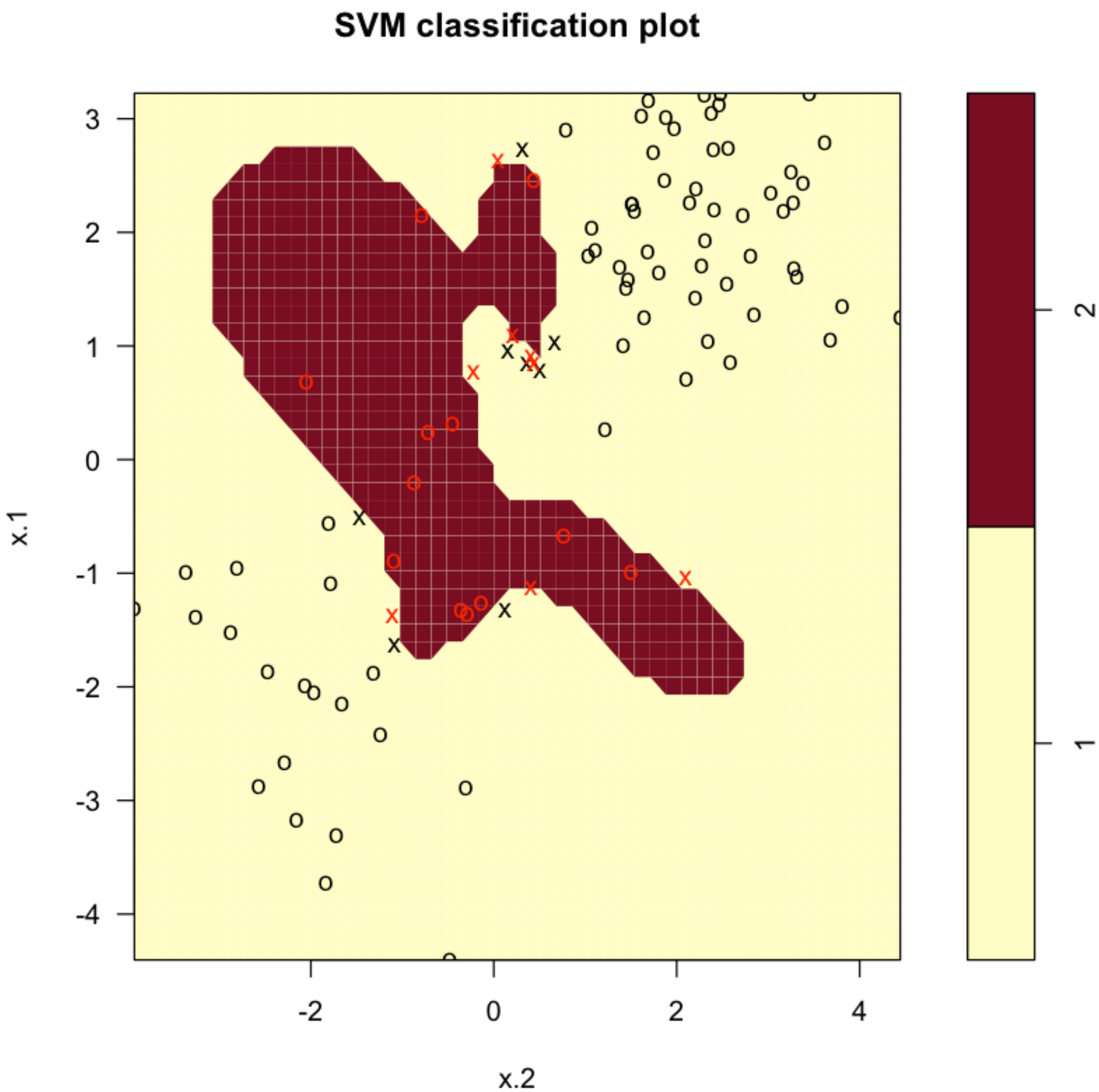


In [30]:
```
#rplot shows esulting SVM has a decidedly non-linear boundary.
summary(svmfit)
```

```
Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma =
1,
    cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  1

Number of Support Vectors:  36

 ( 20 16 )


Number of Classes:  2

Levels:
 1 2
```

In [31]:

```
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1, cost=1e
plot(svmfit ,dat[train ,])
```



In [32]:

```
#fair number of training errors in this SVM fit from plot.
#If we increase the value of cost, we can reduce the n0. of traini
#but there will be more irregular decision boundary that may overf
```

In [33]:

```
set.seed (1)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial",
ranges=list(cost=c(0.1,1,10,100,1000),
gamma=c(0.5,1,2,3,4) ))
summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1     1

- best performance: 0.09

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.20 0.11547005
2  1e+00   0.5  0.11 0.08755950
3  1e+01   0.5  0.14 0.08432740
4  1e+02   0.5  0.11 0.05676462
5  1e+03   0.5  0.11 0.08755950
6  1e-01   1.0  0.20 0.11547005
7  1e+00   1.0  0.09 0.07378648
8  1e+01   1.0  0.11 0.07378648
9  1e+02   1.0  0.11 0.08755950
10 1e+03   1.0  0.12 0.10327956
11 1e-01   2.0  0.20 0.11547005
12 1e+00   2.0  0.11 0.07378648
13 1e+01   2.0  0.12 0.09189366
14 1e+02   2.0  0.15 0.09718253
15 1e+03   2.0  0.14 0.10749677
16 1e-01   3.0  0.20 0.11547005
17 1e+00   3.0  0.11 0.07378648
18 1e+01   3.0  0.14 0.08432740
19 1e+02   3.0  0.15 0.09718253
20 1e+03   3.0  0.13 0.08232726
21 1e-01   4.0  0.20 0.11547005
22 1e+00   4.0  0.12 0.07888106
23 1e+01   4.0  0.15 0.09718253
24 1e+02   4.0  0.17 0.10593499
25 1e+03   4.0  0.16 0.10749677
```

In [34]:

```
#the best choice of parameters involves cost=1 and gamma=1
```

In [35]:
```
#predict() to view test set predictions for this model

table(true=dat[-train,"y"], pred=predict(tune.out$best.model,
                                         newdata=dat[-train ,]))
```

```
    pred
true  1  2
   1 65  5
   2  8 22
```

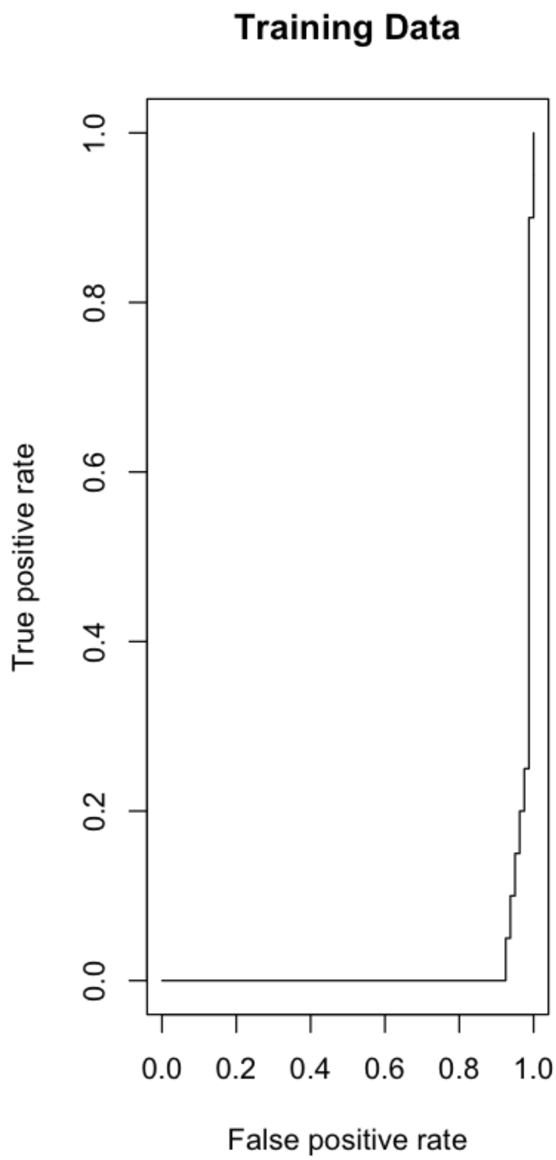
In [36]:
```
### # ROC Curves
library(ROCR)
```

```
Loading required package: gplots

Attaching package: ‘gplots’

The following object is masked from ‘package:stats’:

    lowess
```

In [37]:
```
rocplot=function(pred, truth, ...){
    predob = prediction (pred, truth)
    perf = performance (predob , "tpr", "fpr")
    plot(perf ,...)
}
```

In [38]:
```
svmfit.opt=svm(y~., data=dat[train,], kernel="radial", gamma=2,
                   cost=1,decision.values=T)

fitted=attributes(predict(svmfit.opt,dat[train,],
                          decision.values=TRUE))$decision.values
```

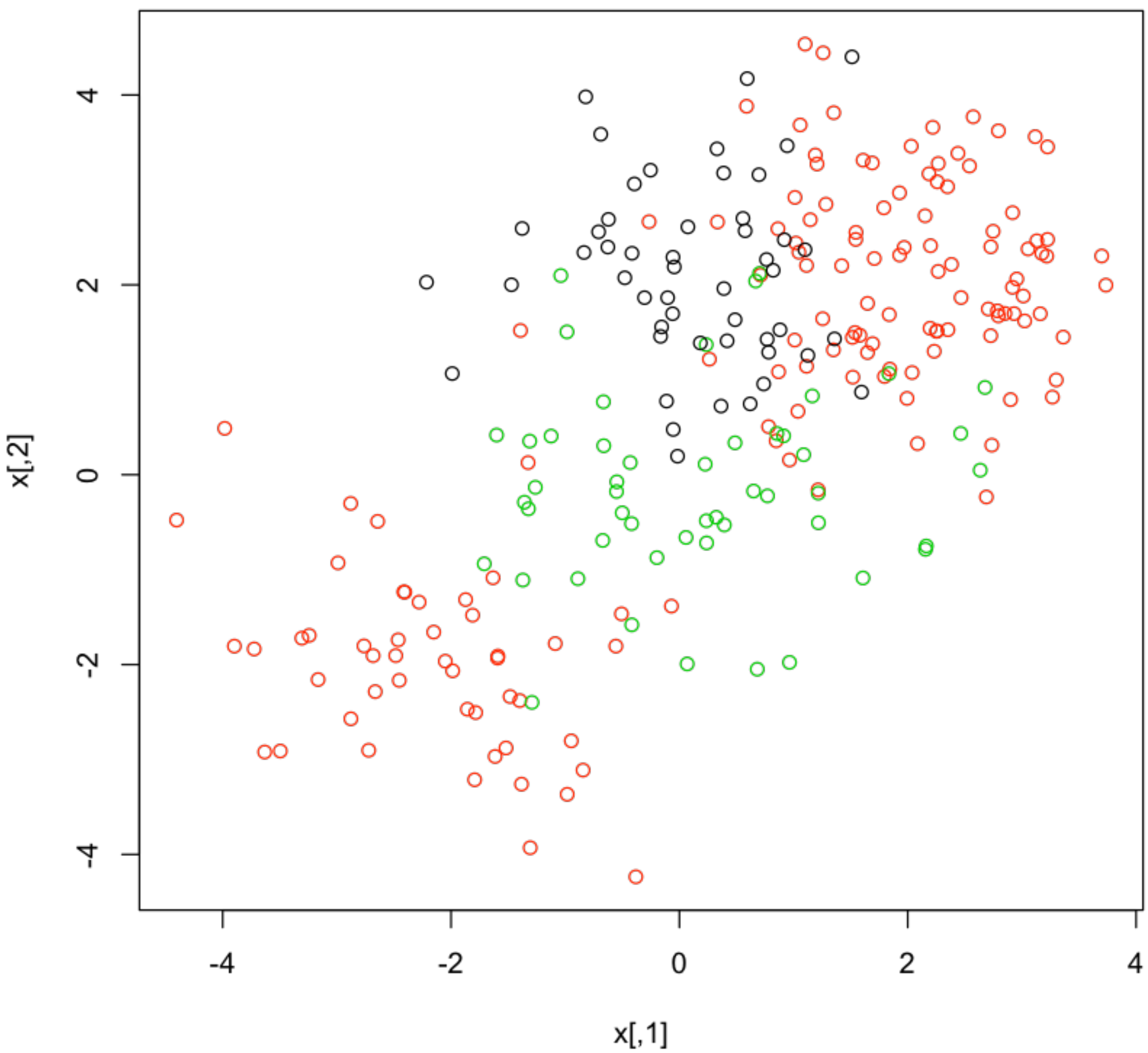
In [39]:
```
#Roc plot
par(mfrow=c(1,2))
rocplot(fitted ,dat[train ,"y"],main="Training Data")

```



In [40]:
```
#SVM appears to be producing accurate predictions. increasing γ ca
#more flexible fit and generate further improvements in accuracy.

```

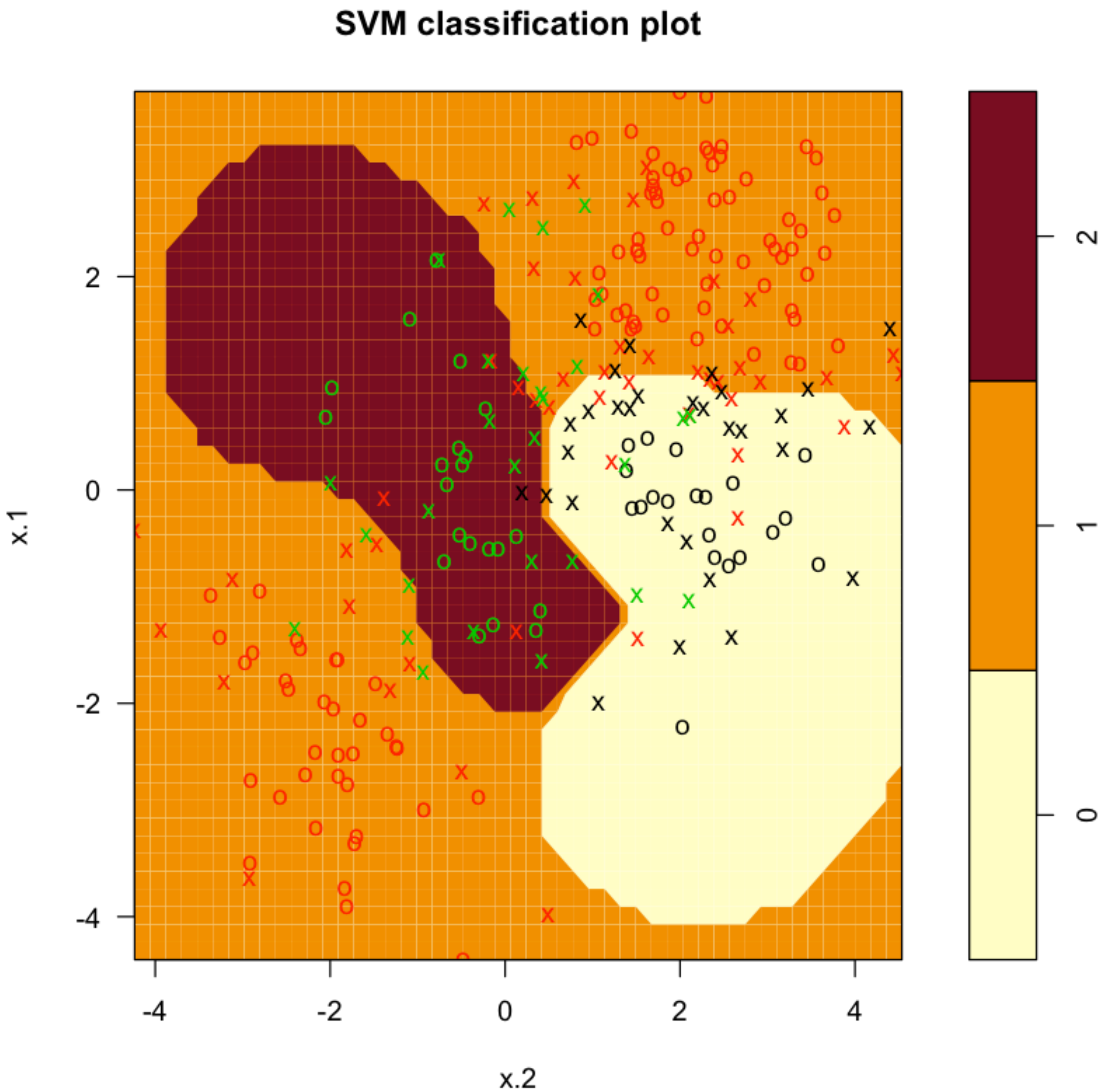
In [41]:
```
### SVM with multiclass
#If the response is a factor containing more than two levels, then
#function will perform multi-class classification
```

In [42]:
```
set.seed(1)
x=rbind(x, matrix(rnorm(50*2), ncol=2))
y=c(y, rep(0,50))
x[y==0,2]=x[y==0,2]+2
dat=data.frame(x=x, y=as.factor(y))
par(mfrow=c(1,1))
plot(x,col=(y+1))
```



In [43]:

```
#fitting svm to datra
svmfit=svm(y~., data=dat, kernel="radial", cost=10, gamma=1)
plot(svmfit , dat)
```



In [44]:

```
#he e1071 library can also be used to perform sv regression,
#if the resp vector
#passed in to svm() is numerical rather than a factor.
```

In [45]:
```
#Application to Gene Expression Data

library(ISLR)
names(Khan)
```

'xtrain' 'xtest' 'ytrain' 'ytest'

In [46]:
```
dim(Khan$xtrain)
```

63 2308

In [47]:
```
dim(Khan$xtest )
```

20 2308

In [48]:
```
length(Khan$ytrain )
```

63

In [49]:
```
length(Khan$ytest)
```

20

In [50]:
```
table(Khan$ytrain)
```

```
 1  2  3  4
 8 23 12 20
```

In [51]:
```
table(Khan$ytest)
```

```
1 2 3 4
3 6 6 5
```

In [52]:
```
dat=data.frame(x=Khan$xtrain , y=as.factor(Khan$ytrain ))
out=svm(y~., data=dat, kernel="linear",cost=10)
summary(out)
```

```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.0004332756

Number of Support Vectors:  58

 ( 20 20 11 7 )


Number of Classes:  4

Levels:
 1 2 3 4
```

In [53]:
```
table(out$fitted , dat$y)
```

```
     1  2  3  4
  1  8  0  0  0
  2  0 23  0  0
  3  0  0 12  0
  4  0  0  0 20
```

In [54]:
```
dat.te=data.frame(x=Khan$xtest , y=as.factor(Khan$ytest ))
pred.te=predict(out, newdata=dat.te)
table(pred.te, dat.te$y)
```

```
pred.te 1 2 3 4
      1 3 0 0 0
      2 0 6 2 0
      3 0 0 4 0
      4 0 0 0 5
```

In [55]:

```
#there are no training errors.because the large number of variable
#relative to the no. of observations implies that it is easy to fi
#hyperplanes that fully separate the classes. We are most interest
#in the SV classifier’s performance on the training observations,
#but rather its performance on the test observations
```

In [ ]:

```

```