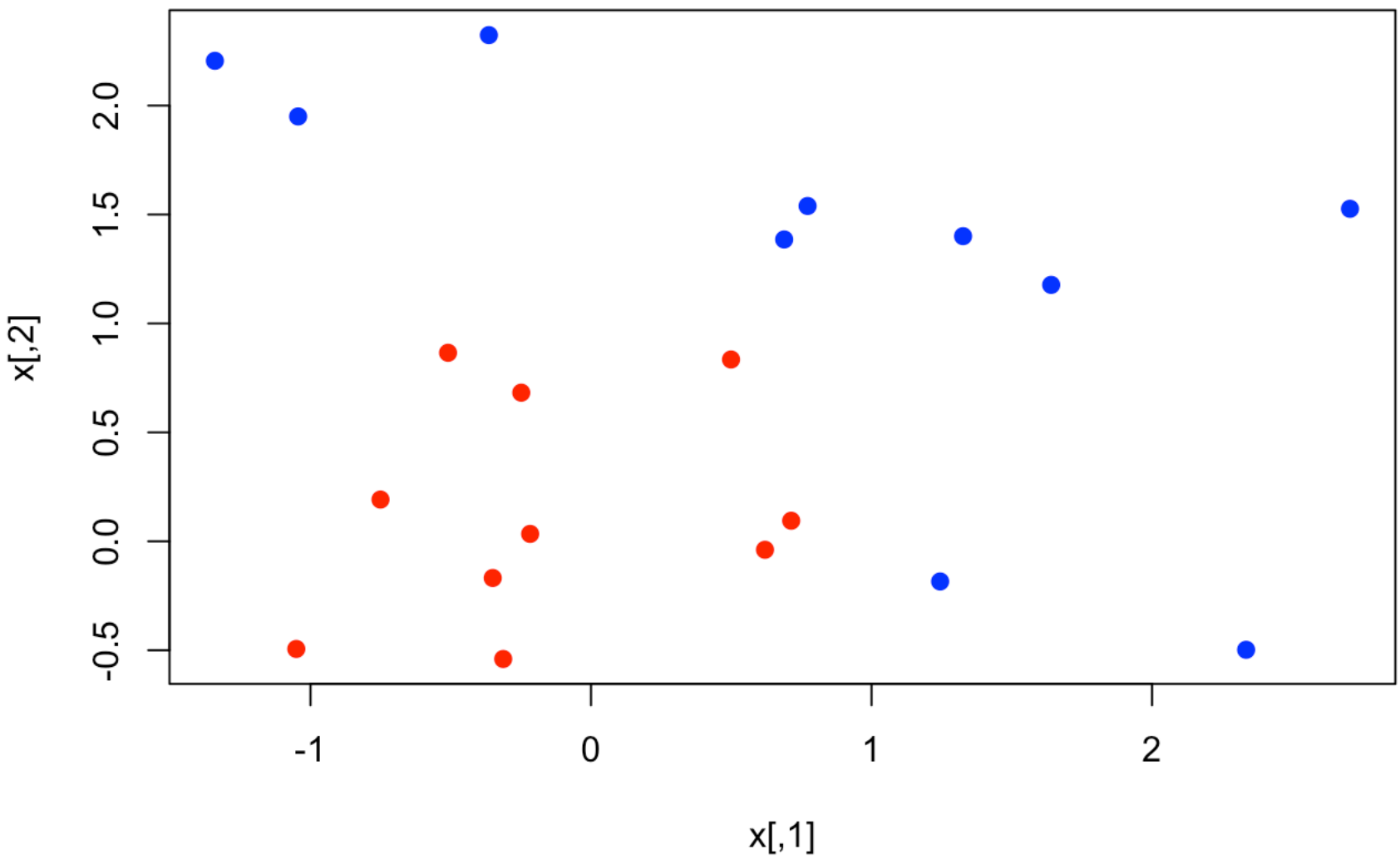
SL RLab9: SUPPORT VECTOR MACHINES AND CLASSIFIERS

SVM Demo easier in low dimension, to see the data.

LINEAR SVM CLASSIFIER

Generate data in two dimensions, a little separated; color code points by response, and create plots. Use CV to select parameters later as exercise

```
set.seed(10111)
x=matrix(rnorm(40), 20, 2)
y=rep(c(-1,1), c(10,10))
x[y==1,]=x[y==1,]+1
plot(x, col=y+3, pch=19)
```



Install ‘e1071’ package which contains svm() function Compute the fit; specify ‘cost’, as tuning parameter install.packages(“e1071”)

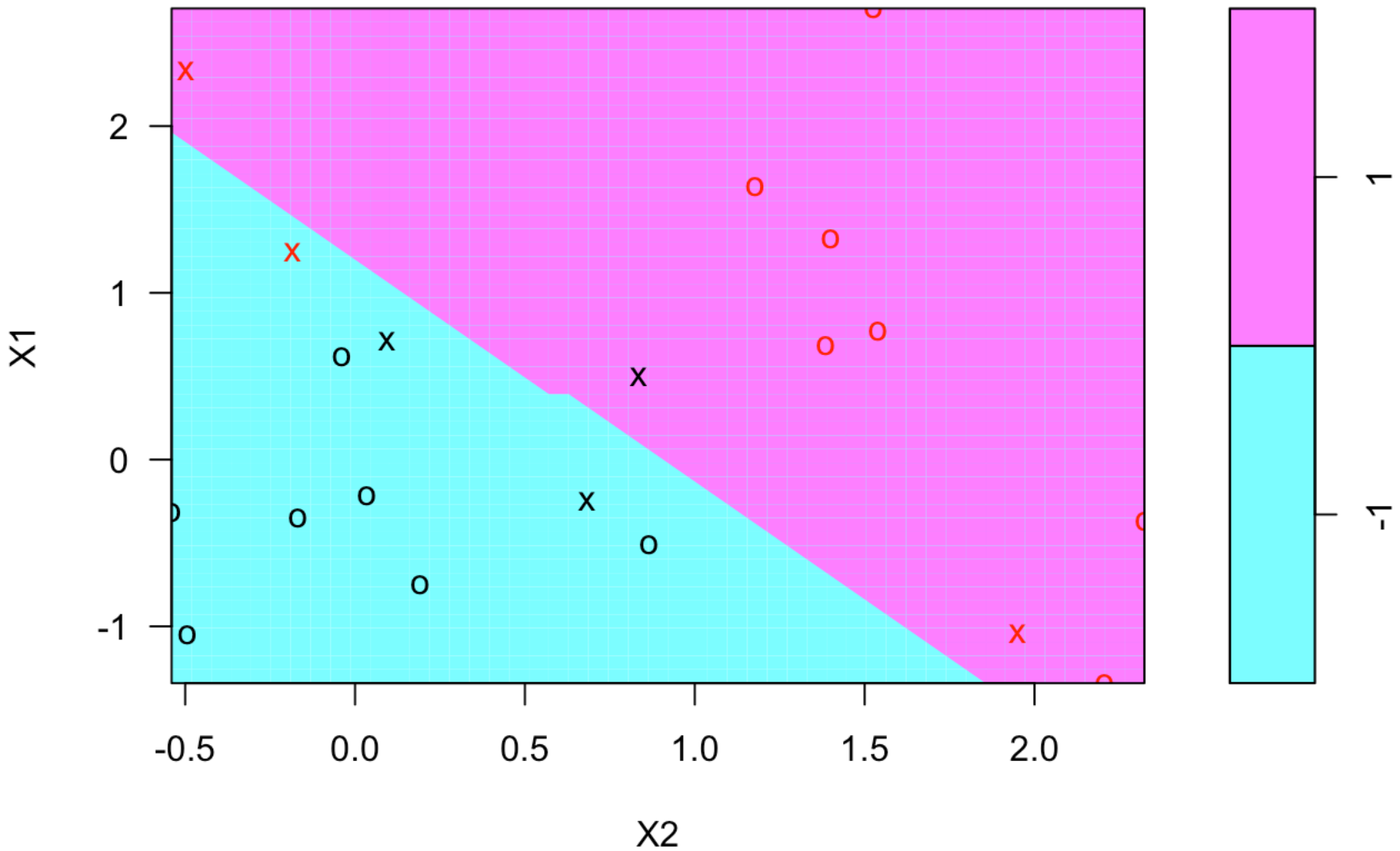
```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.2
```

```
dat = data.frame(x,y=as.factor(y))
svmfit = svm(y~., data=dat, kernel='linear', cost=10, scale=FALSE)
print(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:  6
```

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

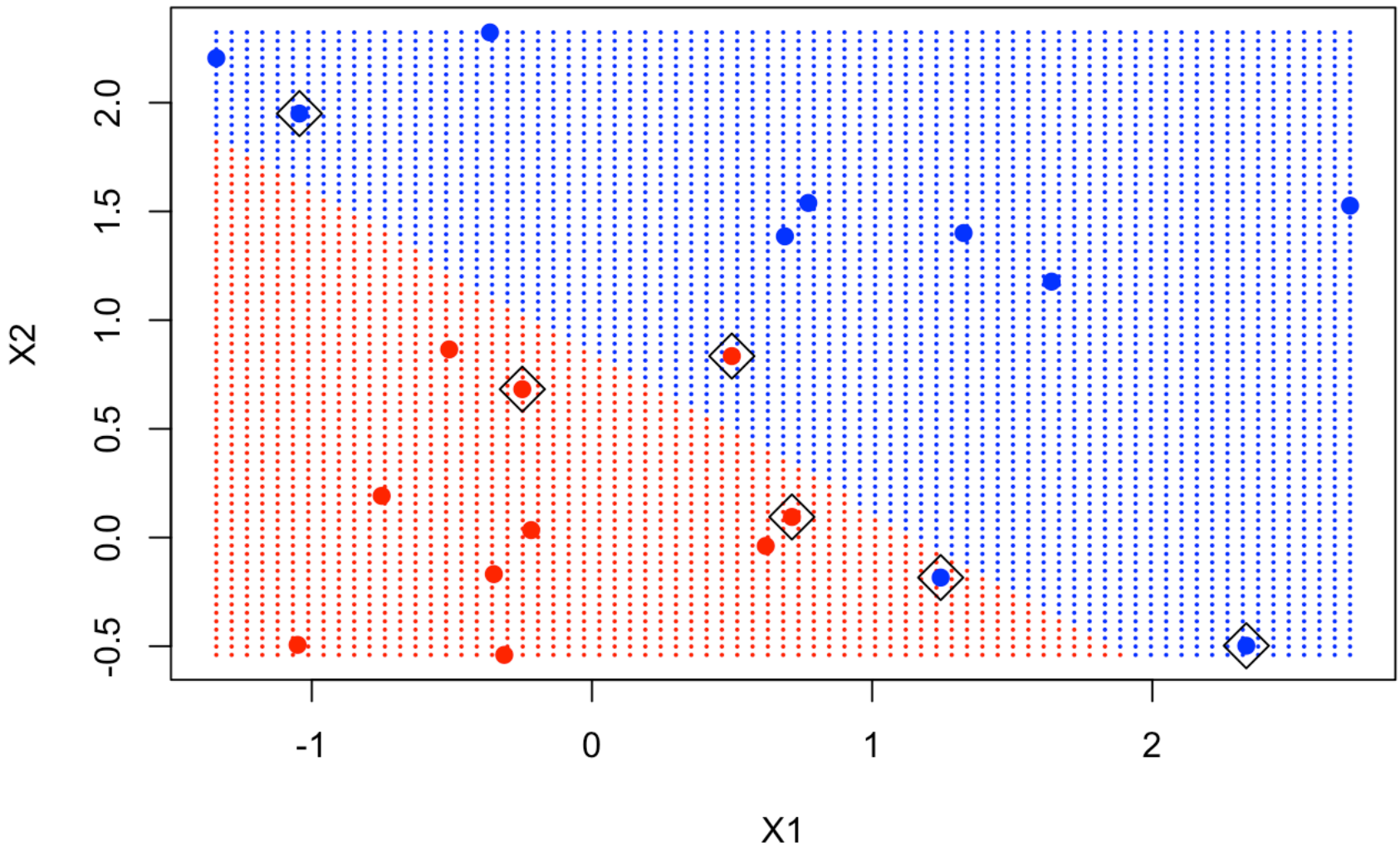


Default plot is unconventional (X2 on horiz axis), therefore, we create a custom plot, by writing a function, for later use:

First make a grid of 75x75 values for X1 and X2 using ‘expand.grid’ function, which produces coordinates of ’n*n’ points on a lattice covering the domain of ‘x’. Next, we make a prediction at each point on lattice. Then, we plot the lattice, color-coded according to classification. This allows us to see the decision boundary.

The support points (points on the margin, or on wrong side of the margin) are indexed in the ‘$index’ component of the fit.

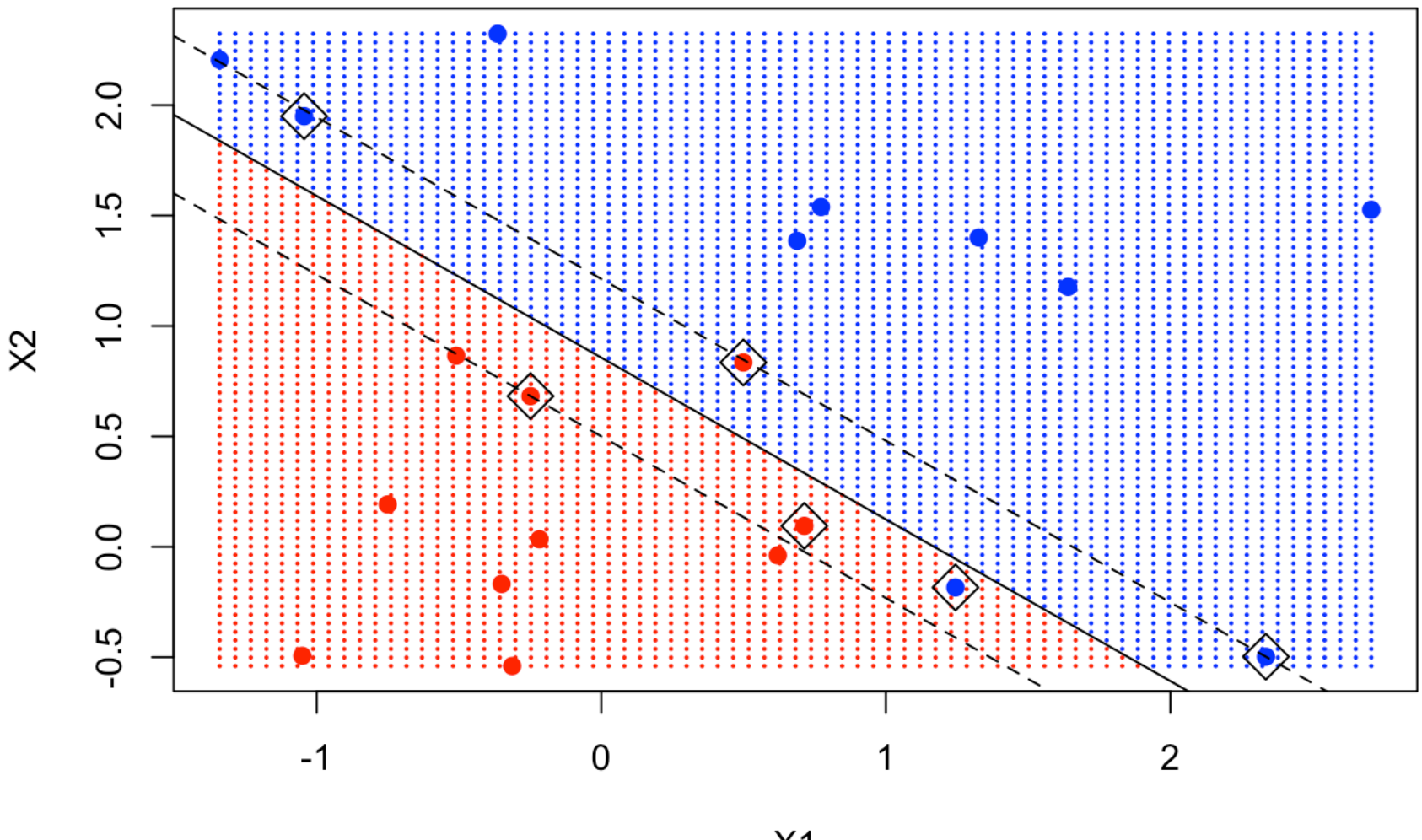
```
make.grid = function(x, n=75){
    grange = apply(x, 2, range)
    x1 = seq(from=grange[1,1], to=grange[2,1], length=n)
    x2 = seq(from=grange[1,2], to=grange[2,2], length=n)
    expand.grid(X1=x1, X2=x2)
}
xgrid = make.grid(x)
ygrid = predict(svmfit, xgrid)
plot(xgrid, col=c("red", "blue")[as.numeric(ygrid)], pch=20, cex=.2)
points(x, col=y+3, pch=19)
points(x[svmfit$index,], pch=5, cex=2)
```



‘SVM’ function is not too friendly, in that we have to do some work to get back linear coefficients, as described in the textbook. The reason is probably that this only makes sense for linear kernels, and the function is more general. Here we use a formula to extract the coefficients; (see Ch. 12 in Elements of Statistical Learning).

We extract the linear coefficients, and with simple algebra, use the coefficients to calculate the decision boundary and margins.

```
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 = svmfit$rho
plot(xgrid, col=c("red", "blue")[as.numeric(ygrid)], pch=20, cex=.2)
points(x, col=y+3, pch=19)
points(x[svmfit$index,], pch=5, cex=2)
abline(beta0/beta[2], -beta[1]/beta[2])
abline((beta0-1)/beta[2], -beta[1]/beta[2], lty=2)
abline((beta0+1)/beta[2], -beta[1]/beta[2], lty=2)
```



NON-LINEAR SVM

Use cross validation to select the tuning parameter, gamma. Run SVM on data where non-linear boundary is called for. Load ‘mixture’ dataset from ESL

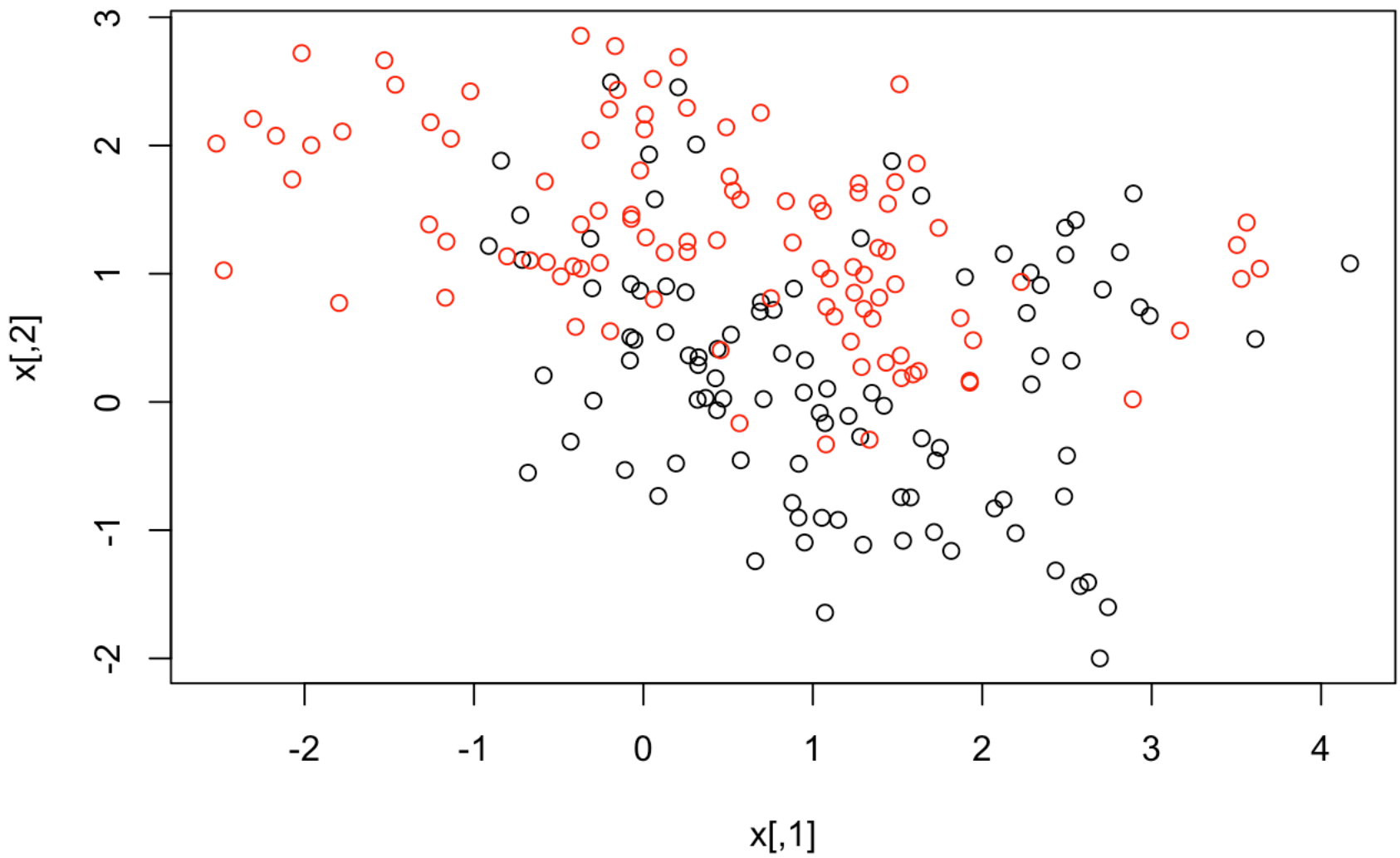
```
load("ESL.mixture.rda")
names(ESL.mixture)
```

```
## [1] "x"        "y"        "xnew"     "prob"     "marginal" "px1"
## [7] "px2"      "means"
```

```
# need to remove x,y variables created above
rm(x,y)
attach(ESL.mixture)
```

These data are 2-dimensional, let’s plot them. There is considerable overlap, but something in this relationship. Let’s fit a nonlinear SVM, using a radial kernel.

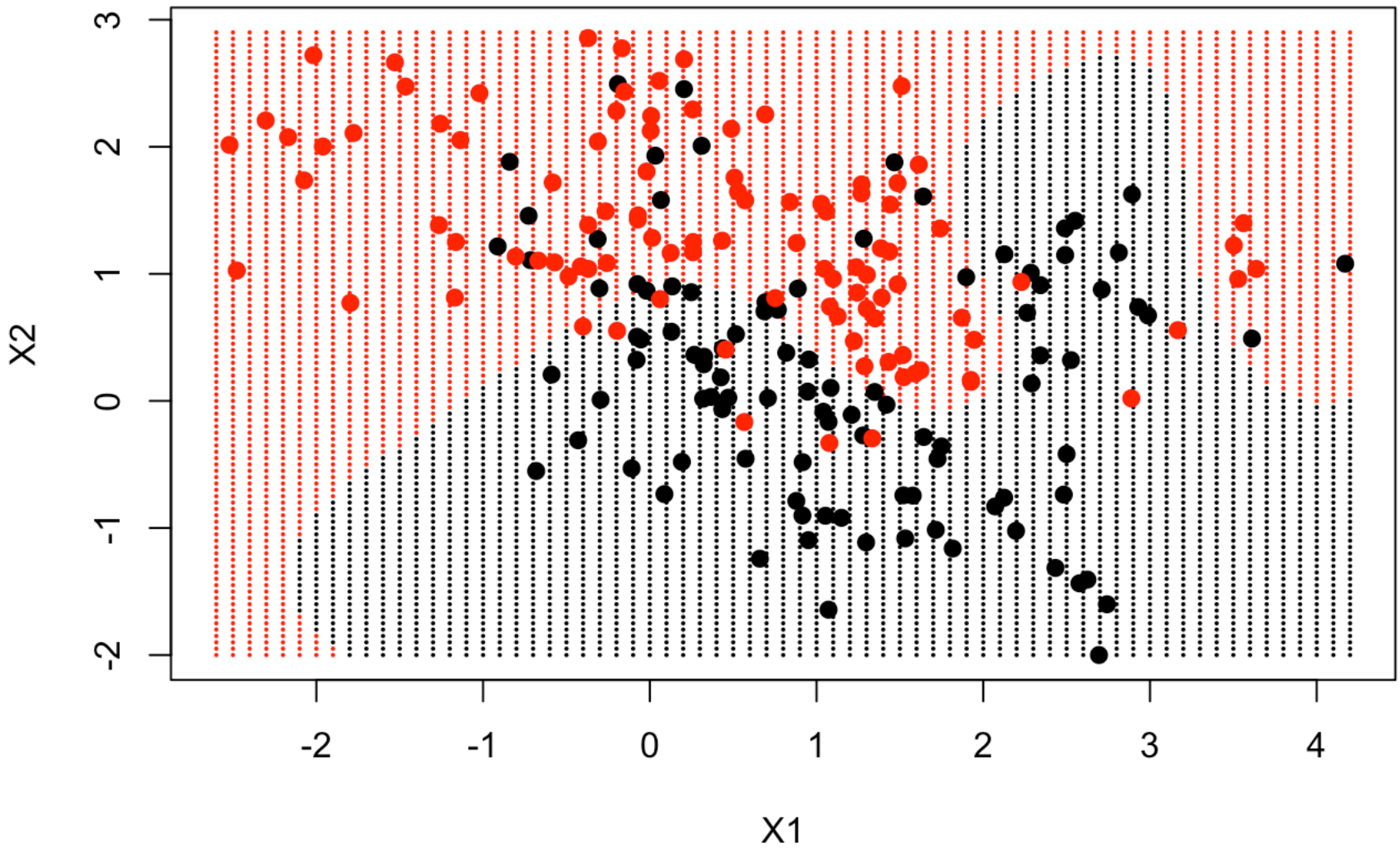
```
plot(x, col=y+1)
```




```
dat = data.frame(y=factor(y),x)
fit = svm(factor(y)~., data=dat, scale=FALSE, kernel="radial", cost=5)
```

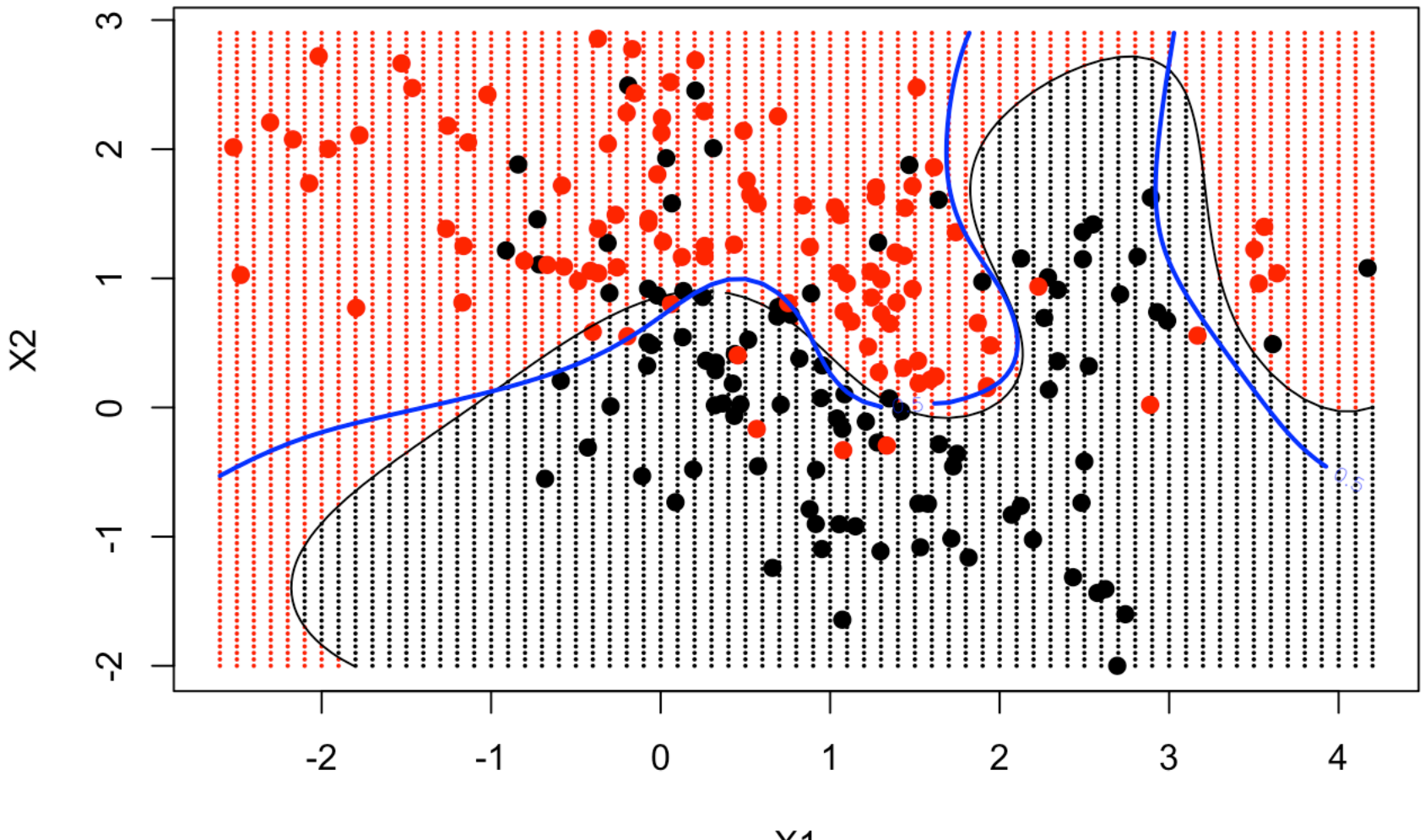
Now, create a grid, as before, and make predictions on the grid. These data have grid points for each variable included on the DF.

```
xgrid = expand.grid(X1=px1, X2=px2)
ygrid = predict(fit, xgrid)
plot(xgrid, col=as.numeric(ygrid), pch=20, cex=.2)
points(x, col=y+1, pch=19)
```



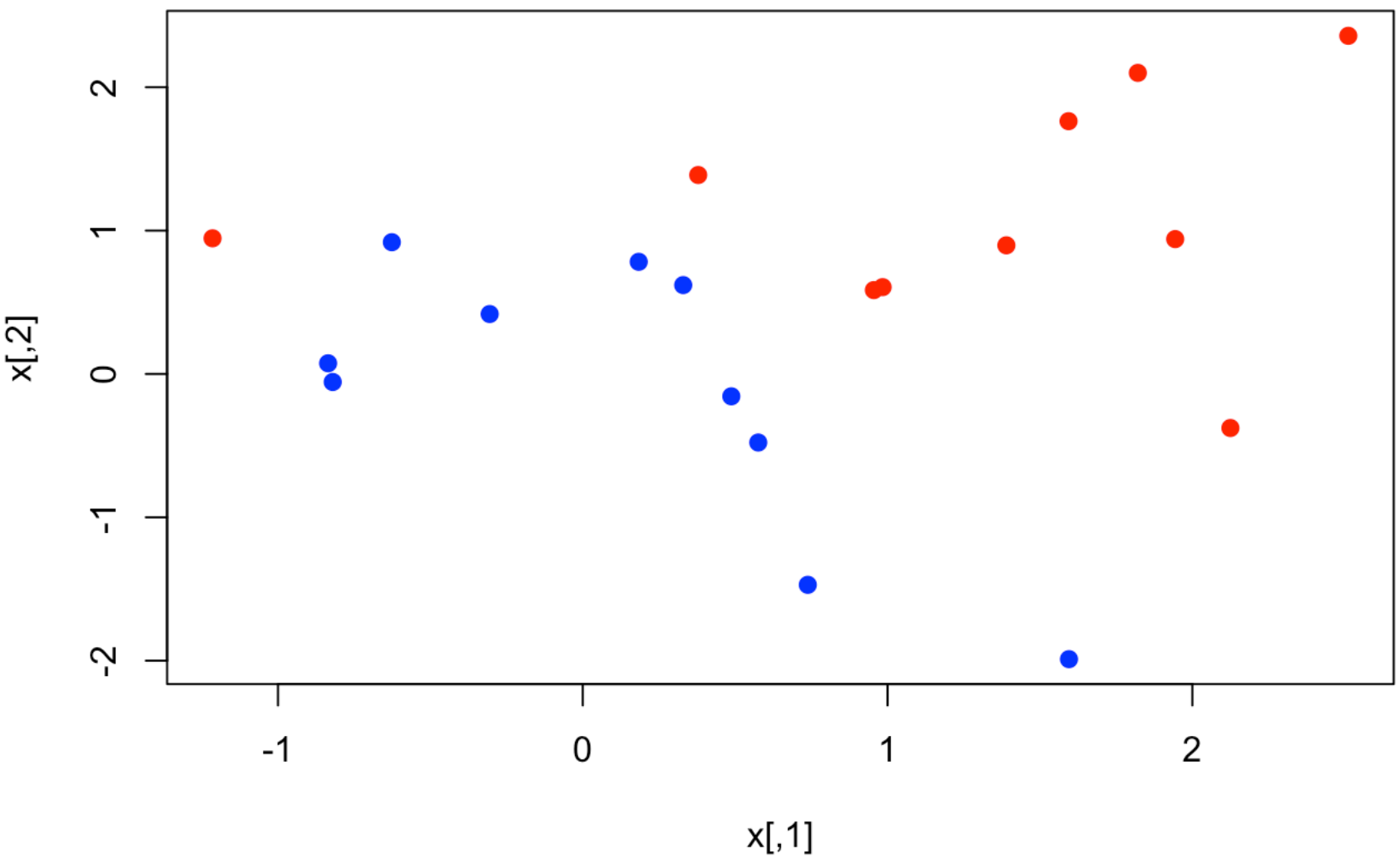
We can go further, and have predict function produce the actual function estimates at each of our grid points. We can include the actual decision boundary on the plot by using countour() function. On the dataframe is also ‘prob’, which is the true probability of class 1 for these data, at the gridpoints. If we plot its 0.5 contour, that will give us the *Bayes Decision Boundary*, which is the best one could ever do.

```
func = predict(fit, xgrid, decision.values=TRUE)
func = attributes(func)$decision

xgrid = expand.grid(X1=px1, X2=px2)
ygrid = predict(fit, xgrid)
plot(xgrid, col=as.numeric(ygrid), pch=20, cex=.2)
points(x, col=y+1, pch=19)

contour(px1, px2, matrix(func,69,99), level=0, add=TRUE)
contour(px1, px2, matrix(prob,69,99), level=0.5, add=TRUE, col="blue", lwd=2)
```

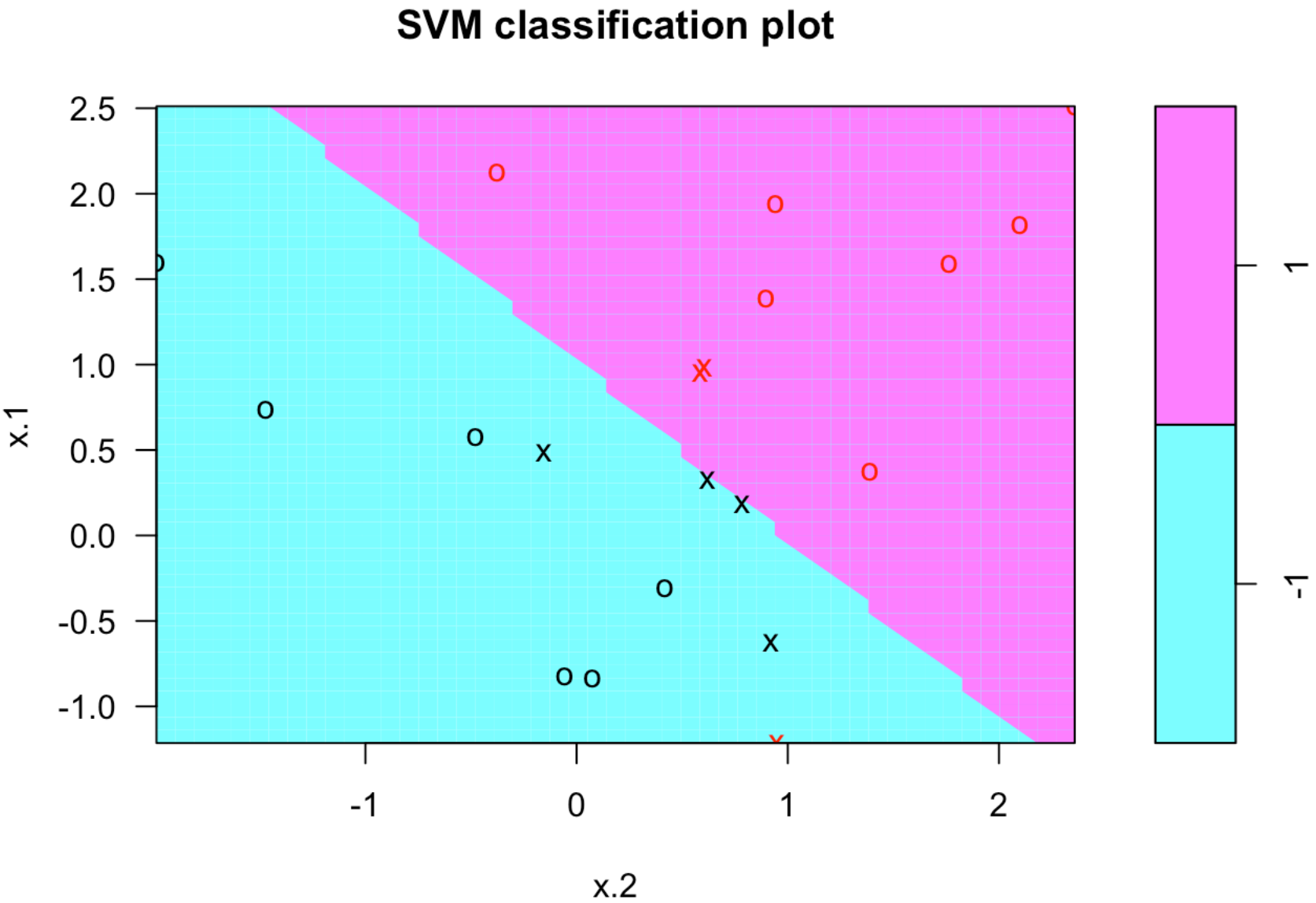


CROSS-VALIDATION FOR LINEAR SVM CLASSIFIER

```
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), pch=19)
```




```
dat = data.frame(x=x, y=as.factor(y))
library(e1071)
svmfit = svm(y~., data=dat, kernel="linear", cost=10, scale=FALSE)
plot(svmfit, dat)
```




```
svmfit$index
```

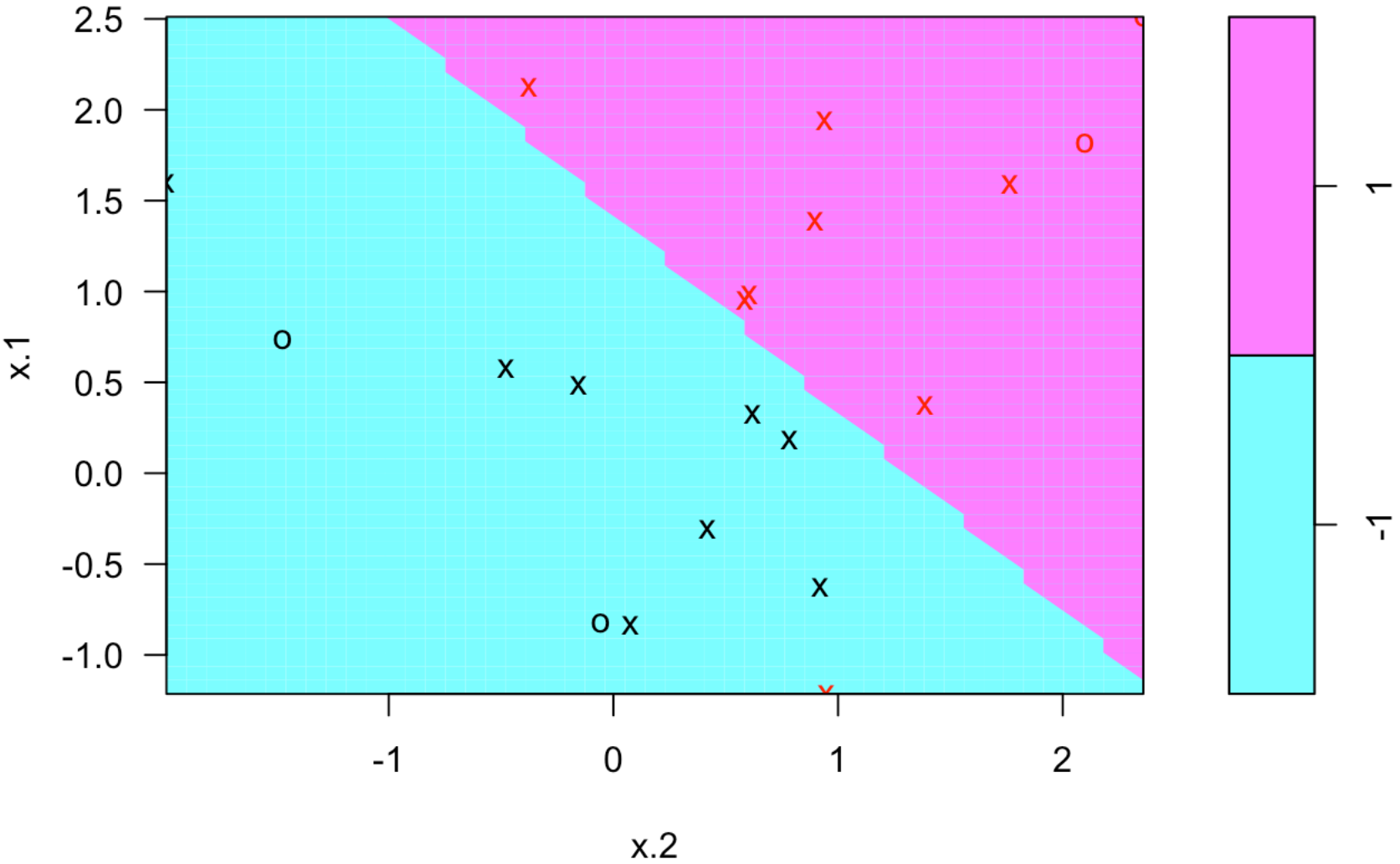
```
## [1]  1  2  5  7 14 16 17
```

```
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
```

What if we use a smaller value of the cost parameter: the margin will be wider and we will obtain a larger number of support vectors.

```
svmfit = svm(y~., data=dat, kernel="linear", cost=0.1, scale=FALSE)
plot(svmfit, dat)
```




```
svmfit$index
```

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

Library includes built-in function tune() to perform CROSS VALIDATION. The tune() function stores the best model obtained, which can be accessed as follows:

```
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)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.70  0.4216370
## 2 1e-02  0.70  0.4216370
## 3 1e-01  0.10  0.2108185
## 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
```

```
bestmod = tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = 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
```

predict() function can predict class lable on a set of test observations, at any given value of the cost parameter. Begin by generating a TEST data set; then predict class labels of test observations, using best model from cross validation.

```
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))

ypred = predict(bestmod, testdat)
table(predict=ypred, truth=testdat$y)
```

```
##         truth
## predict -1  1
##      -1 11  1
##      1   0  8
```

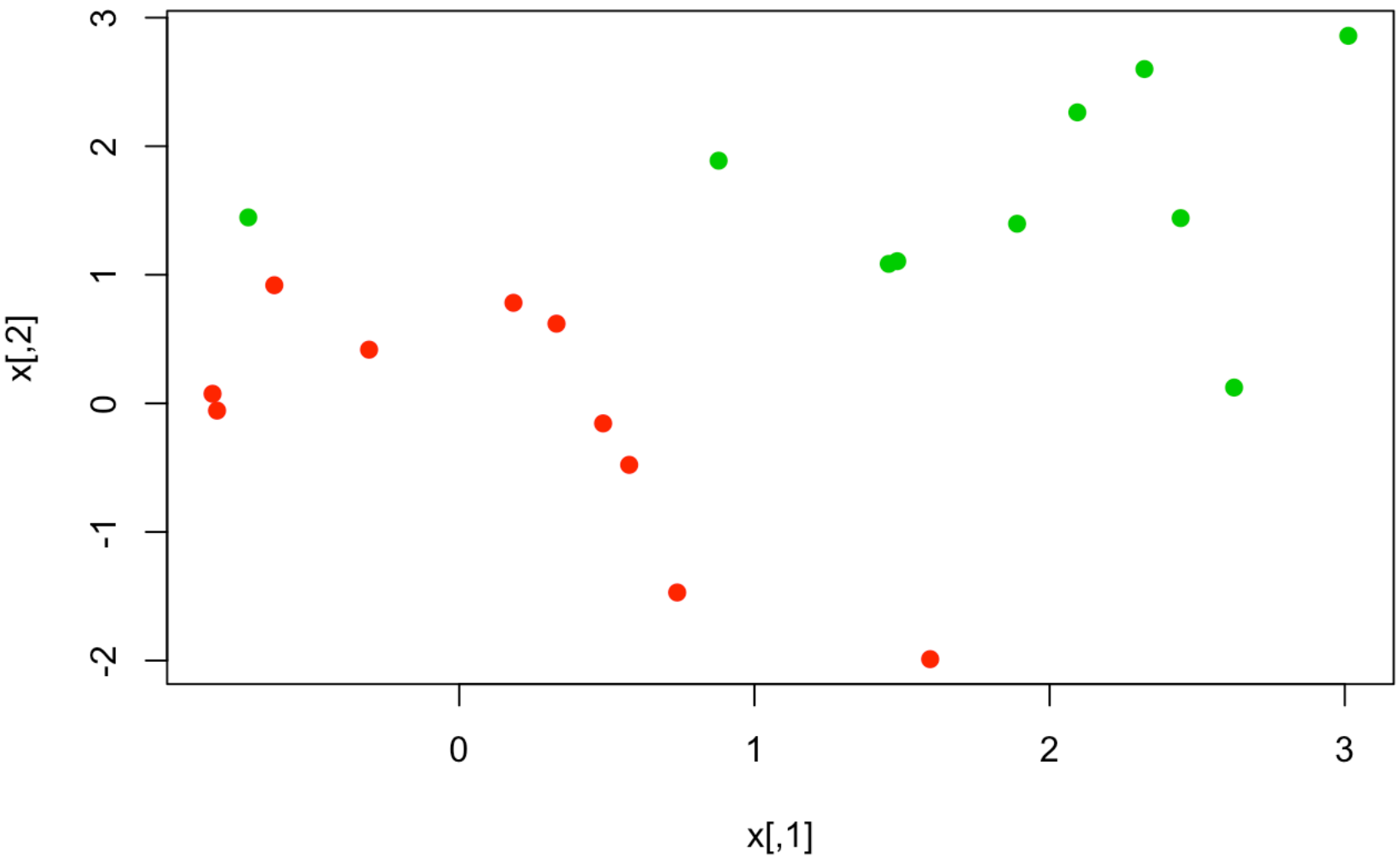
What if we used used cost=0.01 instead?

```
svmfit = svm(y~., data=dat, kernel="linear", cost=0.01, scale=FALSE)
ypred = predict(svmfit, testdat)
table(predict=ypred, truth=testdat$y)
```

```
##         truth
## predict -1  1
##      -1 11  2
##      1   0  7
```

Find separating hyperplane using the svm() function; fit SV classifier and plot the resulting hyperplane, using very large value of cost with no misclassification

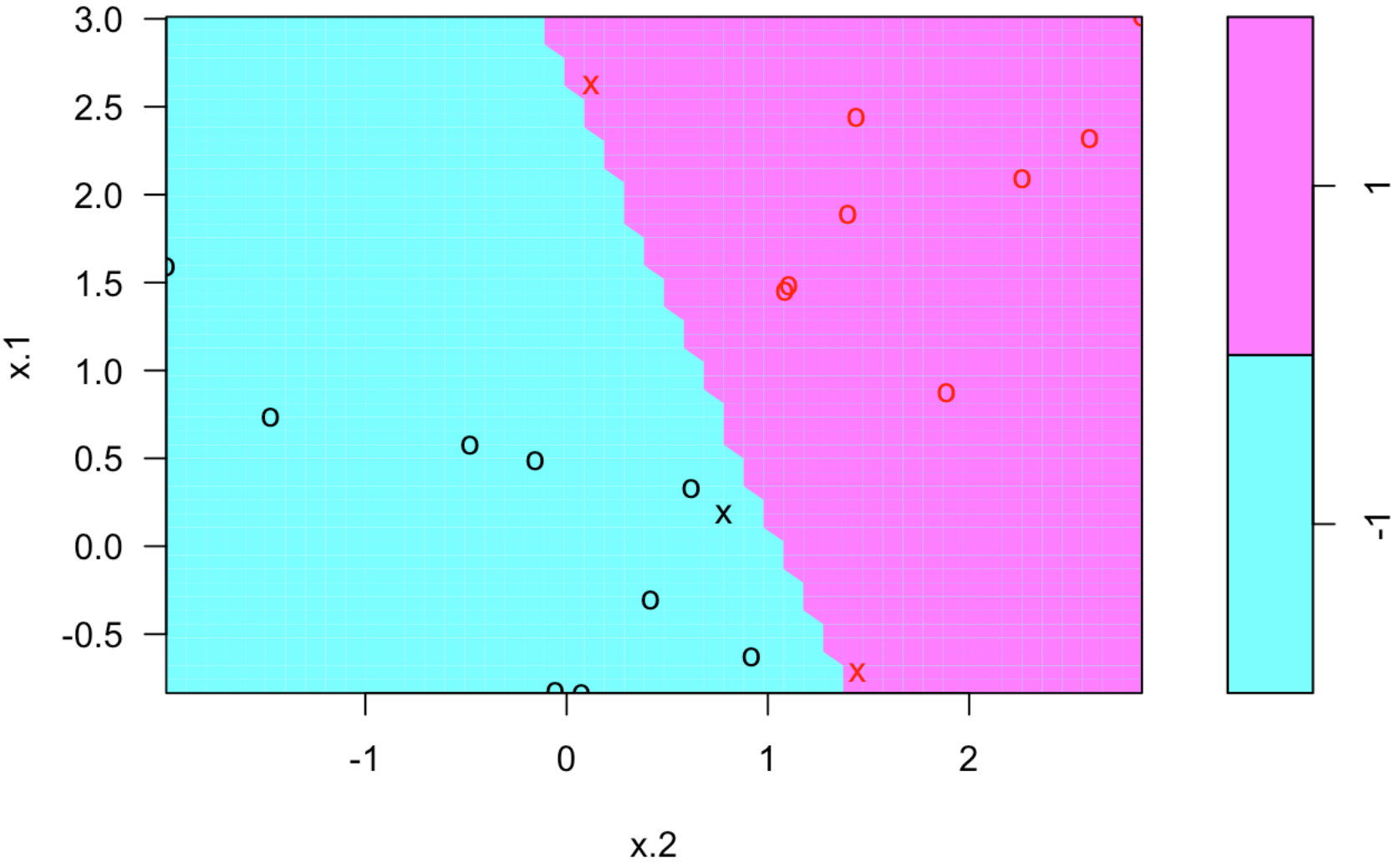
```
x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)
```




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

```
##
## 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
```

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

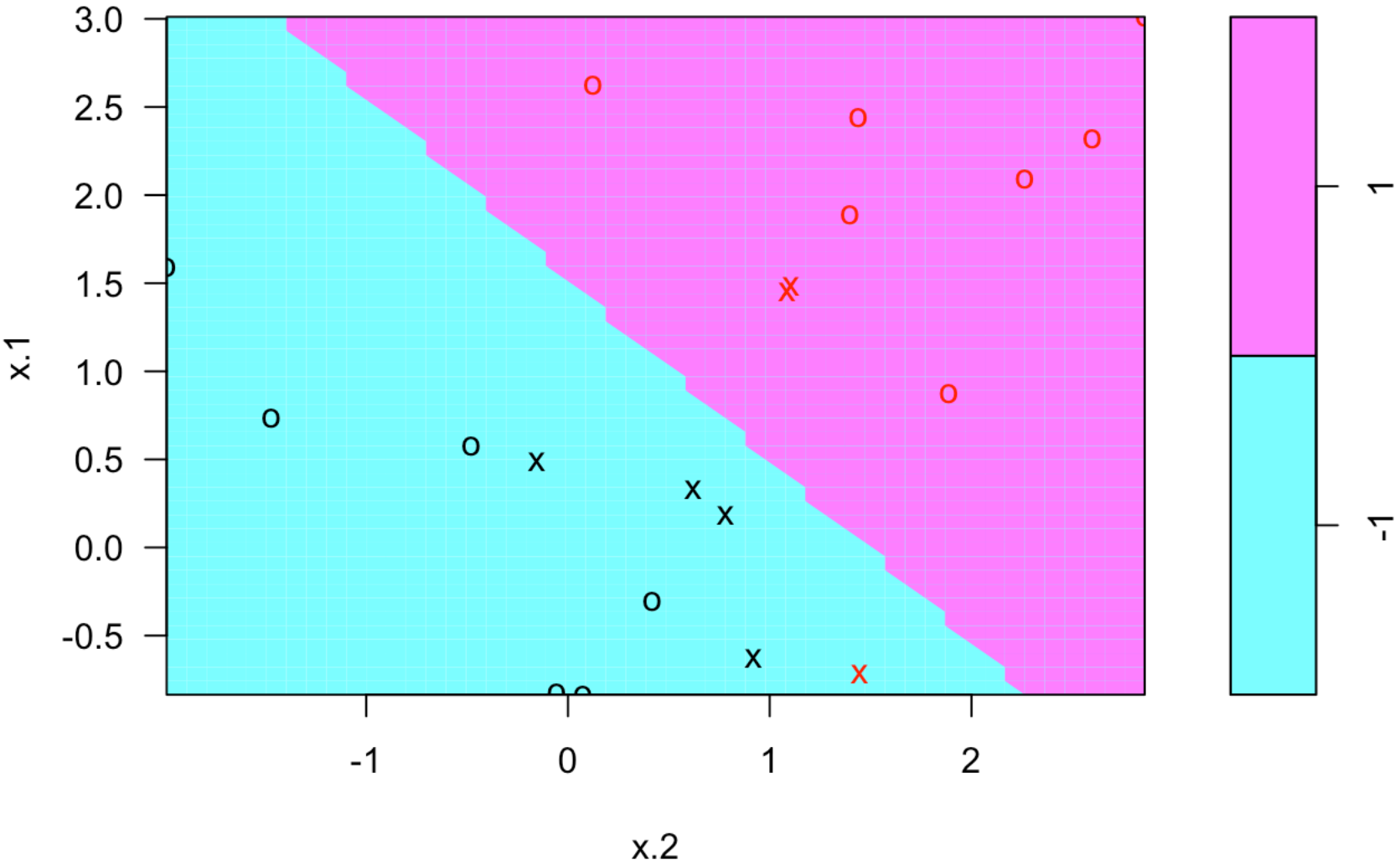


Now try with a smaller value of cost

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

```
##
## 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:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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


SVM classification plot
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
x.1
-1
0
1
2
x.2
1
-1