harinris_HW6_lab

Harin Rishabh

2023-12-10

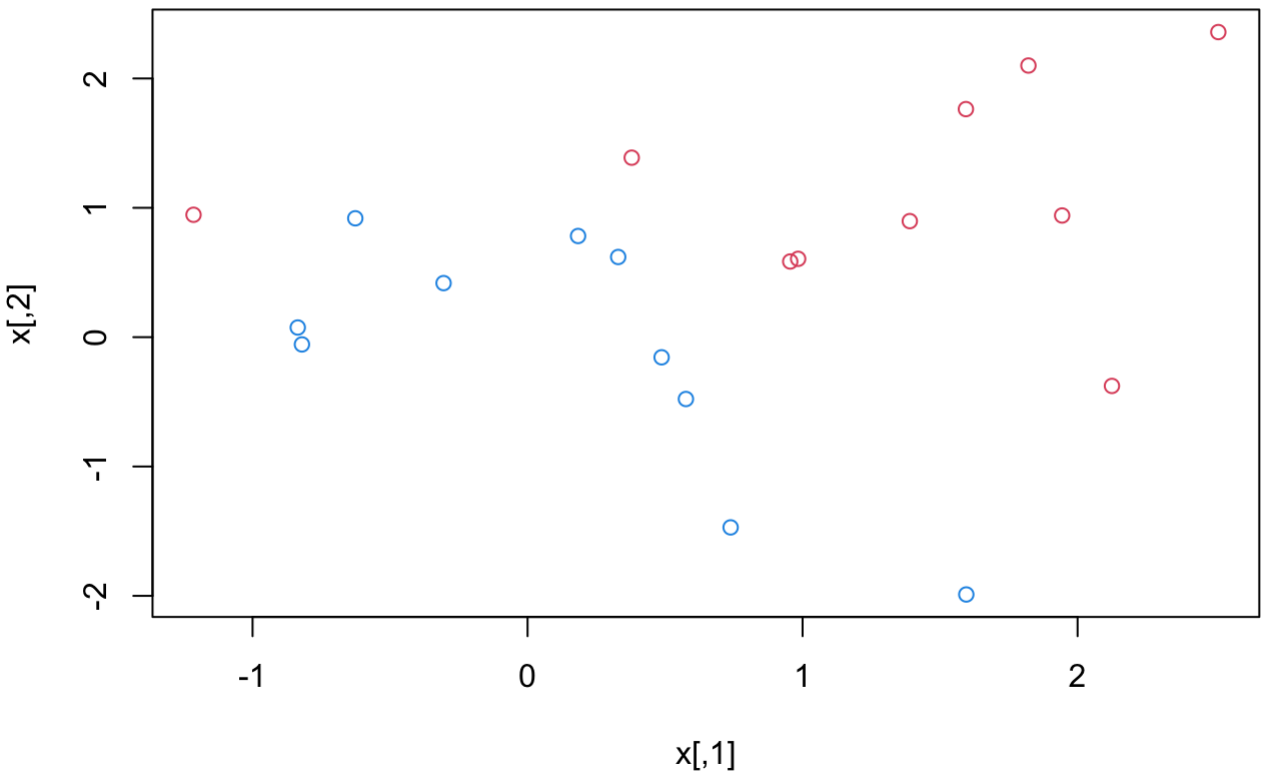
9.6 Lab: Support Vector Machines

9.6.1 Support Vector Classifier

```
library(e1071)
```

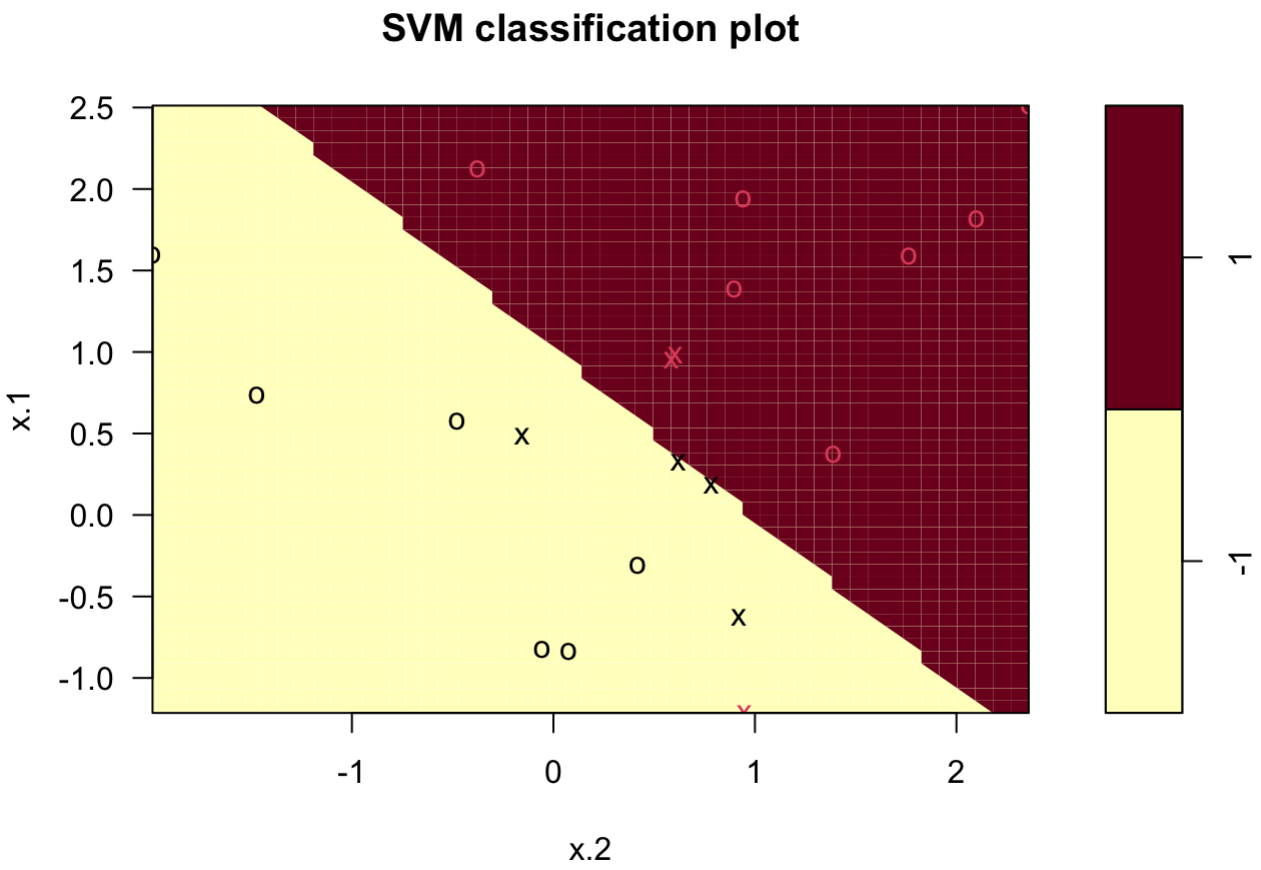
Using the svm() function to fit the support vector classifier for a given value of the cost parameter. We begin by generating the observations, which belong to two classes, and checking whether the classes are linearly separable.

```
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))
```



We now create a data frame with the response coded as a factor.

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



We can inspect the support vectors with the index component of the classifier returned from svm().

```
svmfit$index
```

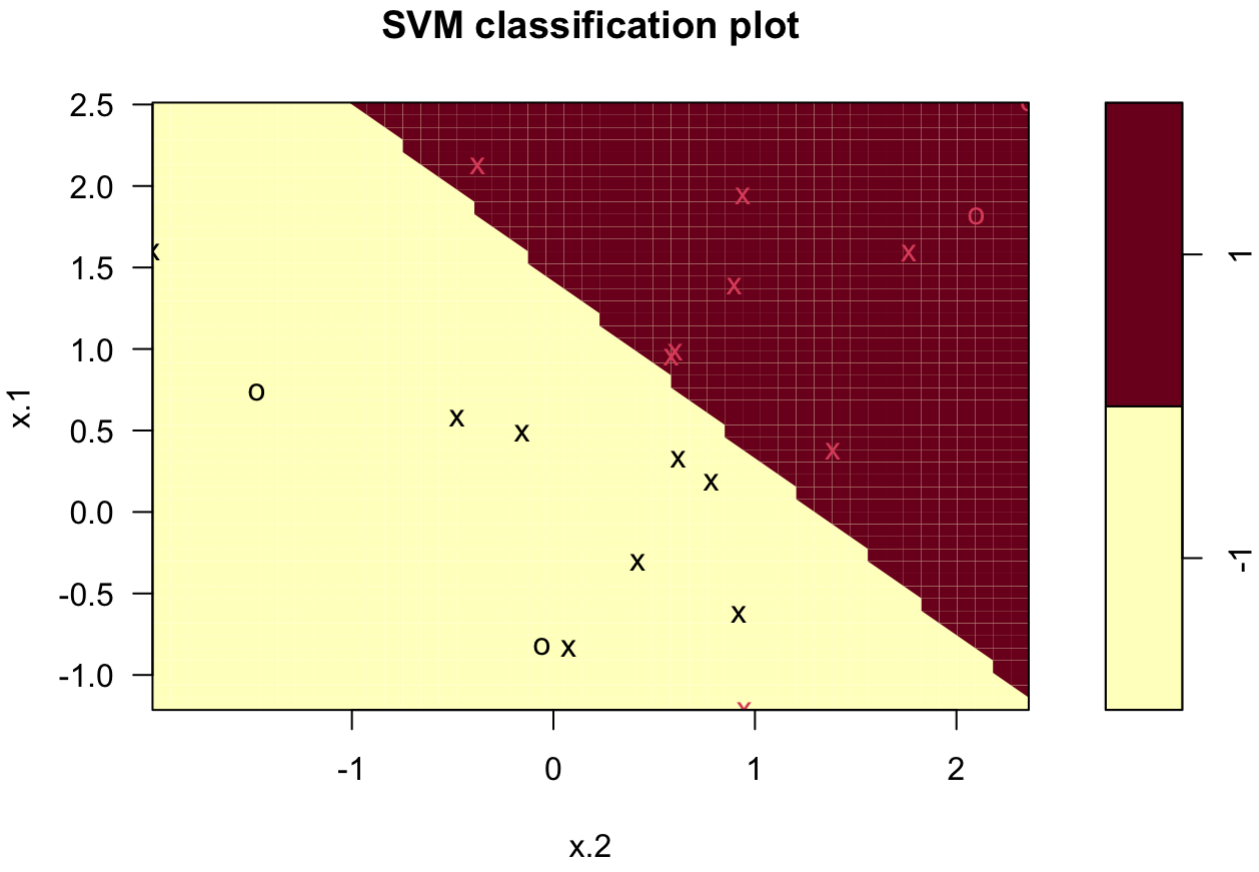
```
## [1]  1  2  5  7 14 16 17
```

```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

hange the cost parameter and re-run the classifier to see how the support vectors change.

```
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
```

We can run cross-validation on a range of values for the cost parameter using the tune() function.

```
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
##     5
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.65  0.4743416
## 2 1e-02  0.65  0.4743416
## 3 1e-01  0.15  0.3374743
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.10  0.2108185
## 6 1e+01  0.10  0.2108185
## 7 1e+02  0.15  0.2415229
```

The best performing model is stored in the best.model component of the object returned from tune().

```
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:  5
##
## Number of Support Vectors:  8
##
##  ( 4 4 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

We can also classify a different set of observations using the best model obtained from tune(). First we create a new set of observations with the rnorm() function.

```
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))
```

We can then use the predict() function to classify the observations in testdat.

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

```
##        truth
## predict -1  1
##      -1 10  2
##      1   1  7
```

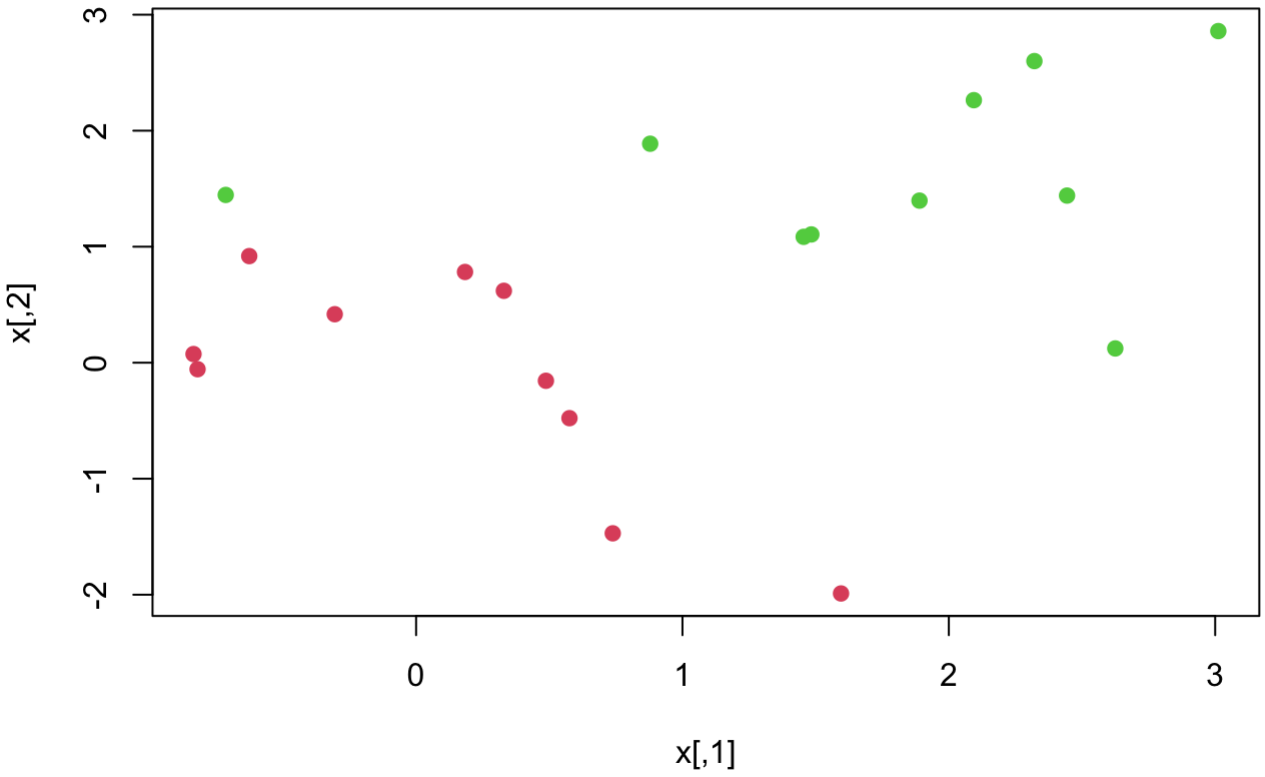
We can also try a different value for the cost parameter and train the classifier again.

```
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  5
##      1   0  4
```

We can linearly separate the two classes and plot the observations.

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

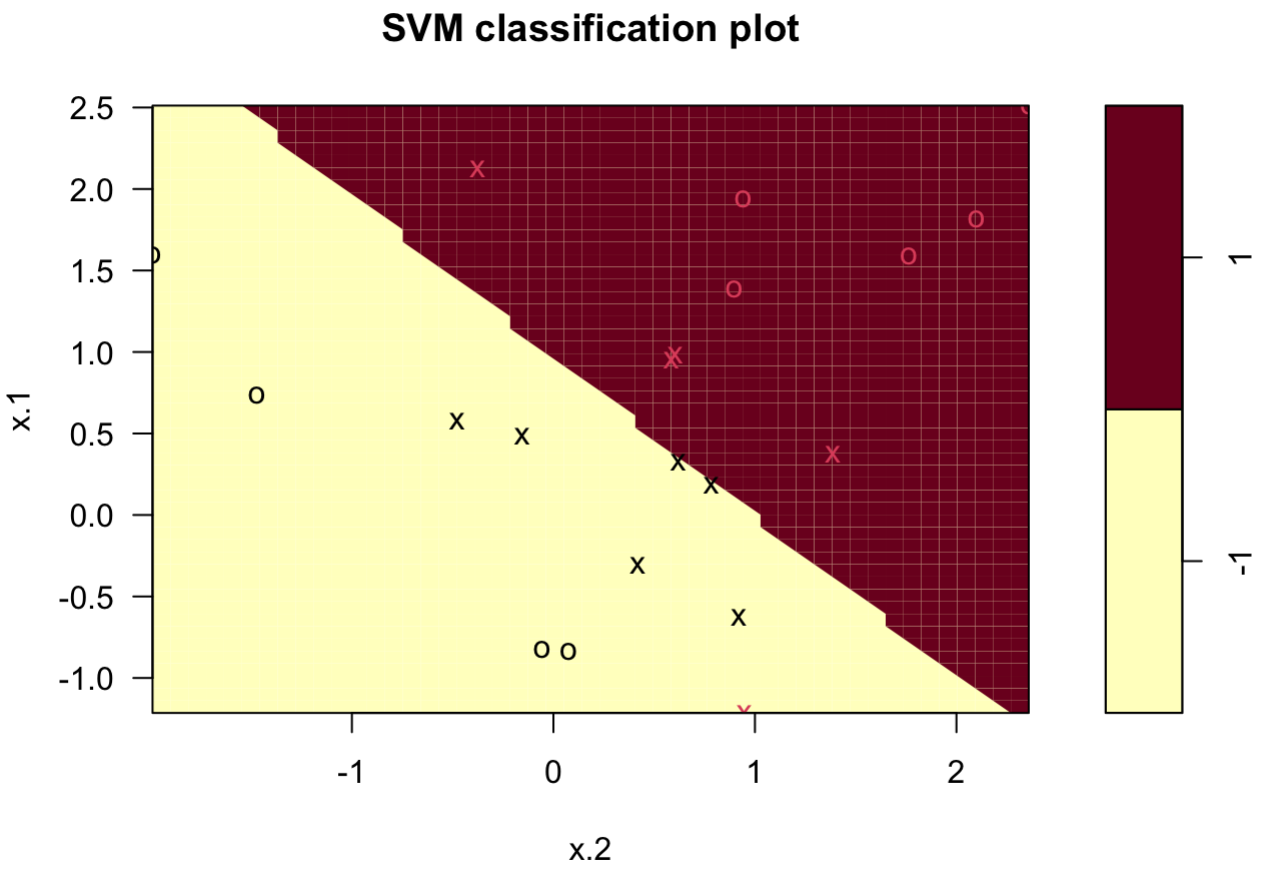


We can also try with a smaller cost value.

```
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
##
## Number of Support Vectors:  11
##
##  ( 6 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

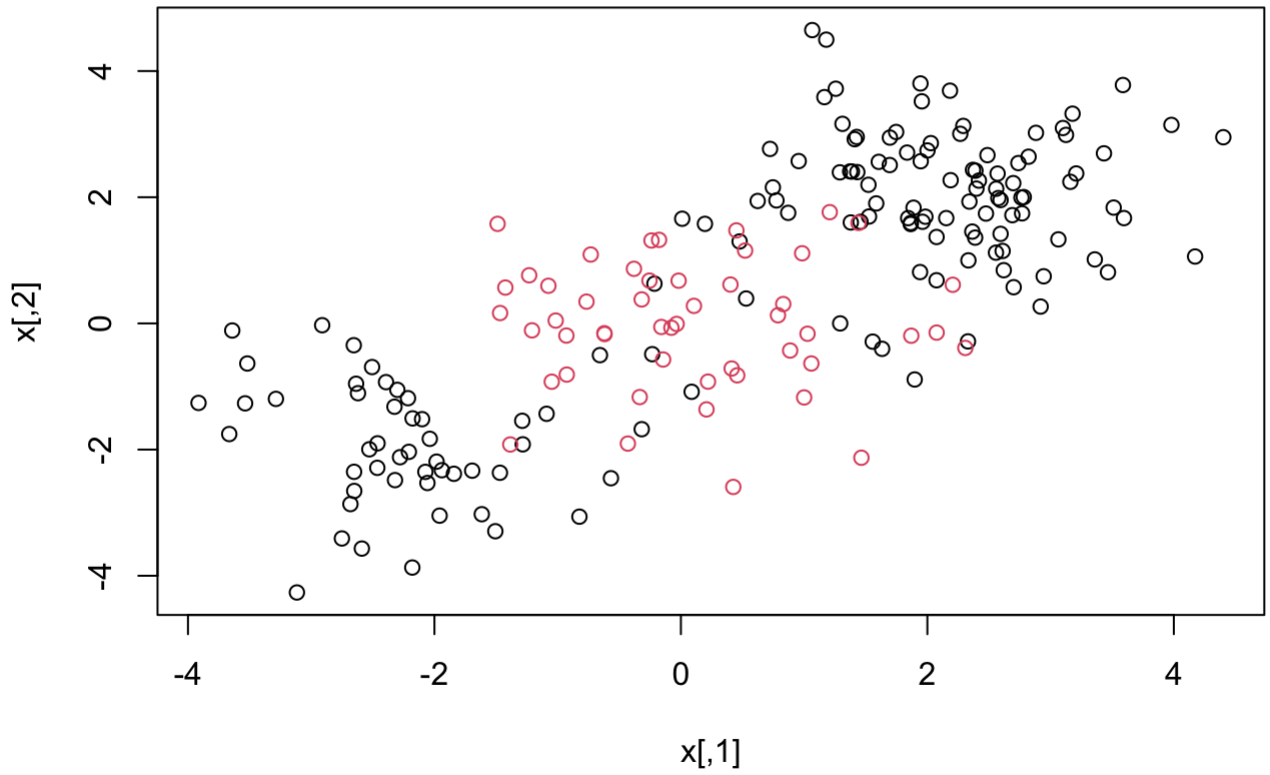
```
plot(svmfit, dat)
```



9.6.2 Support Vector Machine

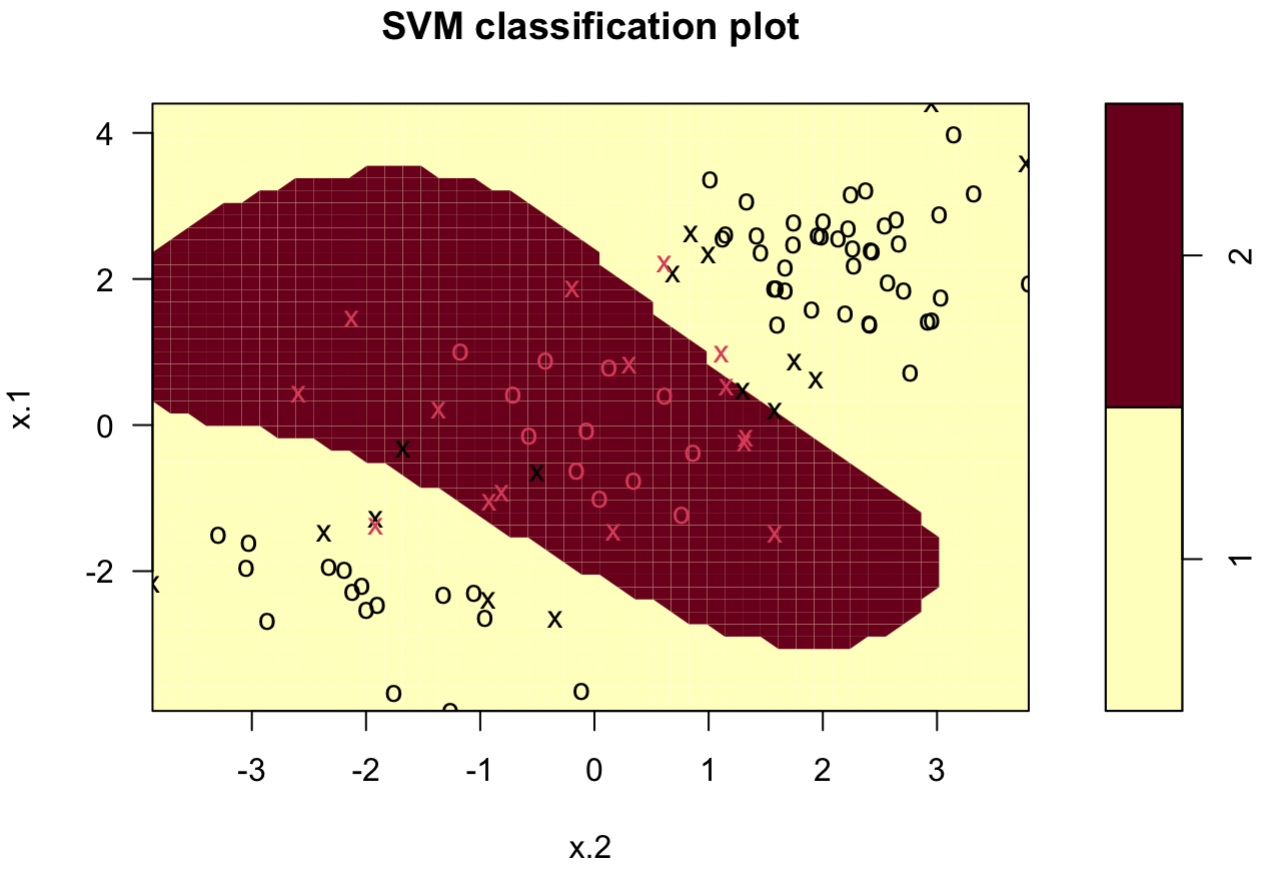
We begin by generating data with non-linear class boundaries. Plotting the data makes it clear that the class boundary is indeed nonlinear.

```
set.seed(1)
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))
plot(x, col = y)
```



We split the data into training and test subsets and run the SVM classifier with kernel = “radial” parameter.

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

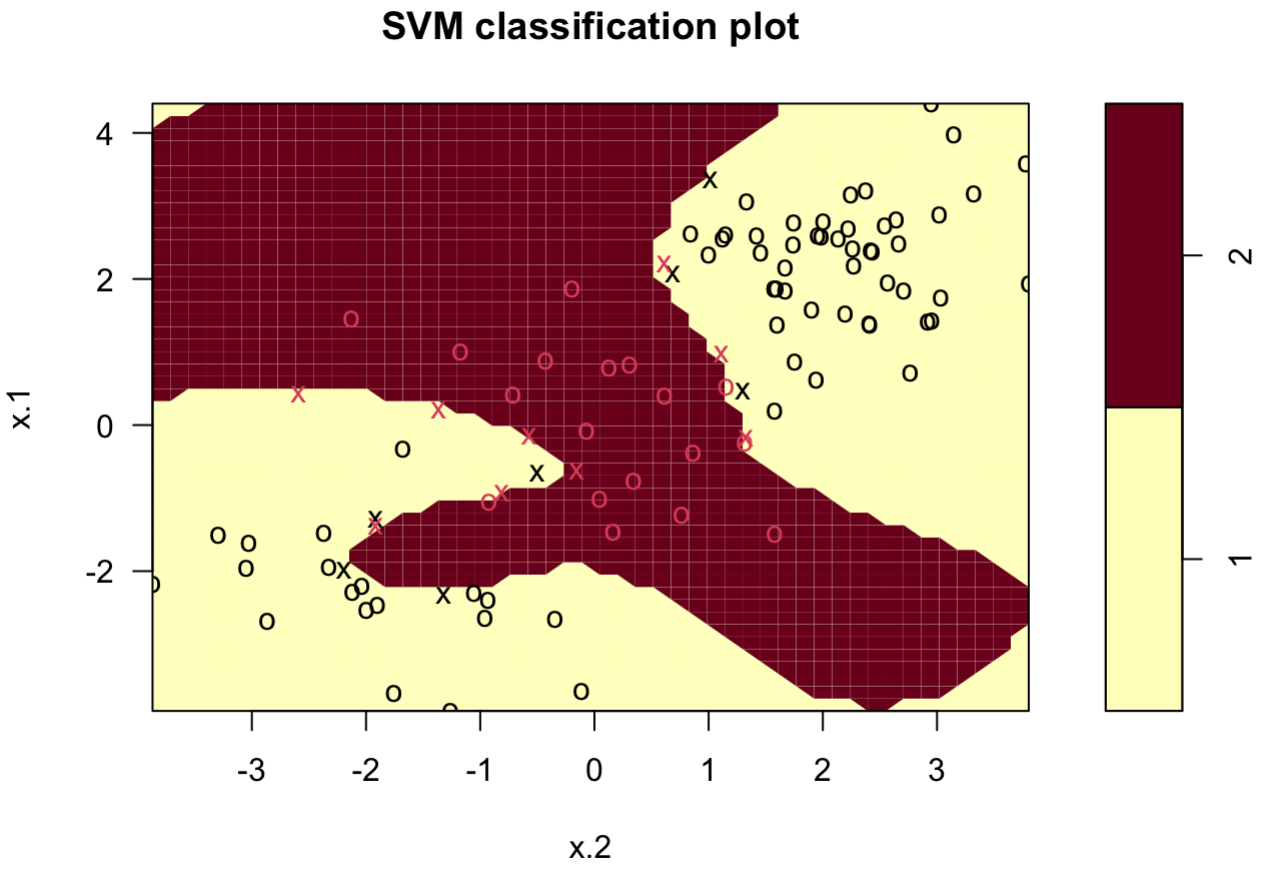



```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

We can use a larger value for the cost parameter and see if it reduces the training errors.

```
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1e+05)
plot(svmfit, dat[train, ])
```



We can run cross-validation using the tune() function.

```
tune.out <- tune(svm, y ~ ., data = dat[train, ], kernel = "radial", ranges = list(co
st = 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   0.5
##
## - best performance: 0.06
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.27 0.14944341
## 2  1e+00   0.5  0.06 0.08432740
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.10 0.09428090
## 5  1e+03   0.5  0.12 0.10327956
## 6  1e-01   1.0  0.18 0.16865481
## 7  1e+00   1.0  0.07 0.09486833
## 8  1e+01   1.0  0.09 0.09944289
## 9  1e+02   1.0  0.10 0.08164966
## 10 1e+03   1.0  0.09 0.09944289
## 11 1e-01   2.0  0.26 0.15776213
## 12 1e+00   2.0  0.08 0.11352924
## 13 1e+01   2.0  0.12 0.12292726
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.13 0.10593499
## 16 1e-01   3.0  0.27 0.13374935
## 17 1e+00   3.0  0.09 0.11005049
## 18 1e+01   3.0  0.10 0.13333333
## 19 1e+02   3.0  0.13 0.10593499
## 20 1e+03   3.0  0.15 0.10801234
## 21 1e-01   4.0  0.27 0.13374935
## 22 1e+00   4.0  0.09 0.12866839
## 23 1e+01   4.0  0.10 0.13333333
## 24 1e+02   4.0  0.15 0.13540064
## 25 1e+03   4.0  0.17 0.13374935
```

We can predict the classes on the test subset and examine the number of observations misclassified.

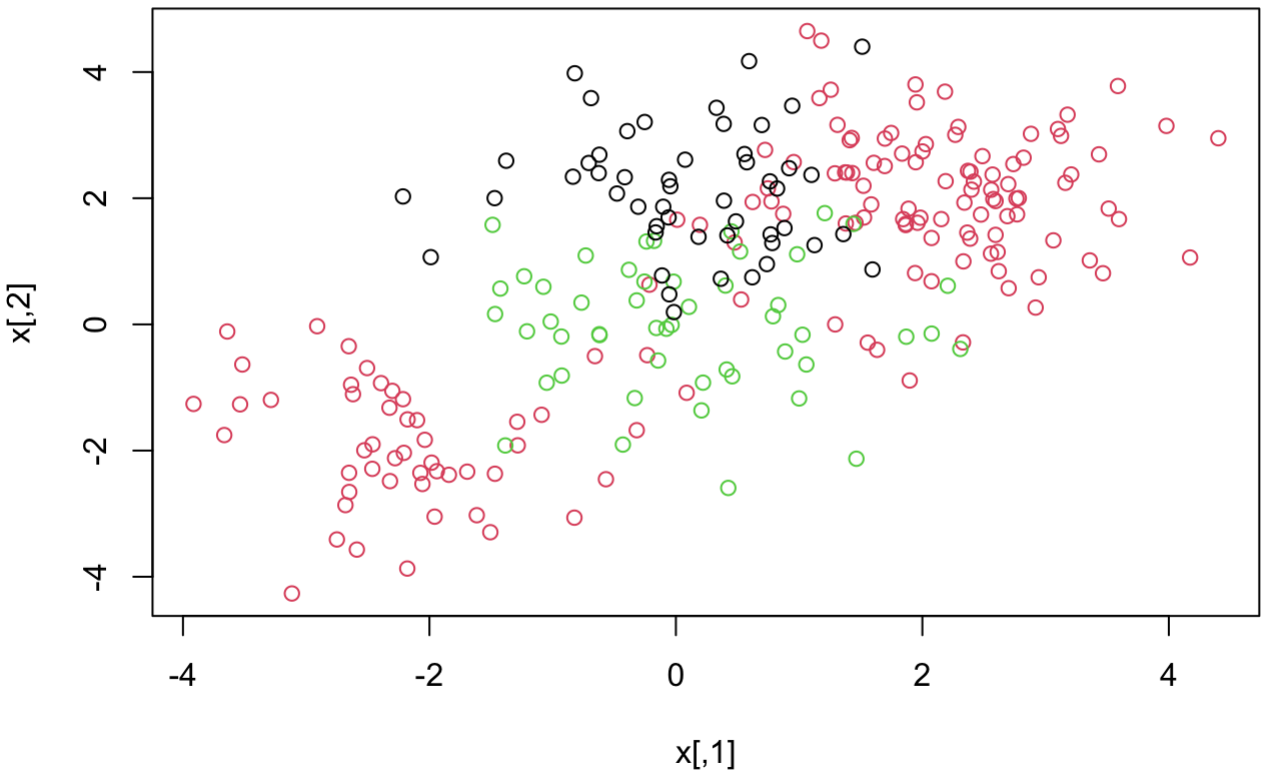
```
table(true = dat[-train, "y"], pred = predict(tune.out$best.model, newdata = dat[-tra
in, ]))
```

```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

9.6.4 SVM with Multiple Classes

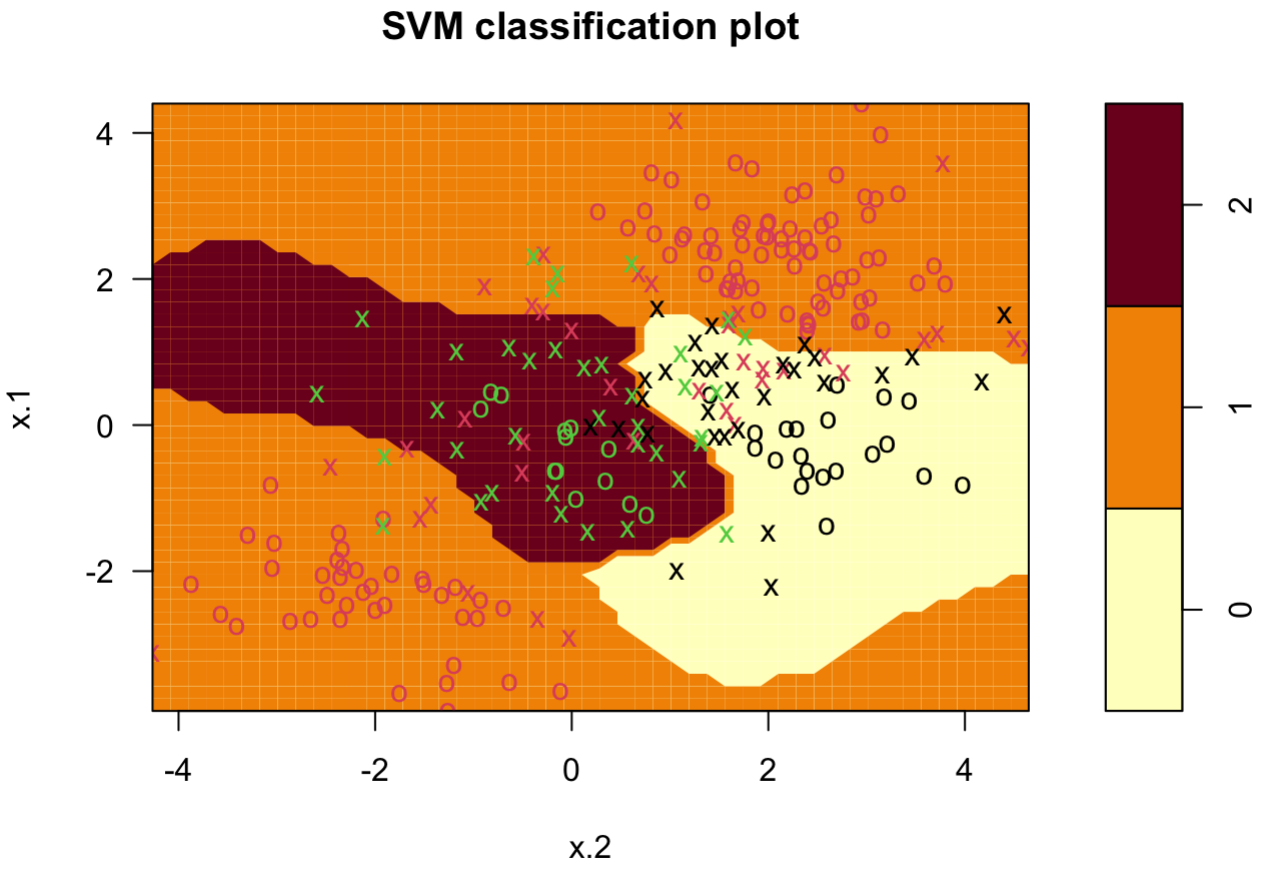
The svm() function can also be used to classify observations from multiple-classes.

```
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))
```



The svm() function now will perform multi-class classification since the dataset we generated now has three class labels.

```
svmfit <- svm(y ~ ., data = dat, kernel = "radial", cost = 10, gamma = 1)
plot(svmfit, dat)
```


SVM classification plot
x.1
x.2
4
2
0
-2
-4
-2
0
2
4
2
1
0