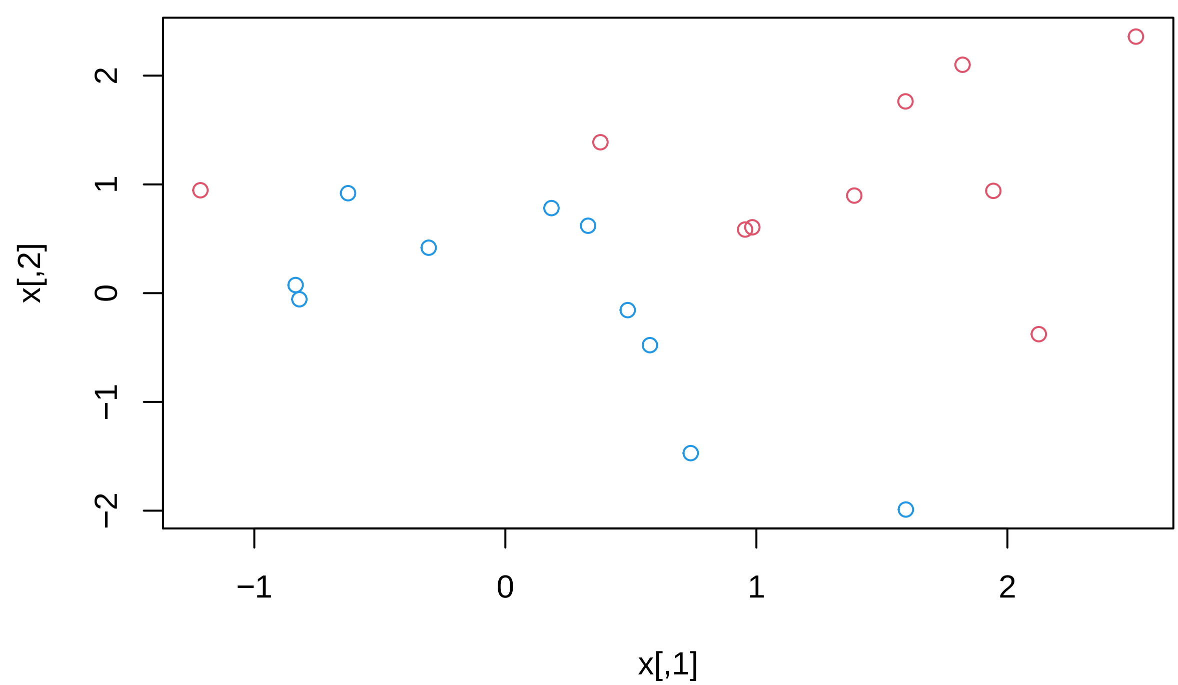
SVM example 3 (ISLR)

Abhirup Sen

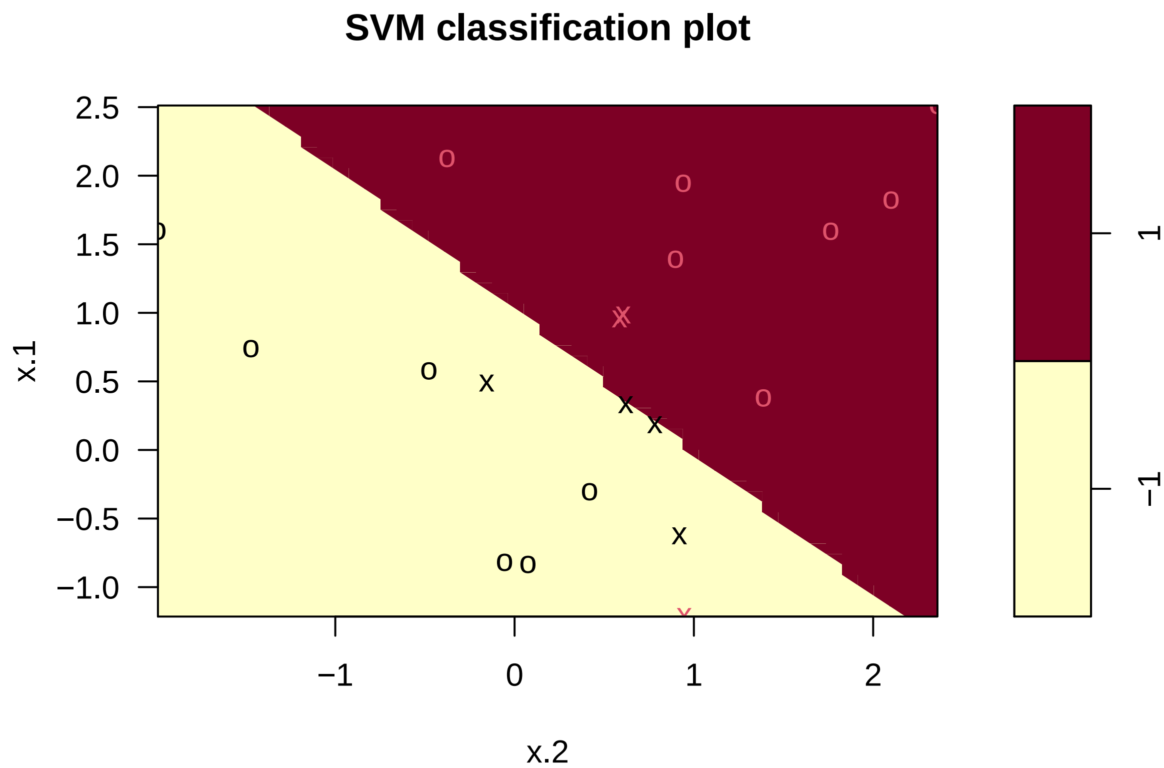
24/05/2021

```
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))
```



Now lets fit the Support vector classifier.

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




```
svmfit$index
```

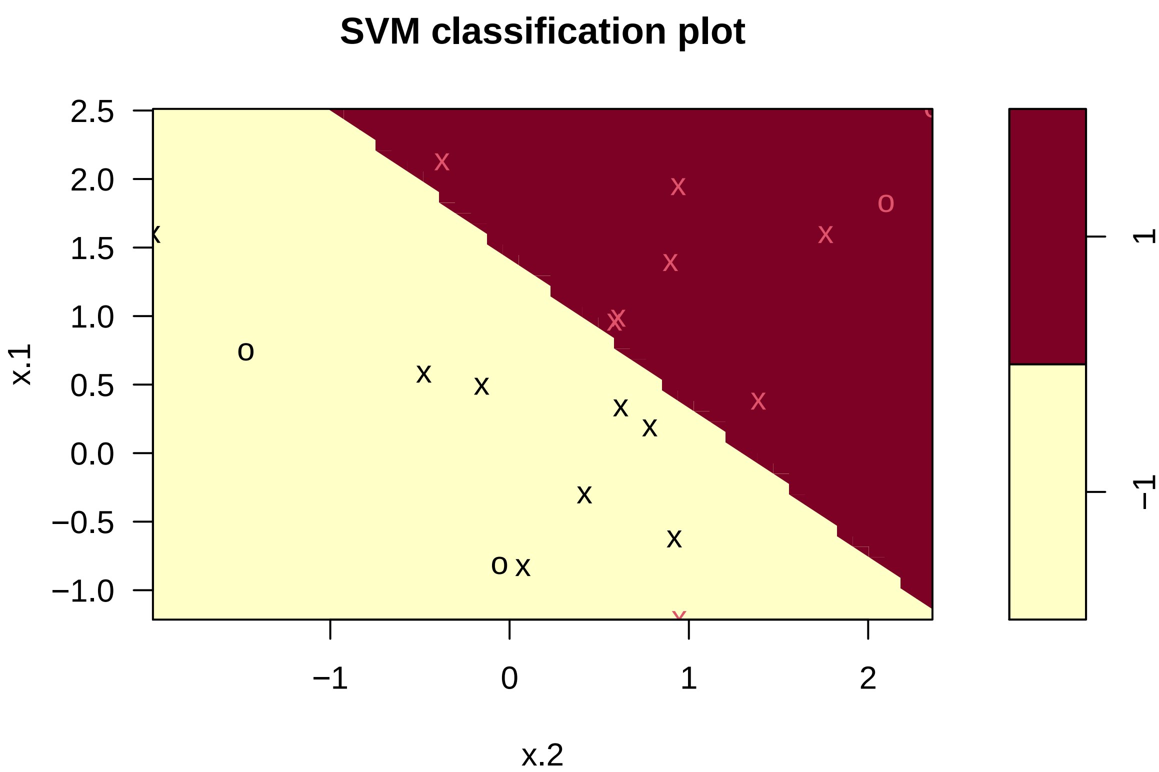
```
## [1]  1  2  5  7 14 16 17
```

```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = data, 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
```

What if we use a smaller value of cost

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




```
svmfit$index
```

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

The tune() function aspart of library e1071 ; performs crossValidation, 10 fold cross validation.

```
set.seed(1)
tune.out = tune(svm, y~., data = data, 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: 0.05
##
## - Detailed performance results:
##      cost error dispersion
## 1   0.001  0.55  0.4377975
## 2   0.010  0.55  0.4377975
## 3   0.100  0.05  0.1581139
## 4   1.500  0.15  0.2415229
## 5  10.000  0.15  0.2415229
## 6 100.000  0.15  0.2415229
```

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

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = data, 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
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
xtest = matrix(rnorm(20*2), ncol=2)
ytest = sample(c(-1,1),20,rep = TRUE)
xtest[ytest==1,]==xtest[ytest==1,]+1
```

```
##        [,1]  [,2]
##  [1,] FALSE FALSE
##  [2,] FALSE FALSE
##  [3,] FALSE FALSE
##  [4,] FALSE FALSE
##  [5,] FALSE FALSE
##  [6,] FALSE FALSE
##  [7,] FALSE FALSE
##  [8,] FALSE FALSE
##  [9,] FALSE FALSE
```

```
testdata = data.frame(x=xtest,y=as.factor(ytest))
```

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

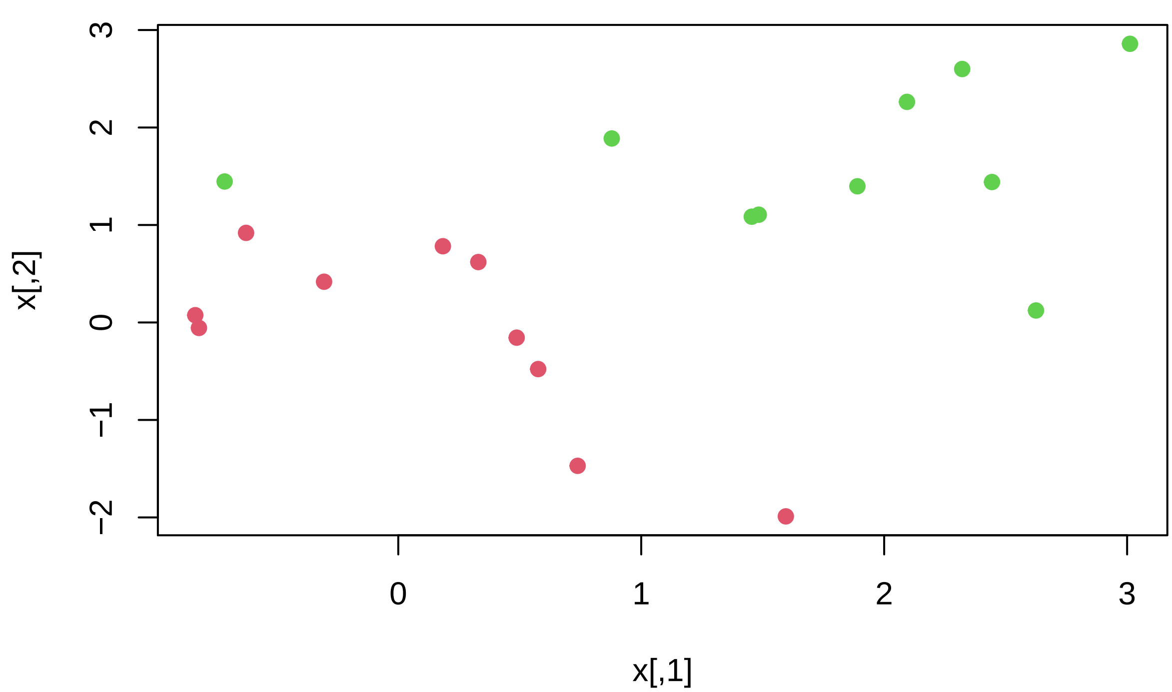
```
##          truth
## predict -1 1
##      -1  9 9
##      1   2 0
```

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

```
##          truth
## predict -1  1
##      -1 11  9
##      1   0  0
```

Now if the 2 classes are linearly seperable

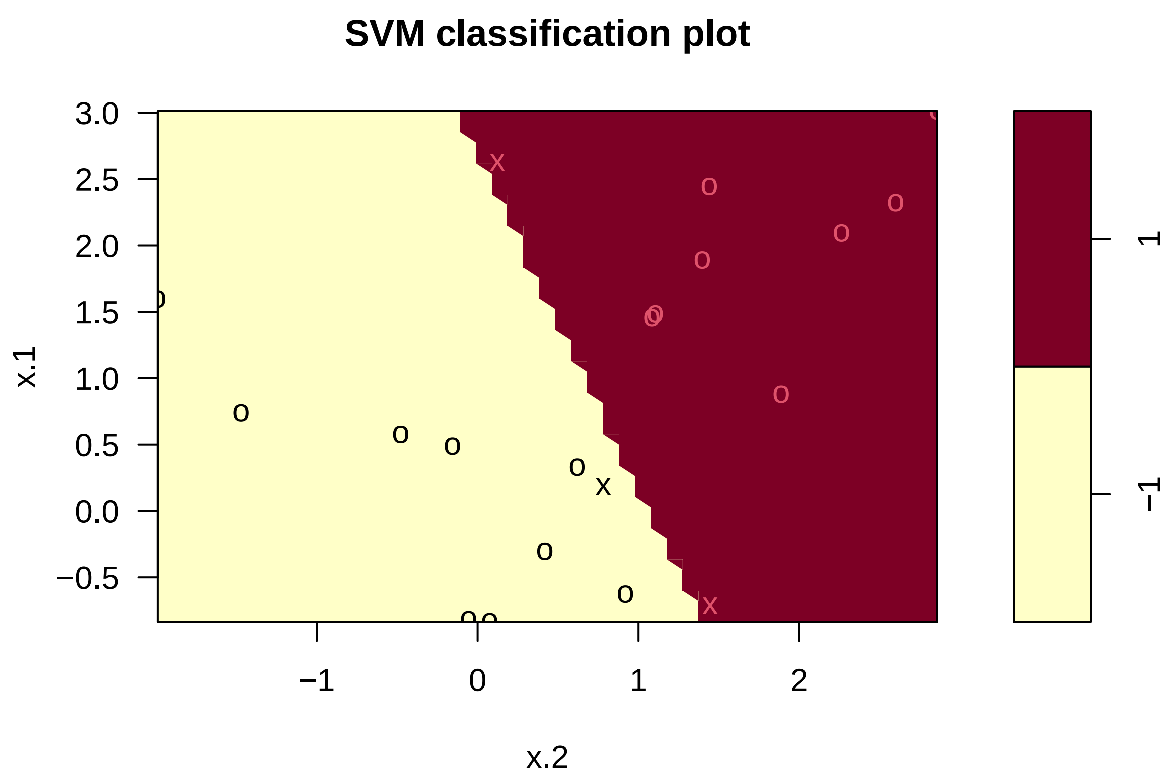
```
x[y==1,]=x[y==1,]+0.5
plot(x,col = (y+5)/2, pch = 19)
```



```r
data = data.frame(x=x,y=as.factor(y))
svmfit = svm(y~., data = data, kernel = "linear", cost = 1e+05)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = data, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

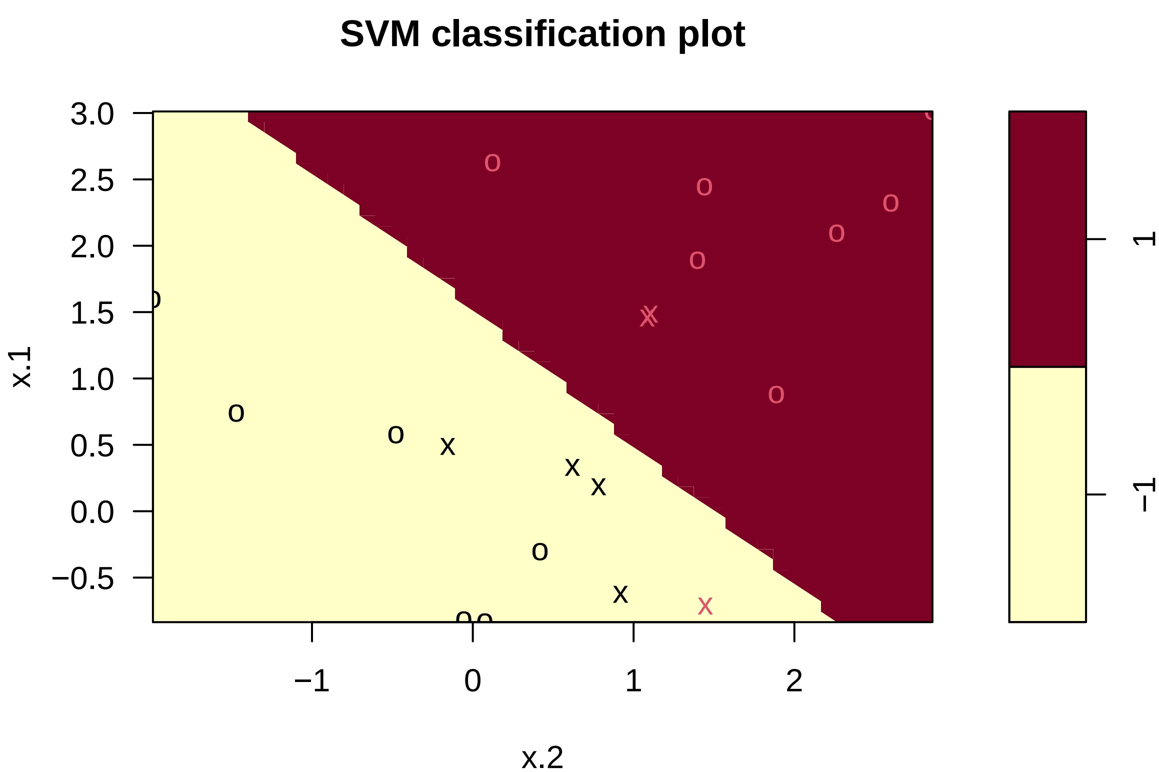
```r
plot(svmfit, data)
```



```r
#We can also try with a a smaller cost value.
svmfit <- svm(y ~ ., data = data, kernel = "linear", cost = 1)
summary(svmfit)
```

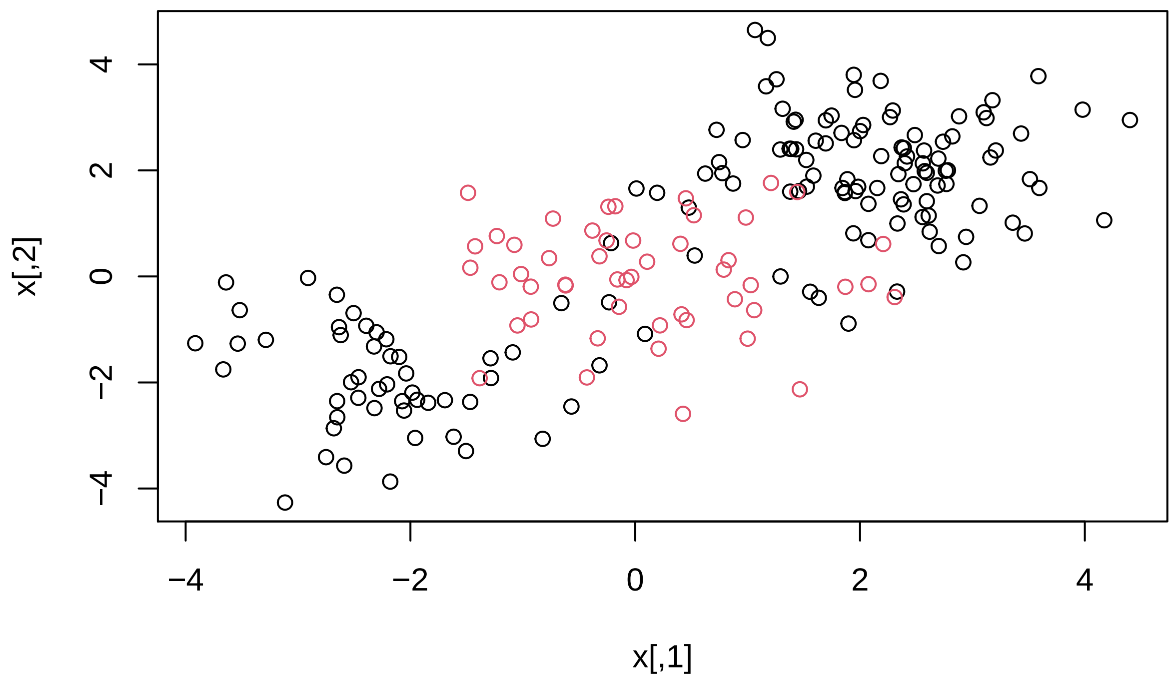
```
##
## Call:
## svm(formula = y ~ ., data = data, kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(svmfit, data)
```



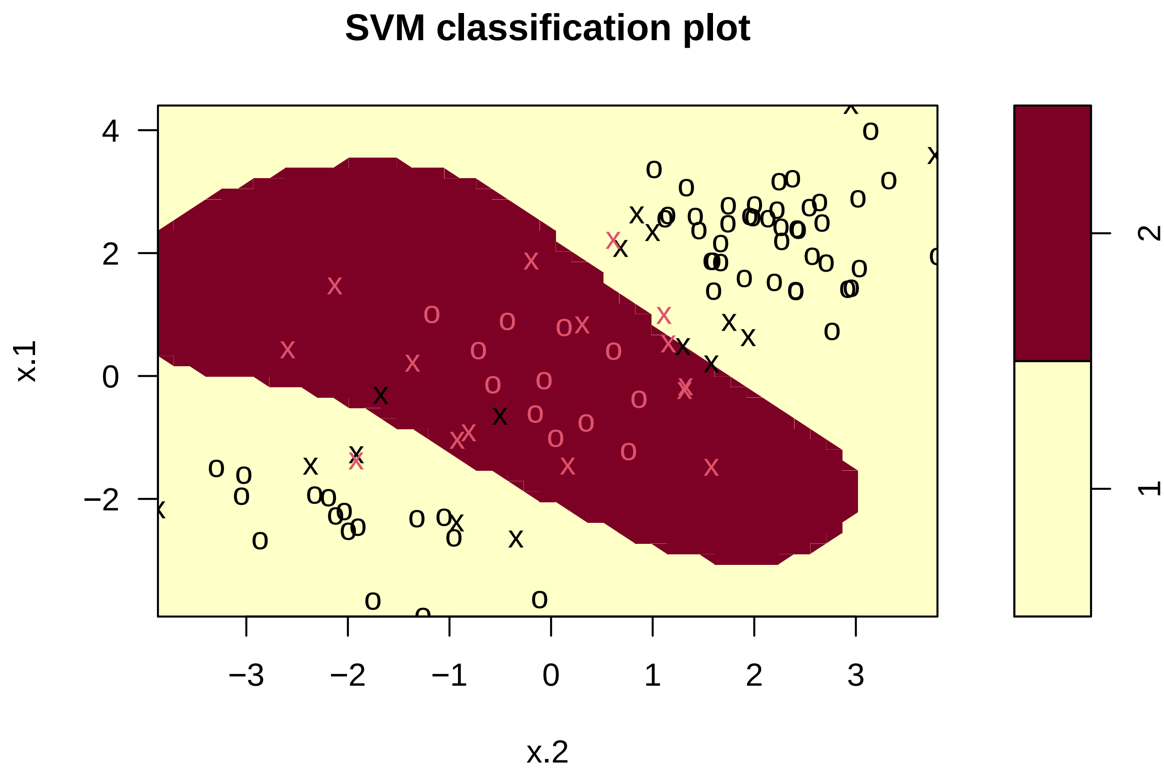
```r
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
```

```r
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.

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



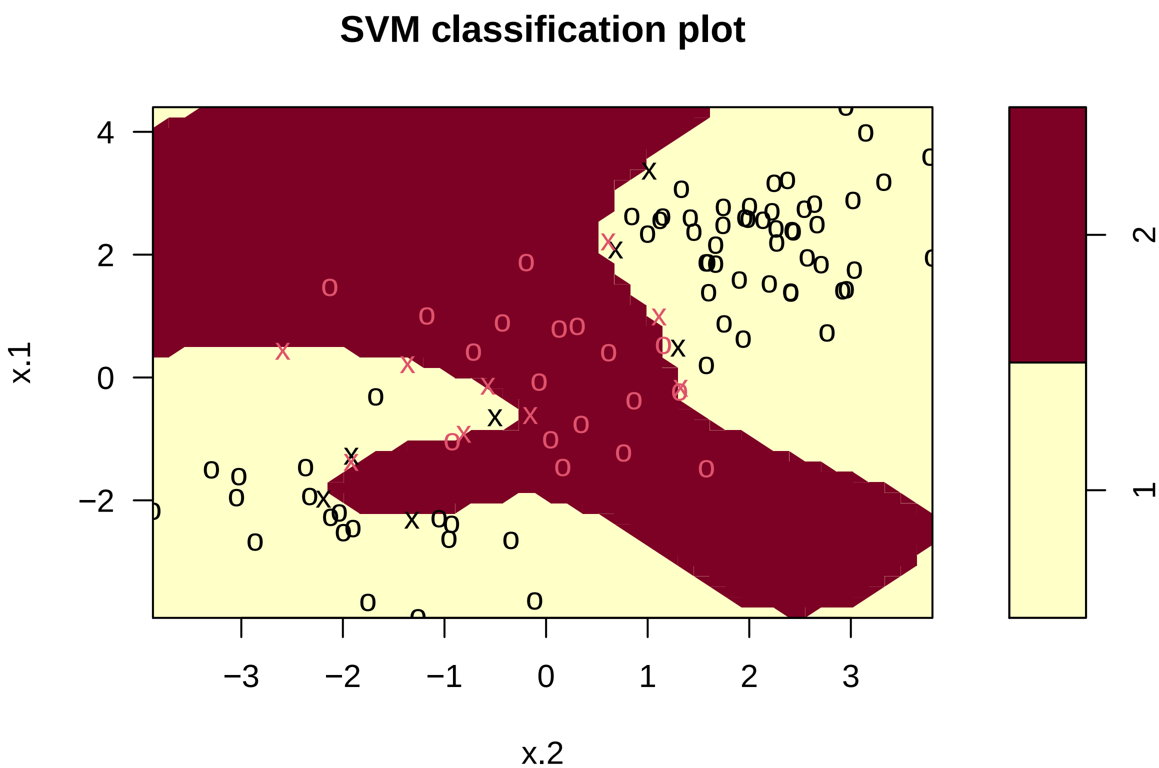
We can examine the model fit with the summary() function.

```
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.

```
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ], kernel = "radial", ranges = list(cost = c(0.1, 1, 10,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.07
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.26 0.15776213
## 2  1e+00   0.5  0.07 0.08232726
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.14 0.15055453
```

```
## 5  1e+03   0.5  0.11 0.07378648
## 6  1e-01   1.0  0.22 0.16193277
## 7  1e+00   1.0  0.07 0.08232726
## 8  1e+01   1.0  0.09 0.07378648
## 9  1e+02   1.0  0.12 0.12292726
## 10 1e+03   1.0  0.11 0.11005049
## 11 1e-01   2.0  0.27 0.15670212
## 12 1e+00   2.0  0.07 0.08232726
## 13 1e+01   2.0  0.11 0.07378648
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.16 0.13498971
## 16 1e-01   3.0  0.27 0.15670212
## 17 1e+00   3.0  0.07 0.08232726
## 18 1e+01   3.0  0.08 0.07888106
## 19 1e+02   3.0  0.13 0.14181365
## 20 1e+03   3.0  0.15 0.13540064
## 21 1e-01   4.0  0.27 0.15670212
## 22 1e+00   4.0  0.07 0.08232726
## 23 1e+01   4.0  0.09 0.07378648
## 24 1e+02   4.0  0.13 0.14181365
## 25 1e+03   4.0  0.15 0.13540064
```

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[-train, ]))
```

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

##9.6.3 ROC Curves##

We use the ROCR package to produce ROC curves on the predictions from the test subset.

```
library(ROCR)
```

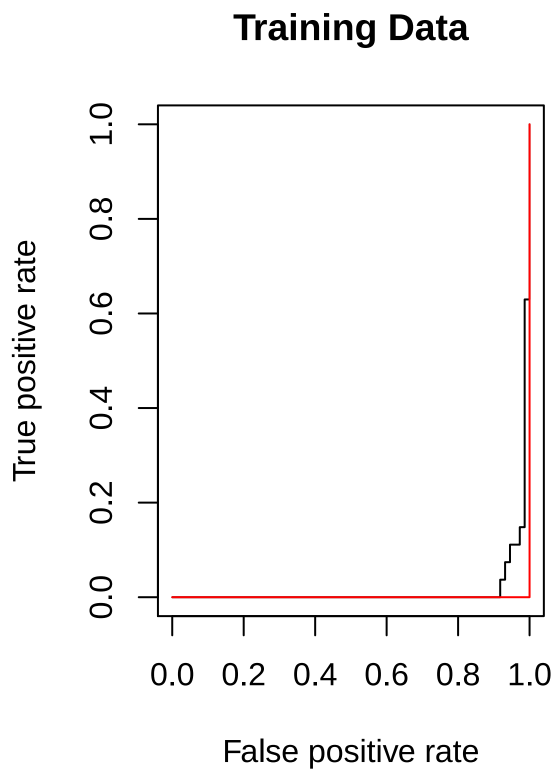
```
rocplot <- function(pred, truth, ...) {
    predob <- prediction(pred, truth)
    perf <- performance(predob, "tpr", "fpr")
    plot(perf, ...)
}
```

Instead of getting the class labels, we can also get fitted values from svm() using decision.values = TRUE parameter.

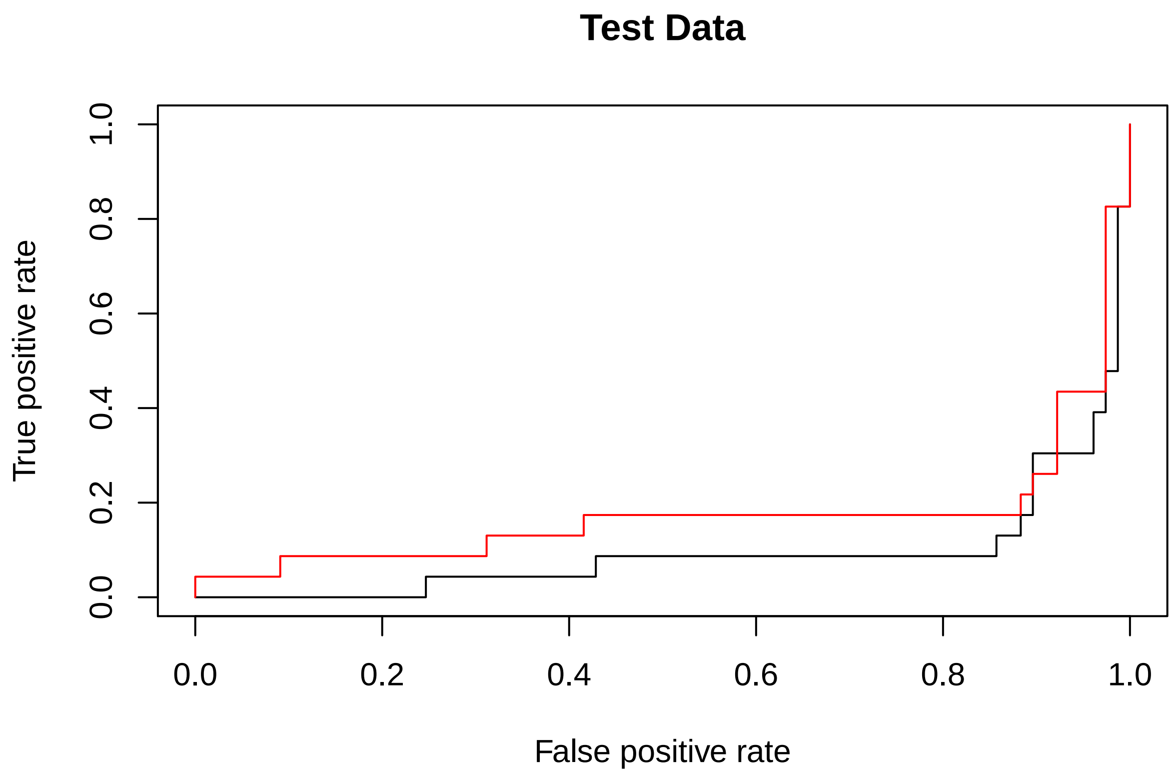
```
svmfit.opt <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 2, cost = 1, decision.values =
fitted <- attributes(predict(svmfit.opt, dat[train, ], decision.values = TRUE))$decision.values
```

```
#We generate the ROC curve with the rocplot() function. We can also change the value of (\gamma) and se
par(mfrow = c(1, 2))
rocplot(fitted, dat[train, "y"], main = "Training Data")
```

```r
svmfit.flex <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 50, cost = 1, decision.values
fitted <- attributes(predict(svmfit.flex, dat[train, ], decision.values = T))$decision.values
rocplot(fitted, dat[train, "y"], add = T, col = "red")
```

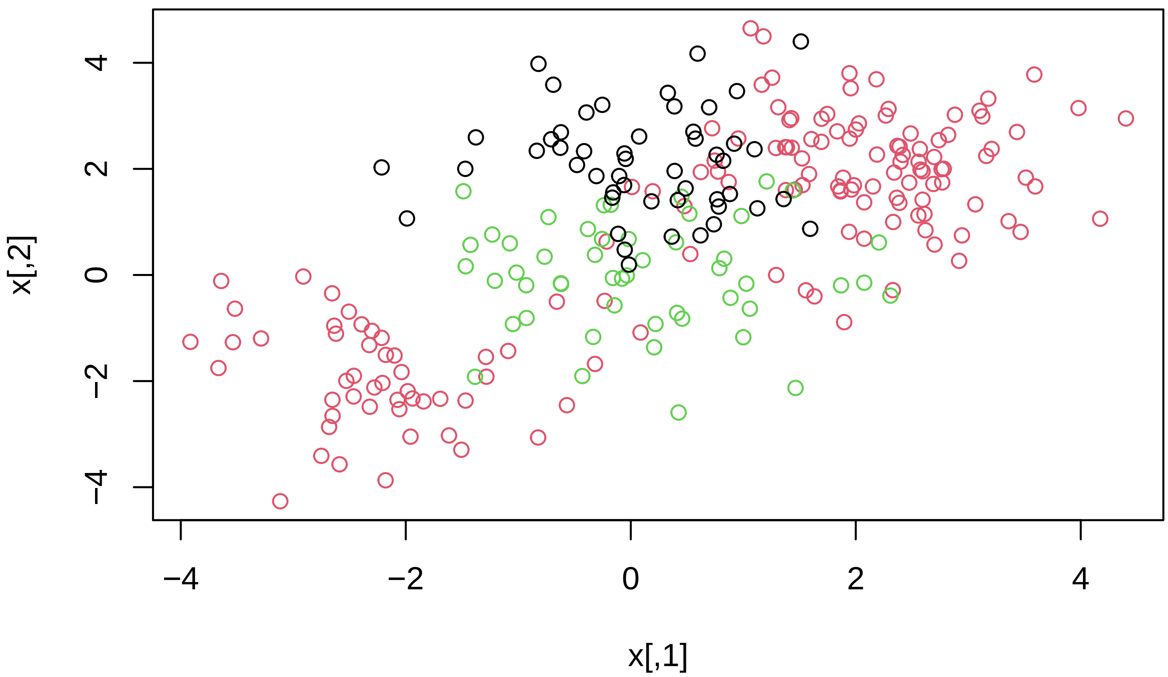



```r
fitted <- attributes(predict(svmfit.opt, dat[-train, ], decision.values = T))$decision.values
rocplot(fitted, dat[-train, "y"], main = "Test Data")
fitted <- attributes(predict(svmfit.flex, dat[-train, ], decision.values = T))$decision.values
rocplot(fitted, dat[-train, "y"], add = T, col = "red")
```



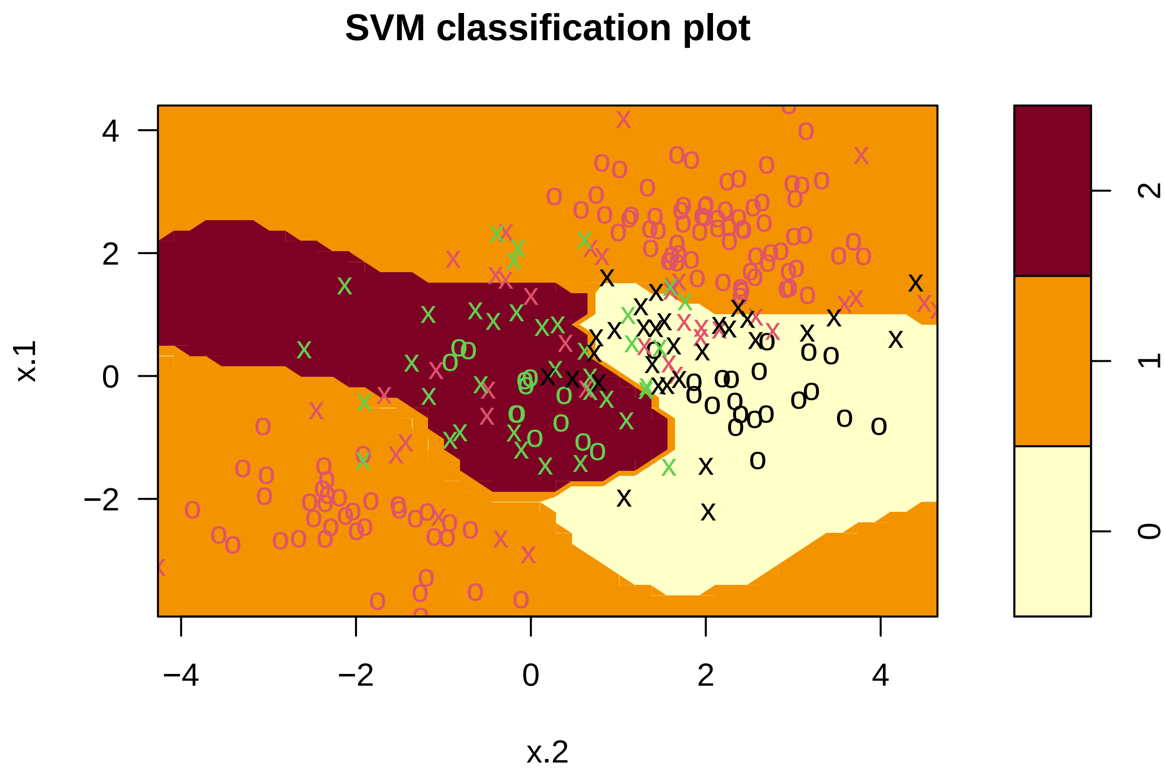
##9.6.4 SVM with Multiple Classes## The svm() function can also be used to classify observations from multiple-classes.

```
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

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



##9.6.5 Application to Gene Expression Data##

```
library(ISLR)
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```
length(Khan$ytrain)
```

```
## [1] 63
```

```
length(Khan$ytest)
```

```
## [1] 20
```

We can examine the class labels associated with the training and test subsets.

```r
table(Khan$ytrain)
```

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

```r
table(Khan$ytest)
```

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

We use a linear kernel and run SVM classifier on the training subset

```r
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
##
## Number of Support Vectors:  58
##
##  ( 20 20 11 7 )
##
##
## Number of Classes:  4
##
## Levels:
##  1 2 3 4
```

```r
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
```

We can then predict the classes on the test subset using the trained classifier.

```r
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
```