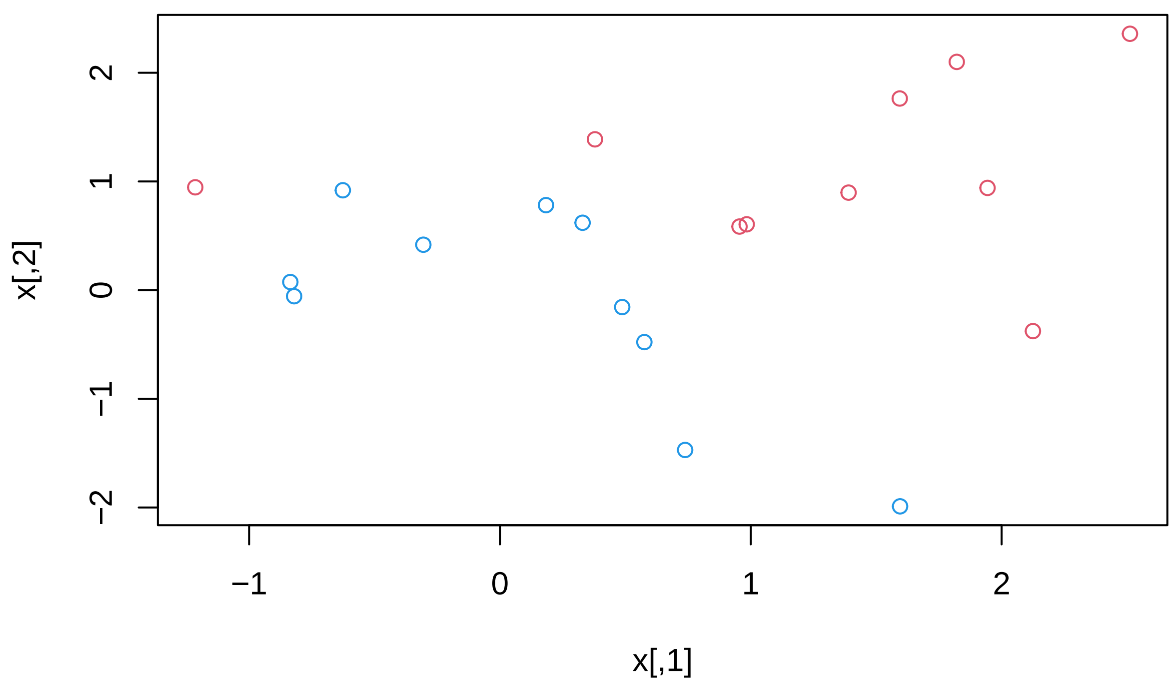
DA8

2025-06-29

9.6.1 Support Vector 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))
```

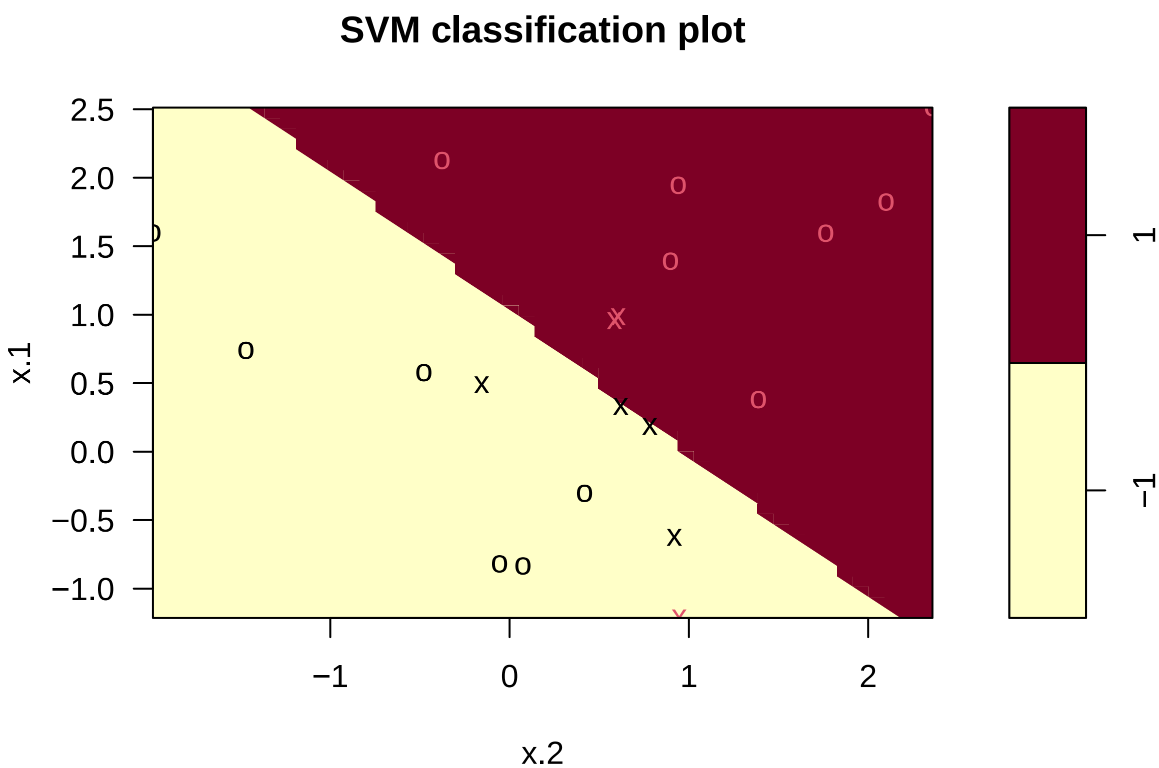


We generated the observations to check whether they are linearly separable. They are not.

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

The svm won't scale each feature to have mean zero or standard deviation one because scale = FALSE

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




```r
svmfit$index
```

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

```r
summary(svmfit)
```

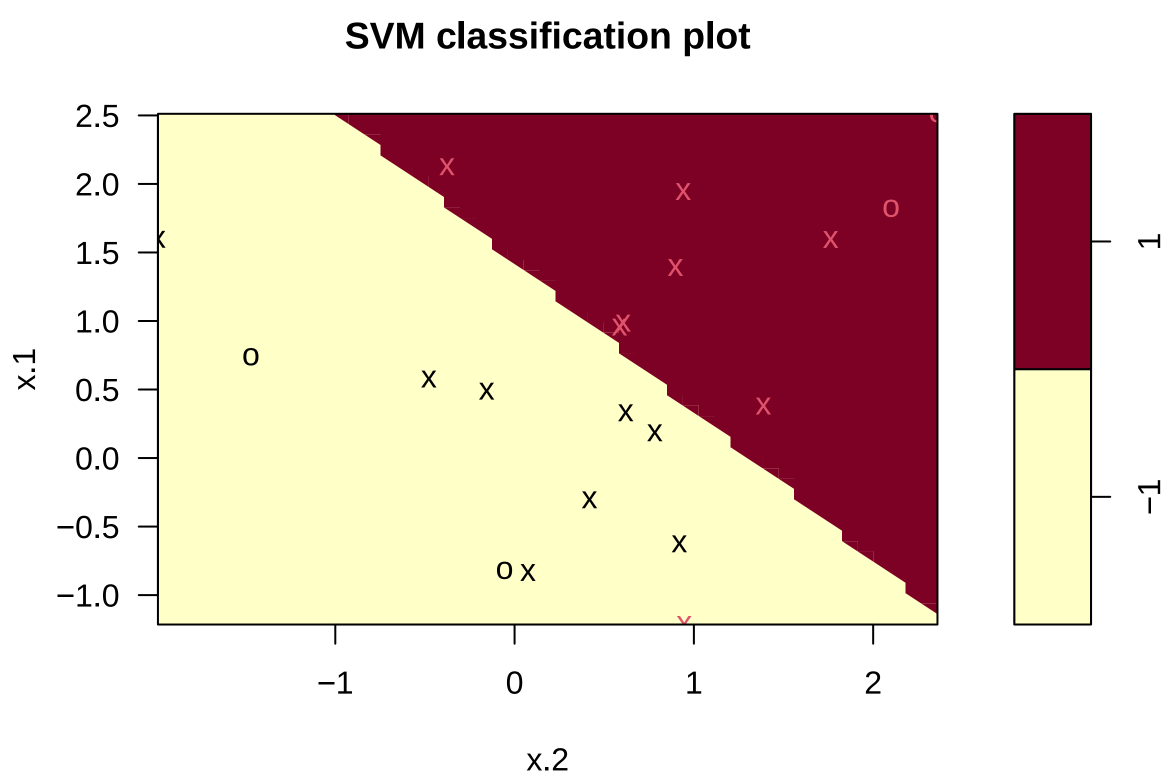
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  -1 1
```

This tells us, for instance, that a linear kernel was used with cost=10, and that there were seven support vectors, four in one class and three in the other.

What if we use a smaller value of cost parameter?

```
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 have a larger number of support vectors, because the margin is now wider

```
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: 0.05
##
## - 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
```

0.05 error is for 0.1 cost, so it is the best one. Tune() stores the best function and we can assess:

```r
best.mod = tune.out$best.model
summary(best.mod)
```

```
##
## 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
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
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))
```

now we made the test set

```r
ypred=predict(best.mod ,testdat)
table(predict = ypred , truth=testdat$y)
```

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

```r
mean(ypred == testdat$y)
```

```
## [1] 0.85
```

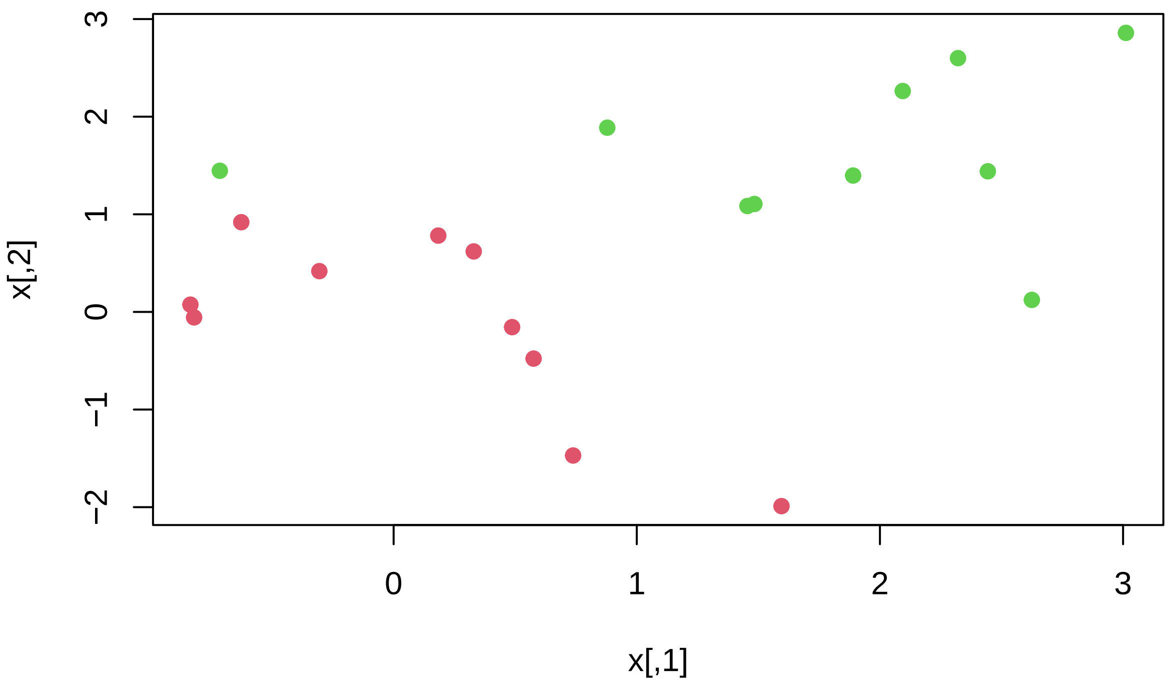
We see that the model predicted with 85% accuracy

Now let's imagine that the observations are barely linearly separable. We fit the support vector classifier and plot the resulting hyperplane, using a very large value of cost so that no observations are misclassified.

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

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



No training errors were made and only three support vectors were used, however we can see that the margin is very narrow which might lead to the errors with test data.

```
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
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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

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

```r
mean(ypred == testdat$y)
```

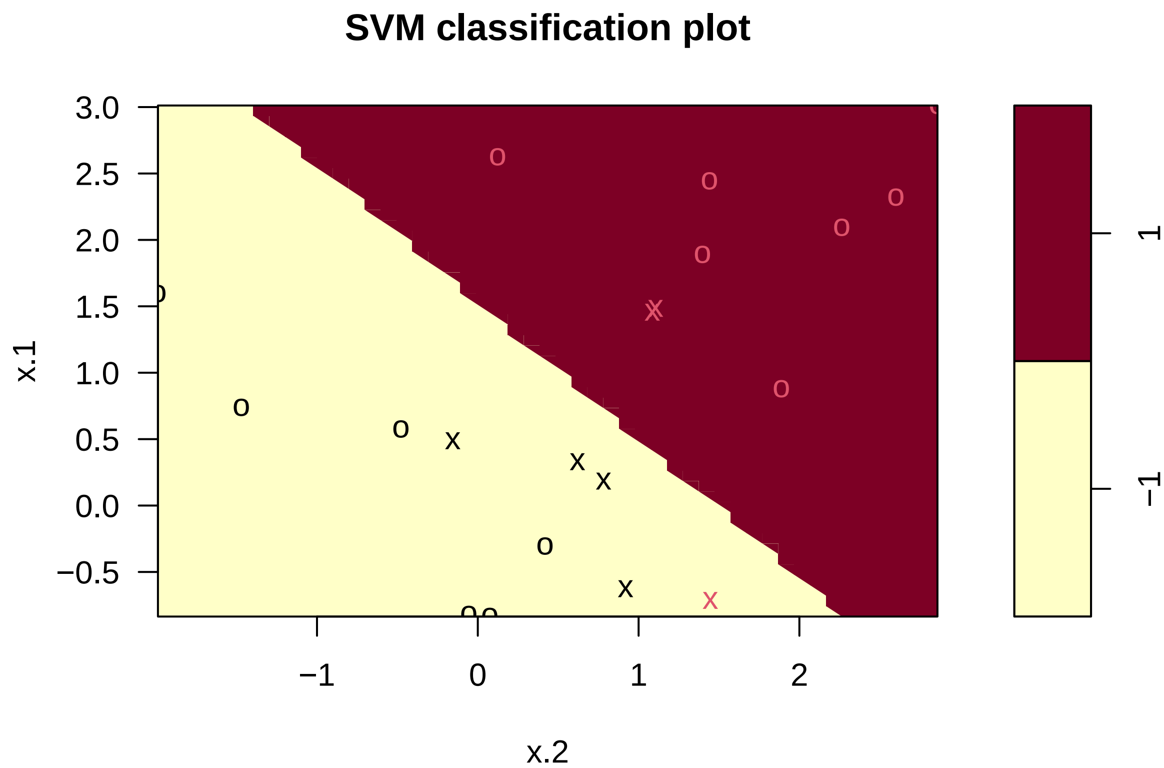
```
## [1] 0.85
```

it is only 65% accuracy

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

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



Using cost=1, we misclassify a training observation, but we also obtain a much wider margin and use seven support vectors. This will eventually perform better on test data.

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

```
##         truth
## predict -1  1
##      -1 13  4
##      1   0  3
```

```
mean(ypred == testdat$y)
```

```
## [1] 0.8
```

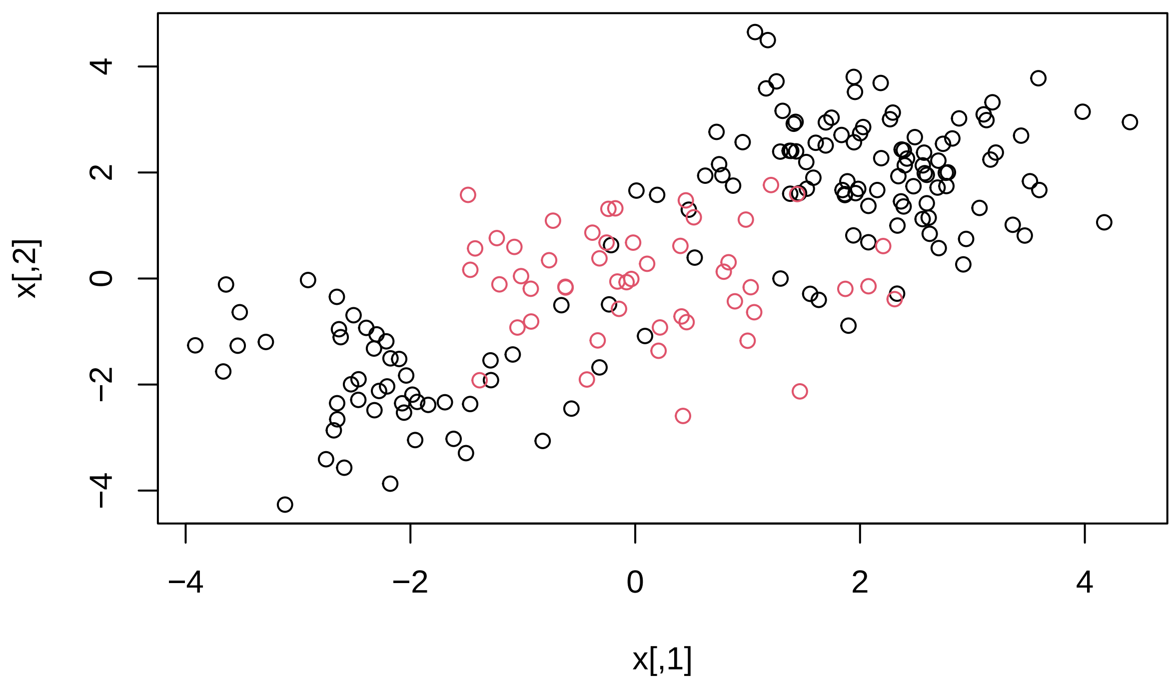
This has a 80% accuracy.

9.6.2 Support Vector Machine

```
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))
```

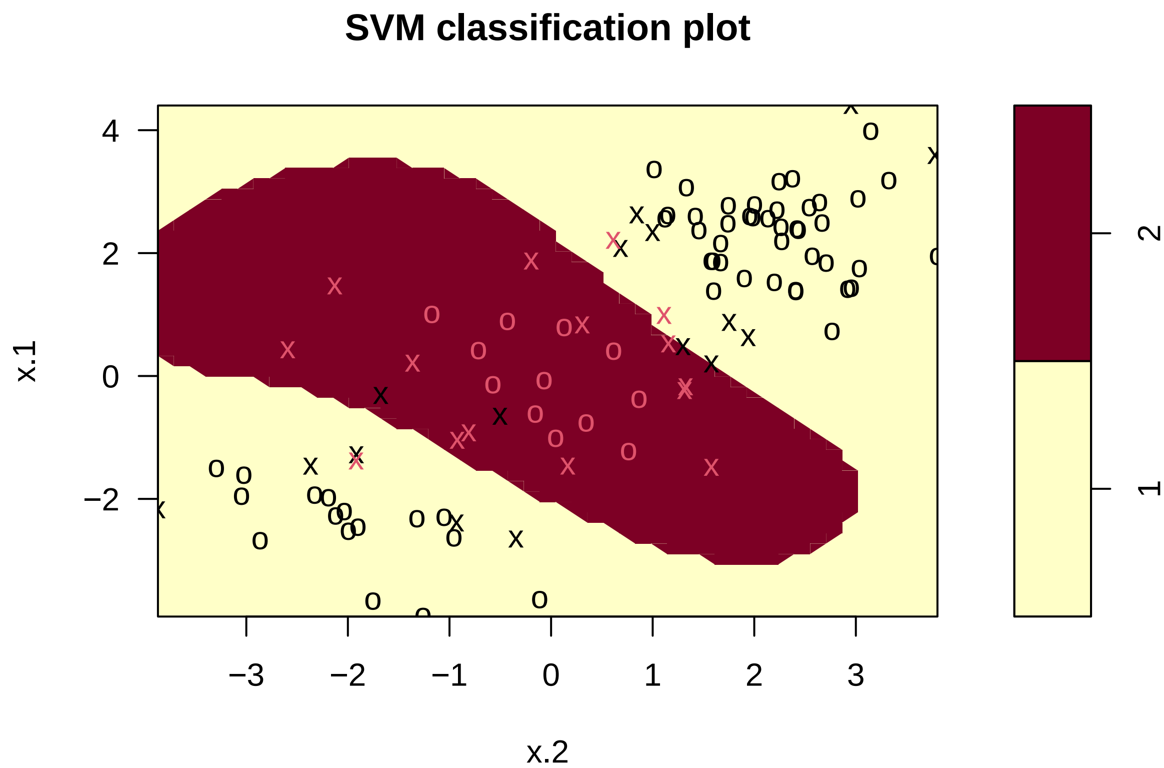
we created the non-linear class boundary

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



This the plot of data

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



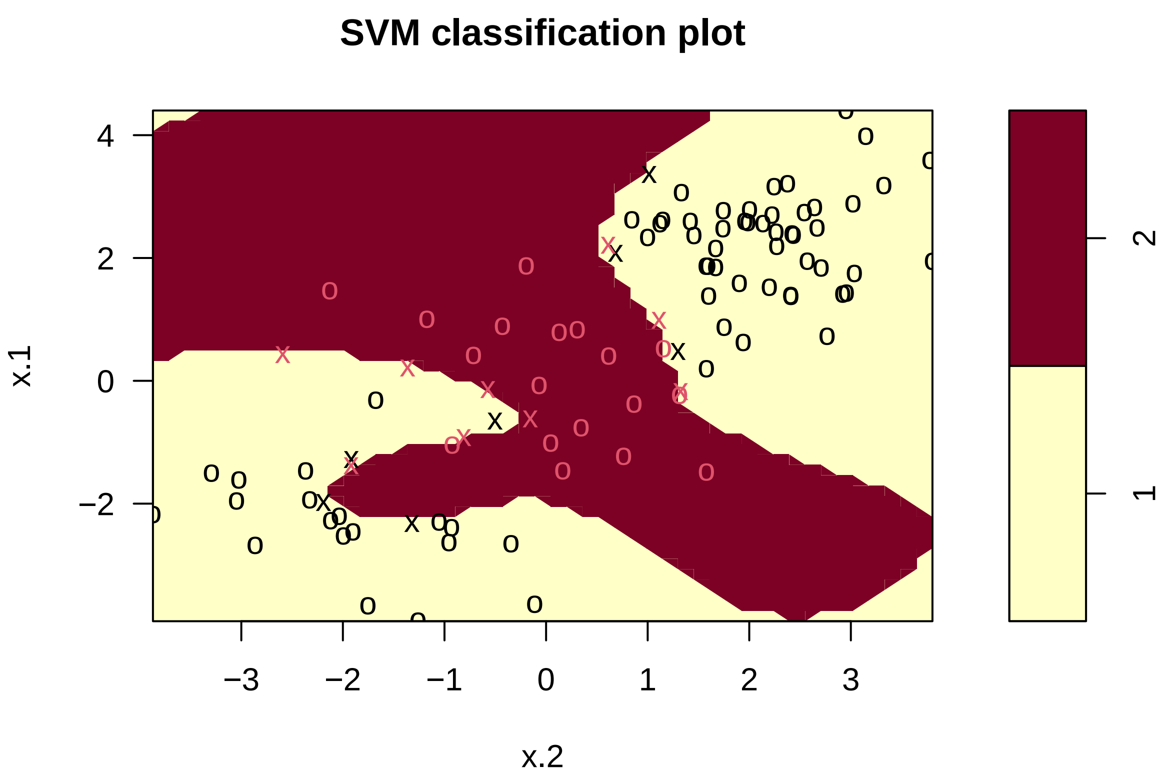
We randomly divided the data into 2 parts and plotted it. There are some mistakes in training set.

```r
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
```

Maybe we can decrease the errors by increasing the cost, however it might be overfitting.

```r
svmfit=svm(y~., data=dat[train ,], kernel ="radial", gamma=1,
  cost=1e5)
plot(svmfit ,dat[train ,])
```



Let's now check the performance

```r
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   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
```

The best is when cost is 1 and gamma is 1

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

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

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

```
## [1] 0.88
```

there is 88% accuracy.

9.6.3 ROC Curves

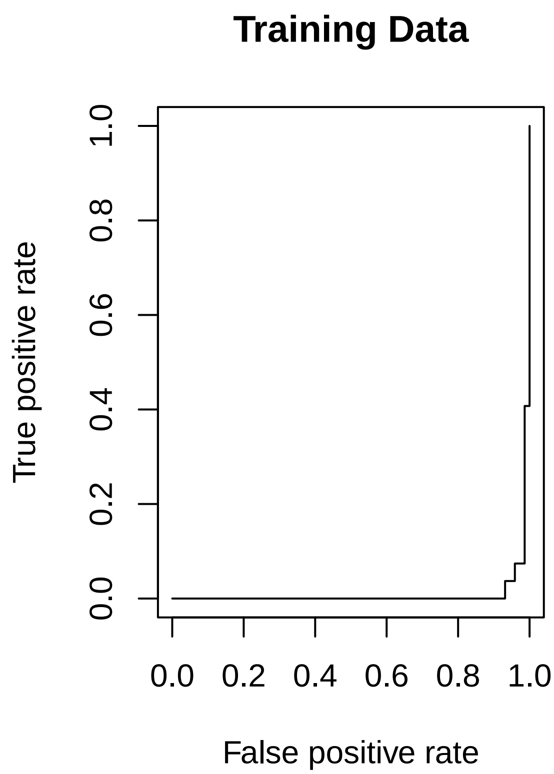
```r
library(ROCR)
rocplot = function(pred , truth , ...){
  predob = prediction(pred, truth)
  perf = performance(predob , "tpr", "fpr")
  plot(perf, ...)}
```

A short function to plot an ROC curve given a vector containing a numerical score for each observation, pred, and a vector containing the class label for each observation, truth.

```
svmfit.opt=svm(y~., data=dat[train ,], kernel ="radial",
  gamma=7, cost=1, decision.values = T)
fitted =attributes(predict(svmfit.opt ,dat[ train ,],
                           decision.values=TRUE))$decision.values
```

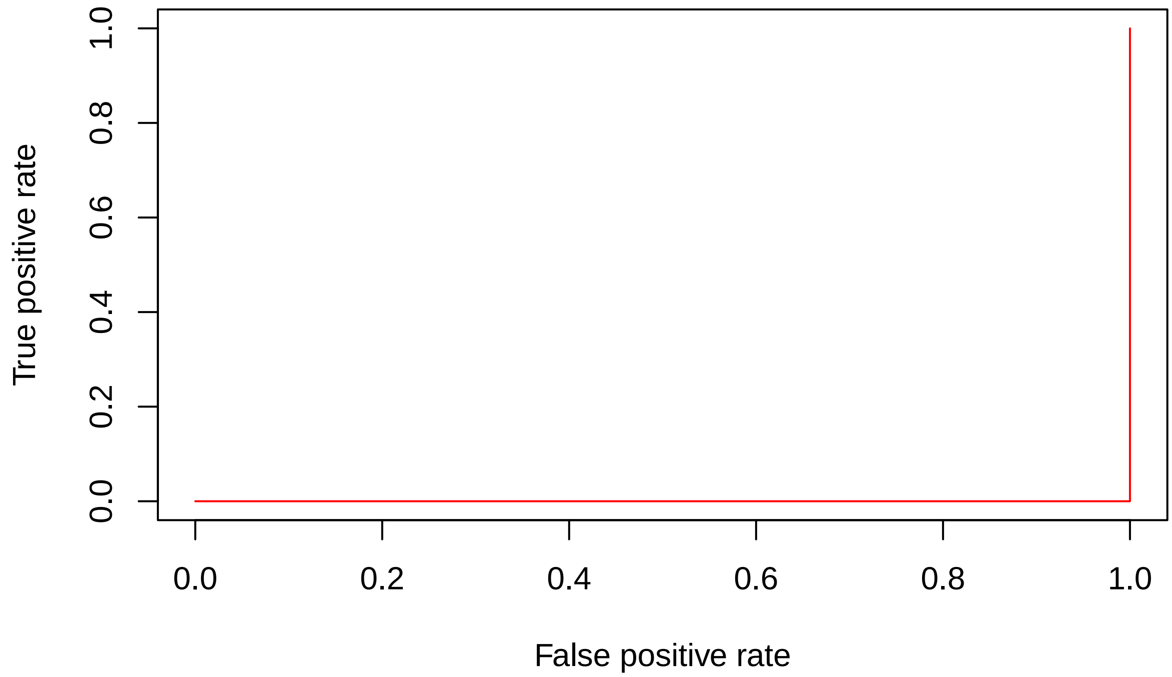
Here we obtained fitted values

```
par(mfrow=c(1,2))
rocplot(fitted, dat[train, "y"], main="Training Data")
```

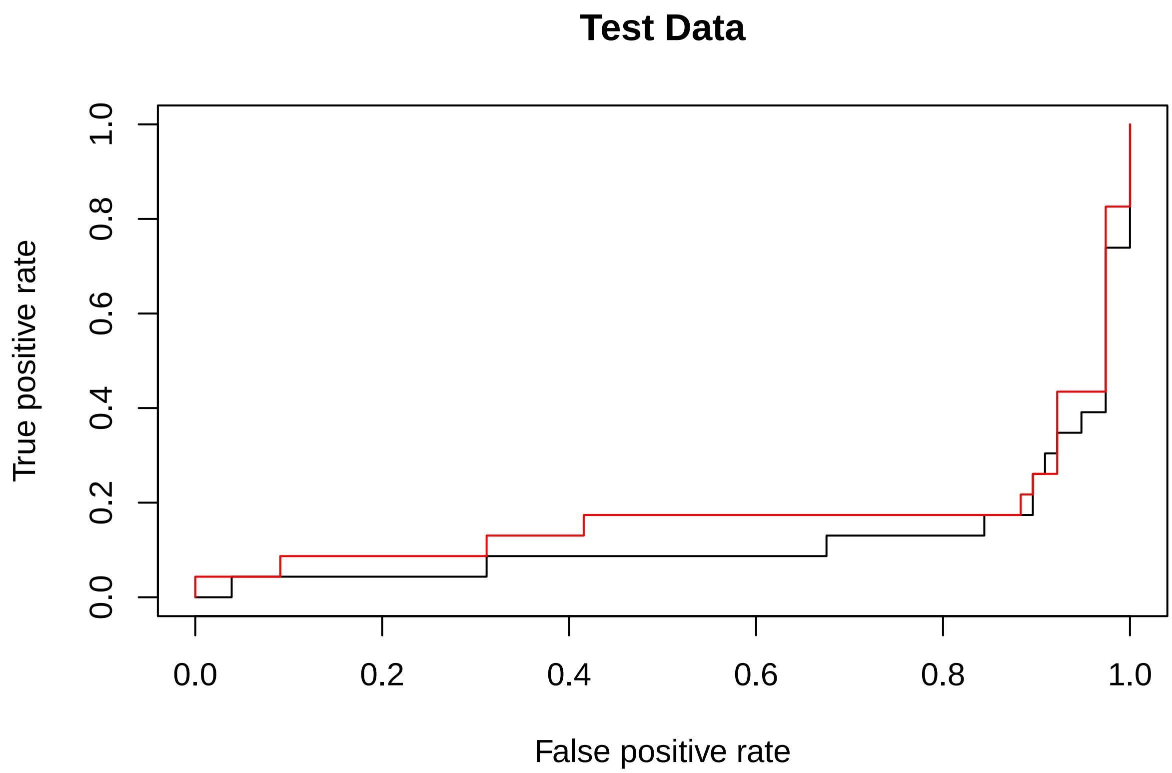


This curve is not as good, but it is for training data.

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




```
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")
```



Although the curve is still pretty bad, it is better on test data set.