Support Vector Machines

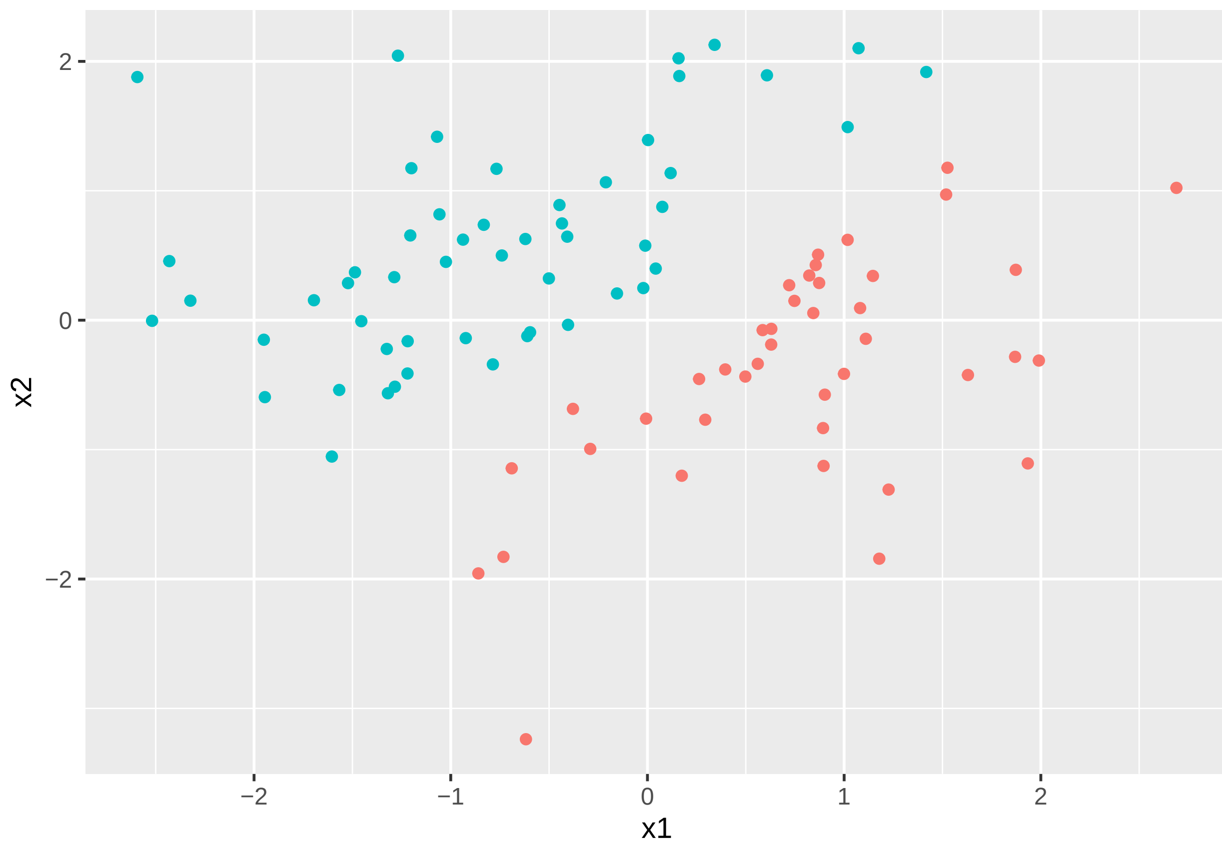
Contents

Linearly Separable.

The following simulated data illustrates a case of linearly separable classes.

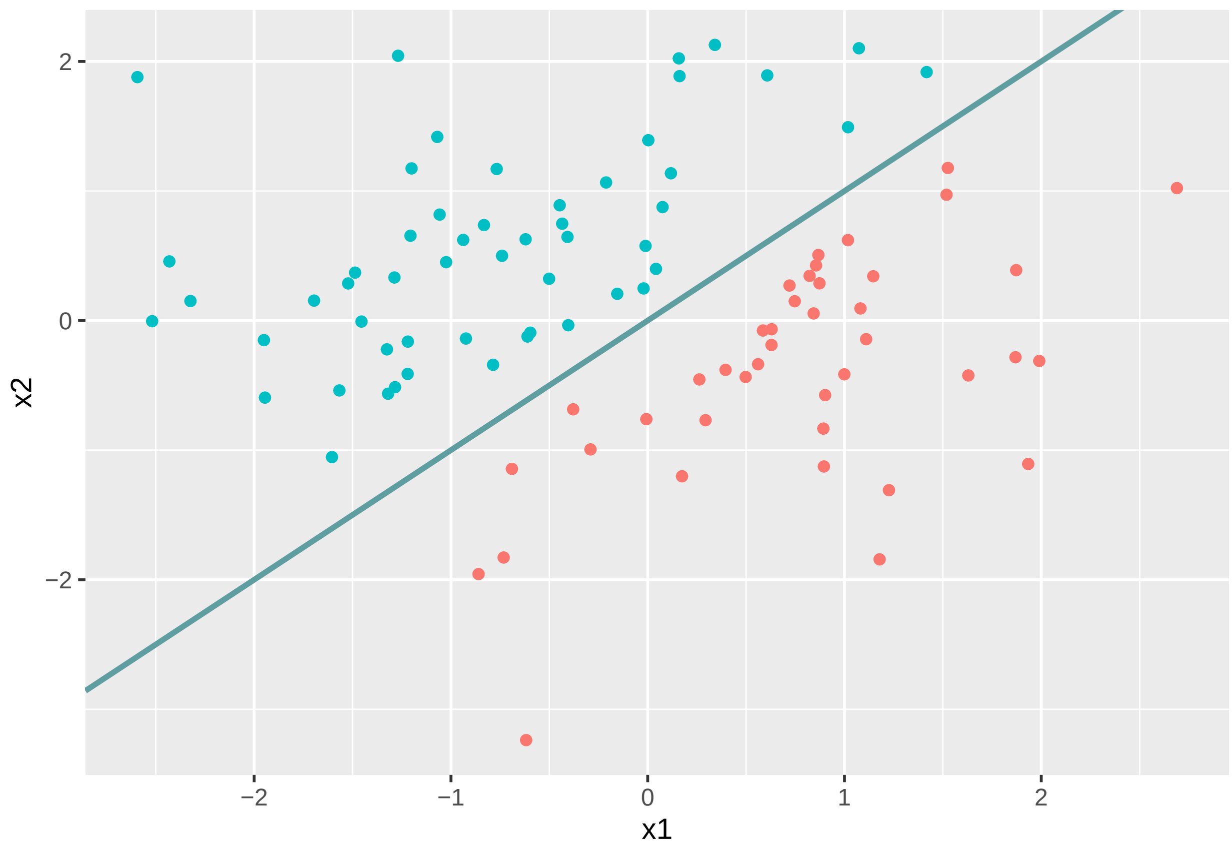
```r
set.seed(0617)
data = data.frame(x1=rnorm(100),x2=rnorm(100))
data$y = factor(ifelse(data$x1>data$x2,0,1))
data = data[abs(data$x1-data$x2)>0.2,]

library(ggplot2)
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)
```



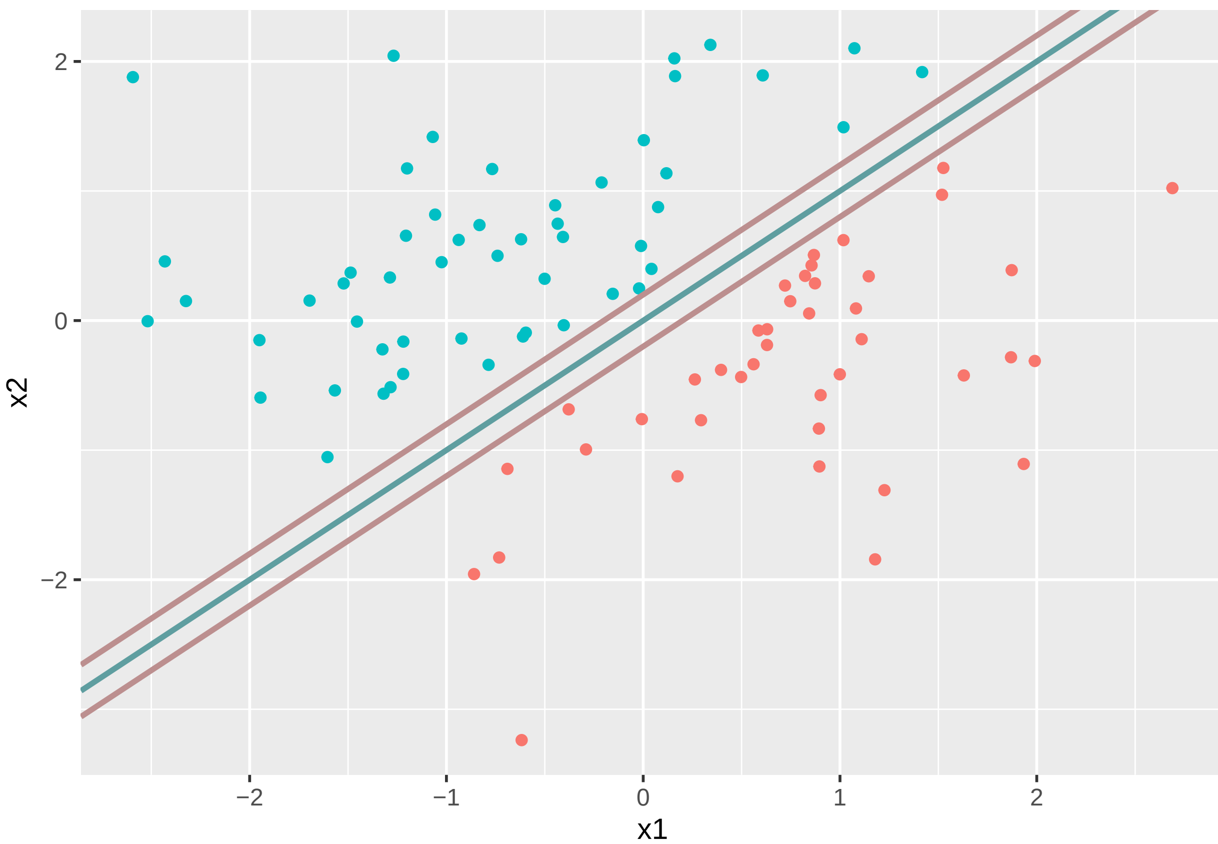
With such data, it is possible to find a linear classifier or a hyperplane.

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)
```



In fact, there are a very large number of hyperplanes that can separate the classes. So, the decision boundary chosen is the one that has the biggest margin and is accordingly called Maximum Margin Classifier.

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)+
  geom_abline(slope = 1,intercept = -0.2,color='rosybrown', size=1)+
  geom_abline(slope = 1,intercept = 0.2,color='rosybrown', size=1)
```



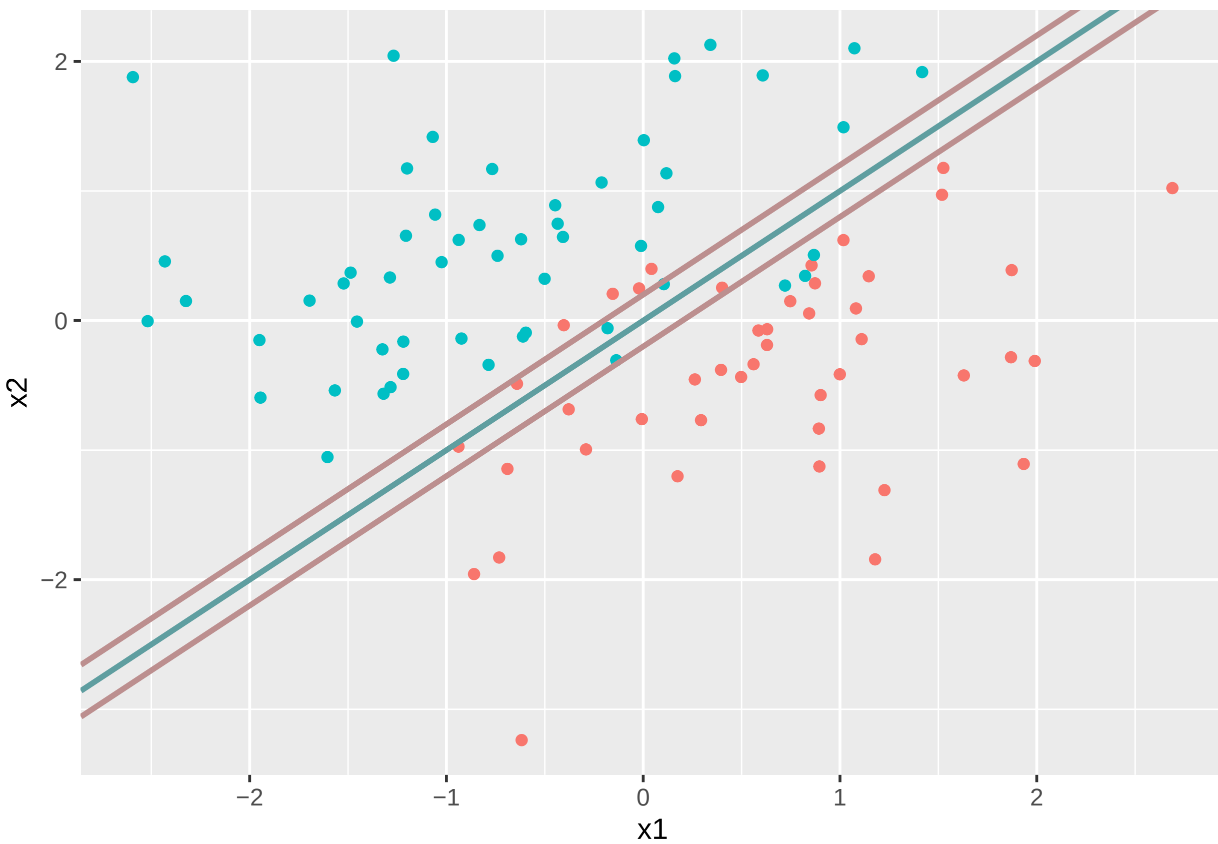
Not Linearly Separable

In practice, classes are seldom linearly separable.

```
set.seed(0617)
data = data.frame(x1=rnorm(100),x2=rnorm(100))
data$y = factor(ifelse(data$x1>data$x2,0,1))
data$y[abs(data$x1-data$x2)<0.5] = factor(sample(c(0,1),size = length(data$y[abs(data$x1-data$x2)<0.5])
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()
```

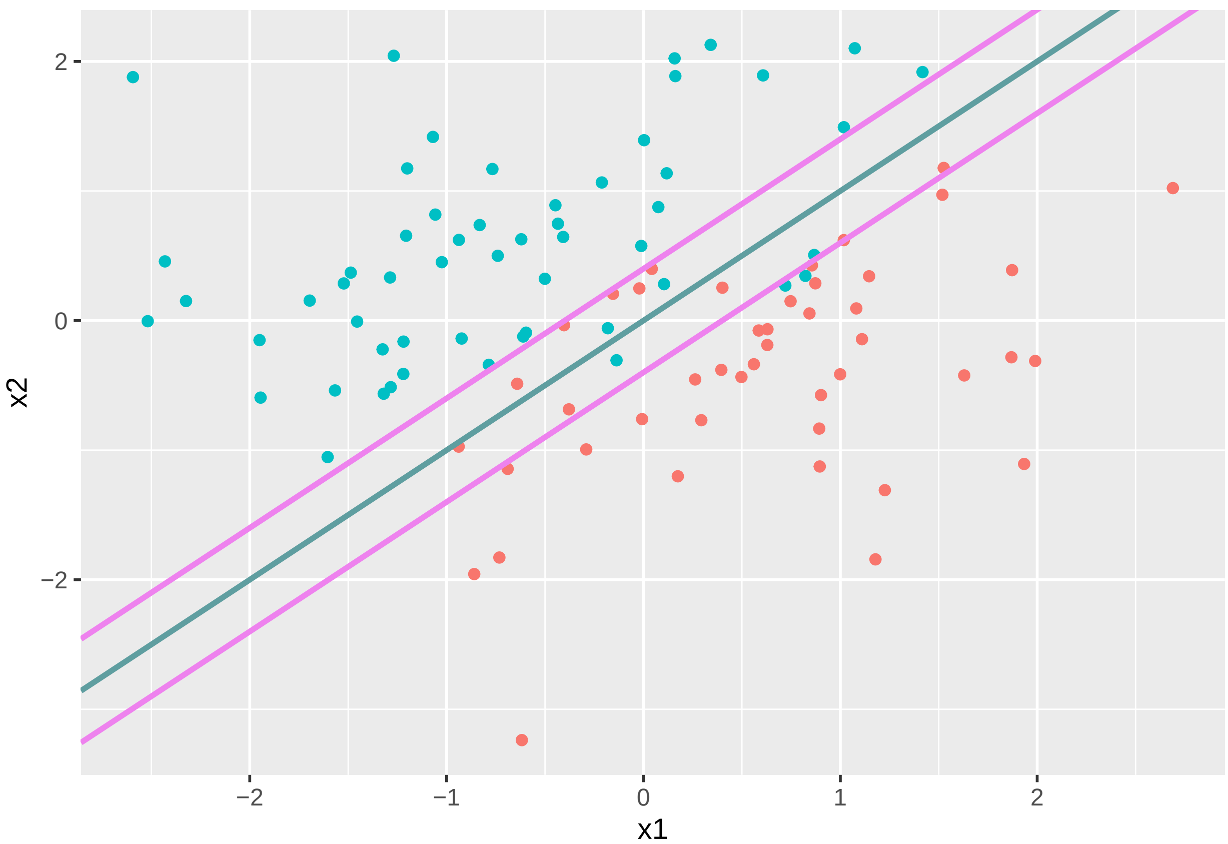
So, the requirement of a hard margin is relaxed. Instead, the support vector classifier maximizes a soft margin.

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)+
  geom_abline(slope = 1,intercept = -0.2,color='rosybrown', size=1)+
  geom_abline(slope = 1,intercept = 0.2,color='rosybrown', size=1)
```



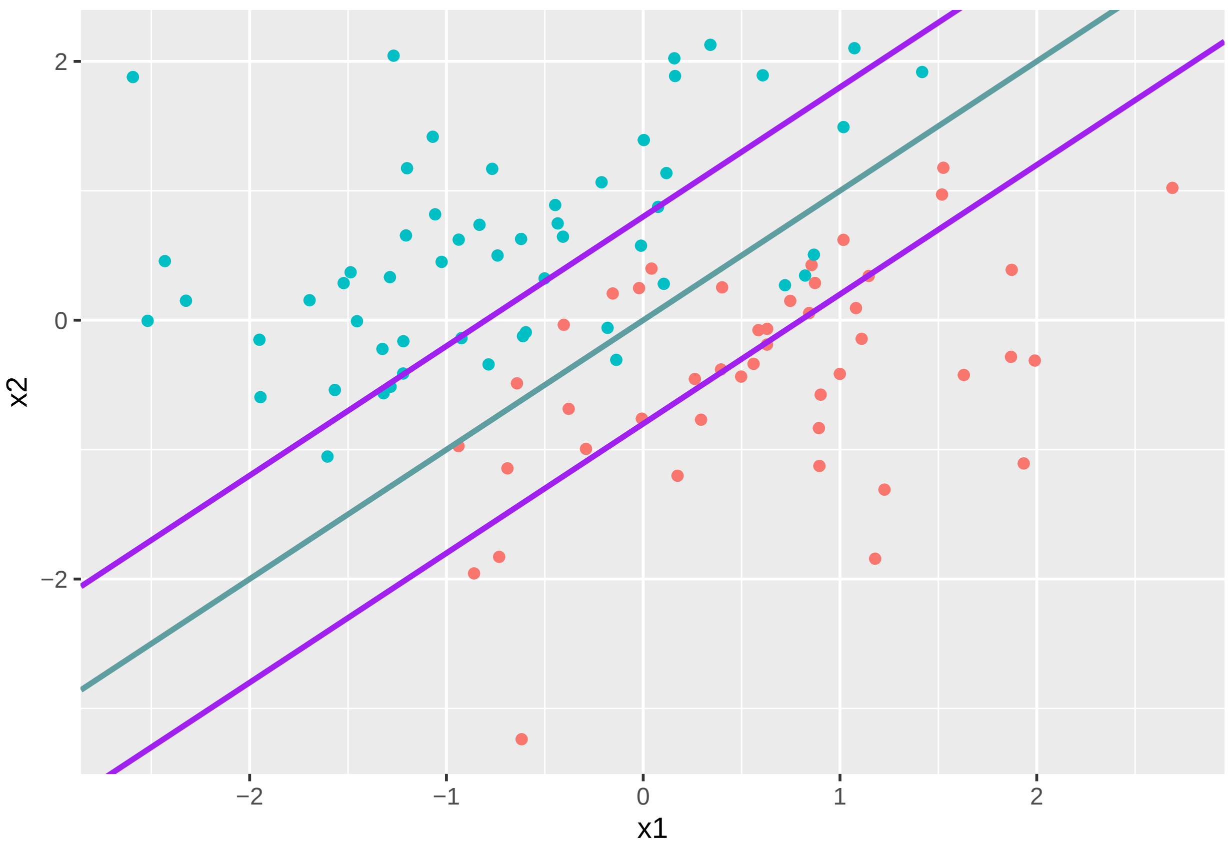
Of course, many soft margins are possible. HEre is another

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)+
  geom_abline(slope = 1,intercept = -0.4,color='violet', size=1)+
  geom_abline(slope = 1,intercept = 0.4,color='violet', size=1)
```



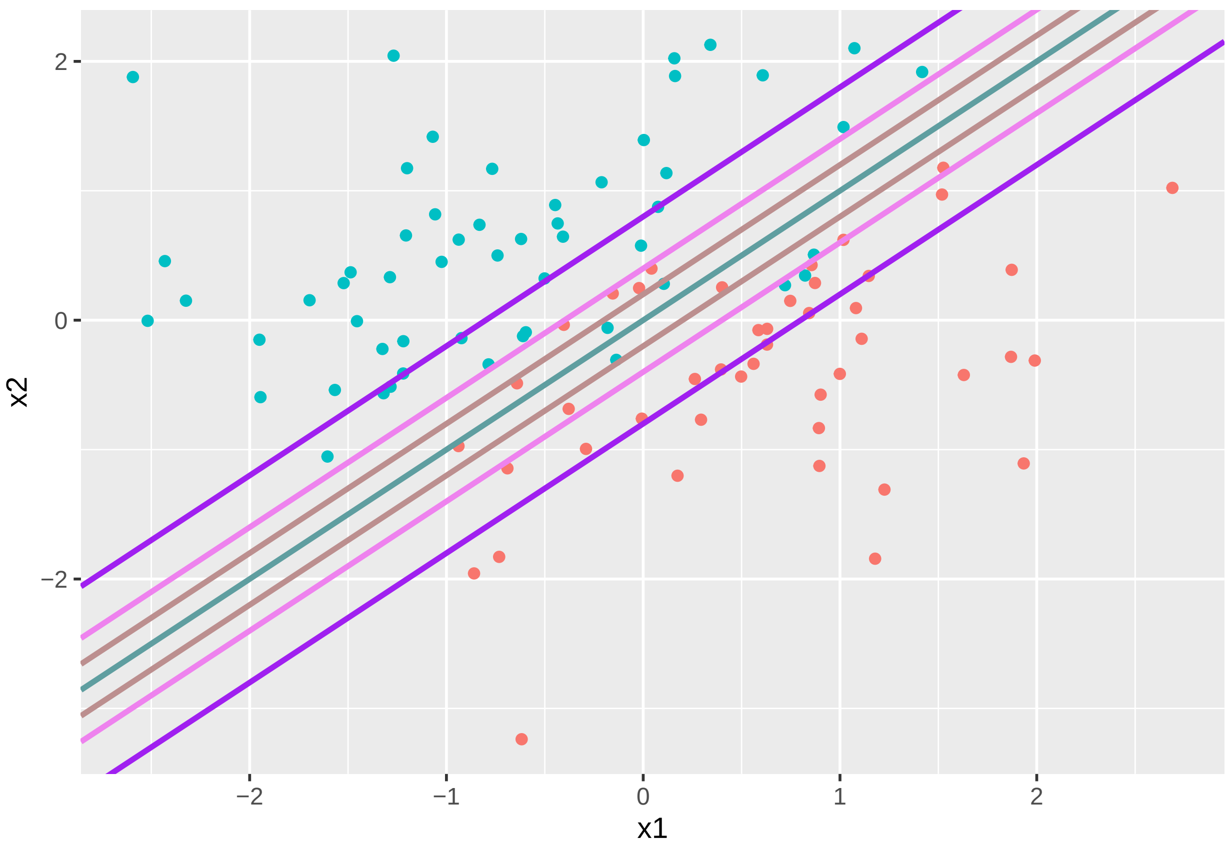
And another.

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)+
  geom_abline(slope = 1,intercept = -0.8,color='purple', size=1)+
  geom_abline(slope = 1,intercept = 0.8,color='purple', size=1)
```



The soft margin used depends on the Cost (C). Higher the cost, narrower the margins. The cost can be determined by the analyst but in practice an SVM model is tuned to determine the optimal cost parameter.

```
ggplot(data,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = 1,intercept = 0,color='cadetblue', size=1)+
  geom_abline(slope = 1,intercept = -0.2,color='rosybrown', size=1)+
  geom_abline(slope = 1,intercept = 0.2,color='rosybrown', size=1)+
  geom_abline(slope = 1,intercept = -0.4,color='violet', size=1)+
  geom_abline(slope = 1,intercept = 0.4,color='violet', size=1)+
  geom_abline(slope = 1,intercept = -0.8,color='purple', size=1)+
  geom_abline(slope = 1,intercept = 0.8,color='purple', size=1)
```



Support Vector Machine Models (Linear)

Let us now examine the type of support vectors fitted by an SVM model to the simulated data we were using above. However, this time we will train the model on a subset of the data.

```
set.seed(0617)
data = data.frame(x1=rnorm(100),x2=rnorm(100))
data$y = factor(ifelse(data$x1>data$x2,0,1))
set.seed(0617)
split = sample(1:nrow(data),0.7*nrow(data))
train = data[split,]
test = data[-split,]
```

SVM: Cost = 1

Fit an SVM model using the default cost of 1. In practice, variables are scaled, but here it has been set to F so that the upcoming graphs are more meaningful.

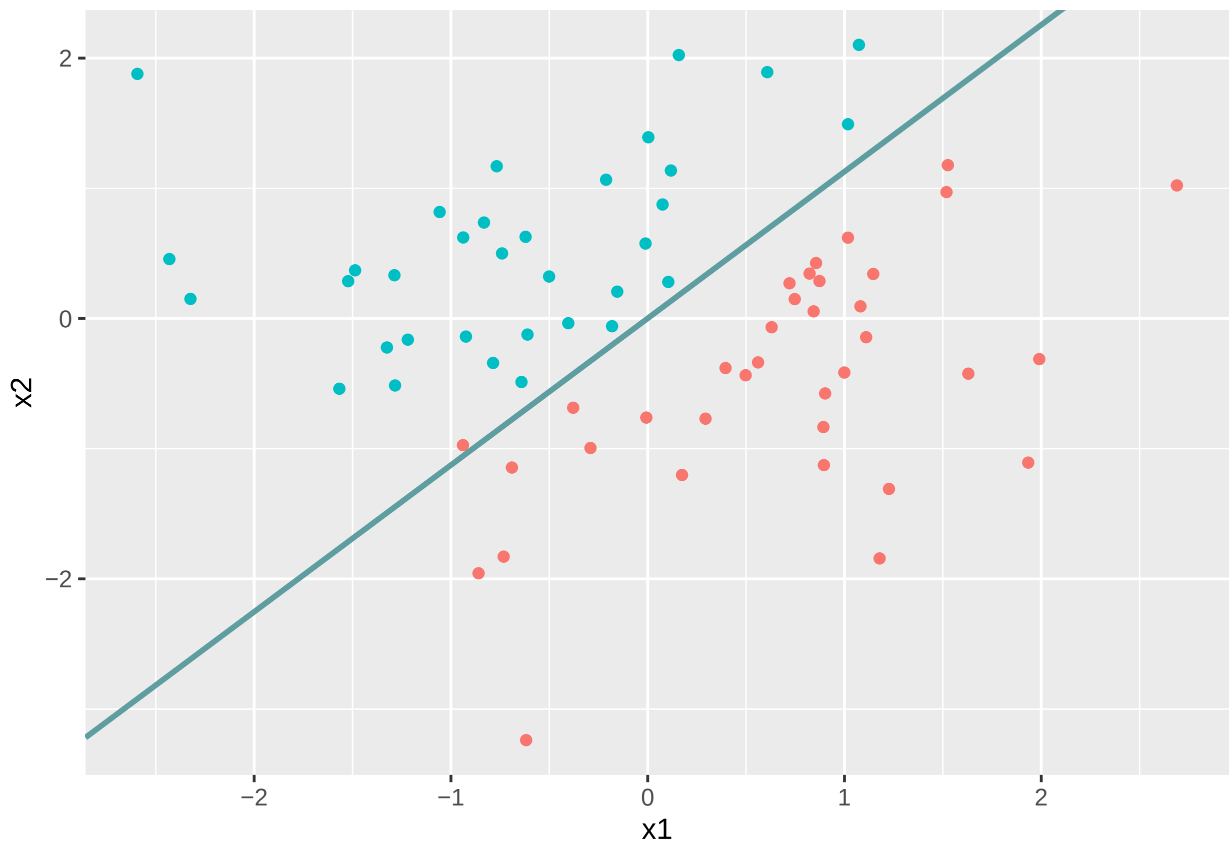
```
library(e1071)
svmLinear = svm(y~.,train,kernel='linear',scale=F,type='C-classification') # if outcome is a factor, de
summary(svmLinear)
```

```
##
```

```
## Call:
## svm(formula = y ~ ., data = train, kernel = "linear", type = "C-classification",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  14
##
##  ( 7 7 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

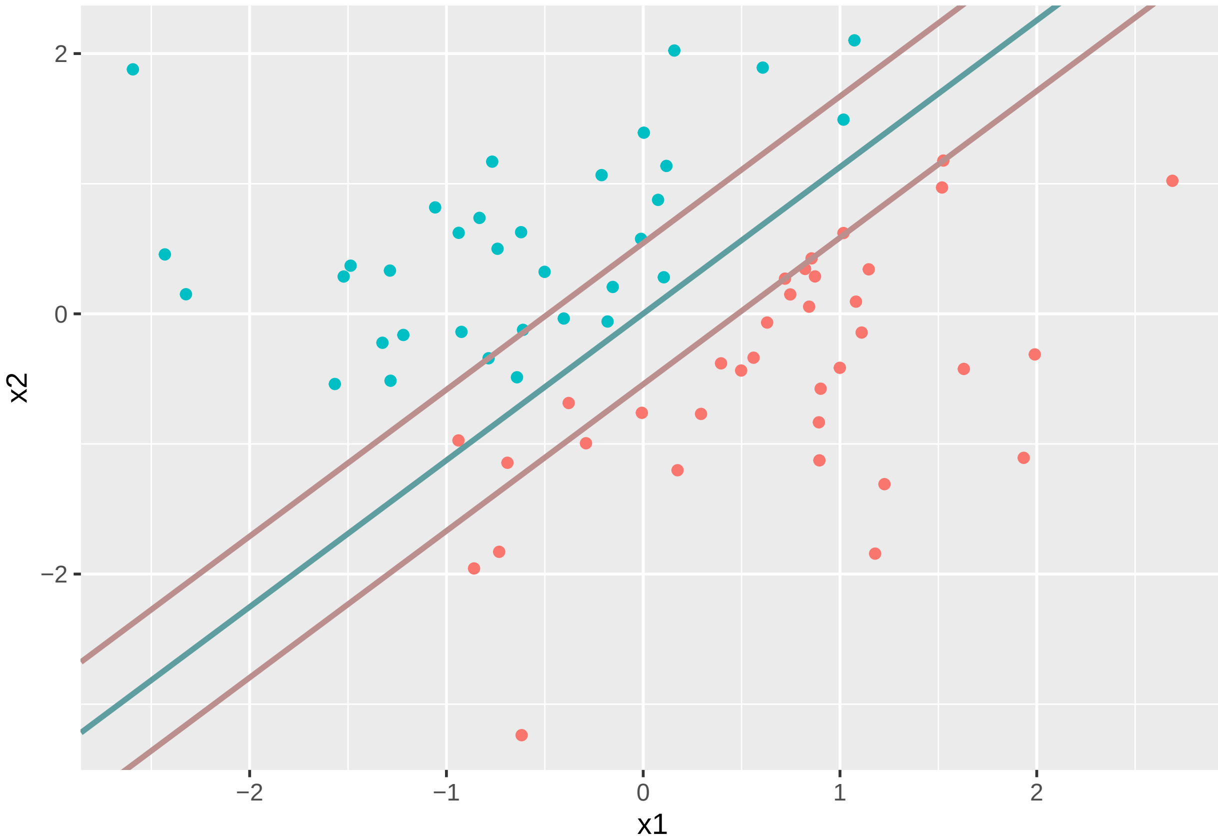
Plot the Decision boundary. Note, svm comes with a plot function that constructs a decision boundary but we are manually constructing it here to draw a comparison with the charts above.

```
beta = t(svmLinear$coefs) %*% svmLinear$SV
slope = -beta[1]/beta[2]
intercept = svmLinear$rho/beta[2]
ggplot(train,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = slope,intercept = intercept,color='cadetblue', size=1)
```



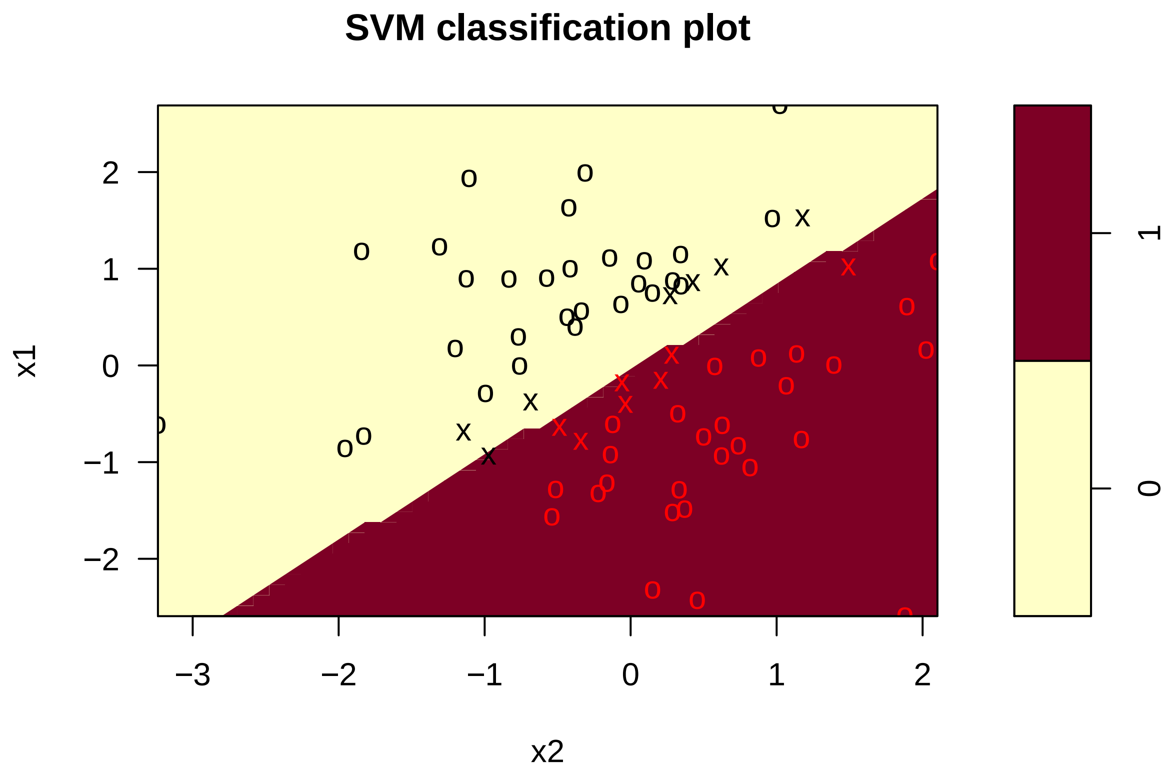
Plot with margins

```
ggplot(train,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = slope,intercept = intercept,color='cadetblue', size=1)+
  geom_abline(slope = slope,intercept = intercept-1/beta[2],color='rosybrown', size=1)+
  geom_abline(slope = slope,intercept = intercept+1/beta[2],color='rosybrown', size=1)
```



Here is the plot that comes with svm()

```
plot(svmLinear, train)
```



Finally, let us examine the performance of the svm on train and test samples

```
pred = predict(svmLinear)
table(pred,train$y)
```

```
##
## pred  0  1
##    0 35  0
##    1  1 34
```

```
pred = predict(svmLinear,newdata=test)
table(pred,test$y)
```

```
##
## pred  0  1
##    0  8  0
##    1  0 22
```

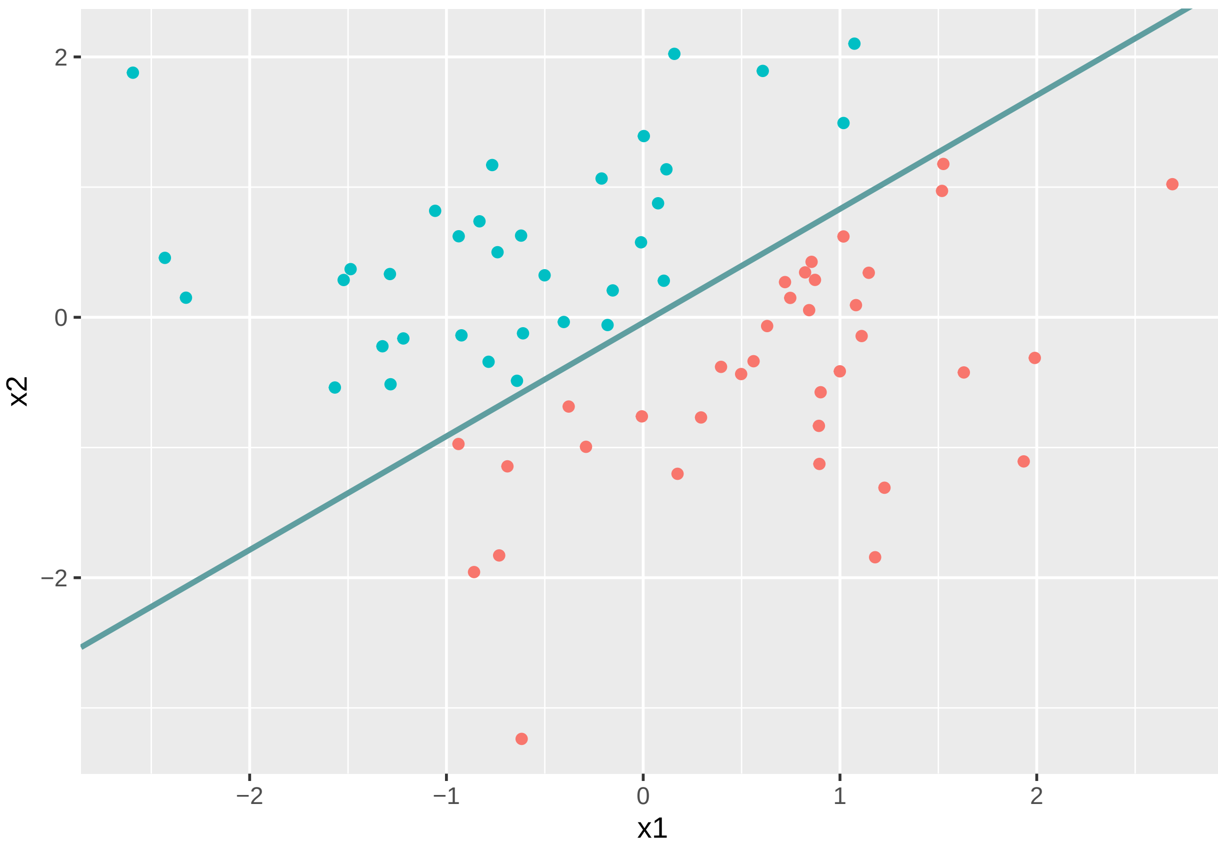
SVM: Cost = 100

Next, we are going to look at an SVM model with a higher cost of 100.

```
library(e1071)
svmLinear = svm(y~.,train,kernel='linear',scale=F,type='C-classification',cost=100) # if outcome is a fa
```

```r
beta = t(svmLinear$coefs) %*% svmLinear$SV
slope = -beta[1]/beta[2]
intercept = svmLinear$rho/beta[2]

ggplot(train,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = slope,intercept = intercept,color='cadetblue', size=1)
```



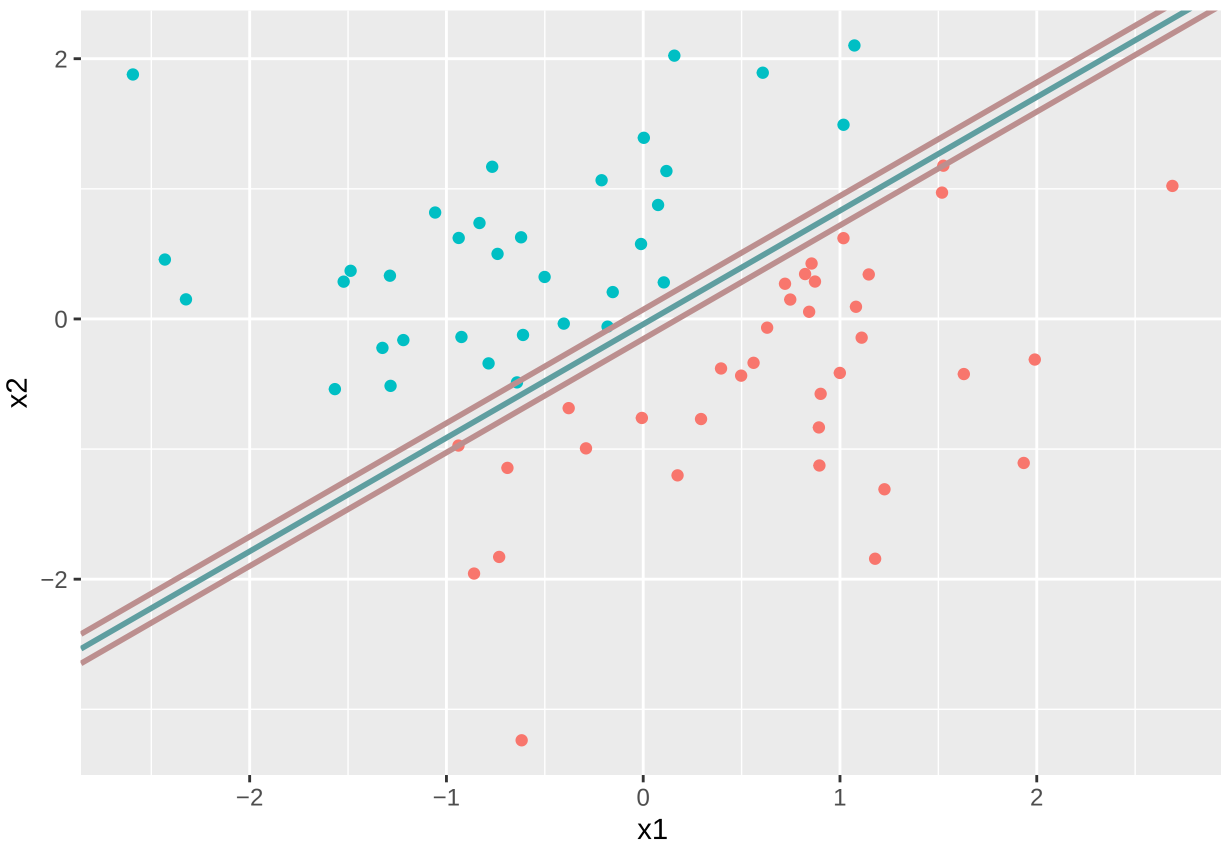

```r
summary(svmLinear)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "linear", type = "C-classification",
##     cost = 100, scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

You may recall, higher the cost, narrower the margins.

```
ggplot(train,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = slope,intercept = intercept,color='cadetblue', size=1)+
  geom_abline(slope = slope,intercept = intercept-1/beta[2],color='rosybrown', size=1)+
  geom_abline(slope = slope,intercept = intercept+1/beta[2],color='rosybrown', size=1)
```



Now, we can compare performance of this SVM with the one with a lower cost

```
pred = predict(svmLinear)
table(pred,train$y)
```

```
##
## pred  0  1
##    0 36  0
##    1  0 34
```

```r
pred = predict(svmLinear,newdata=test)
table(pred,test$y)
```

```
##
## pred  0  1
##    0  8  0
##    1  0 22
```

SVM: Tune for best cost

The best way to figure out the cost parameter is to tune the model.

```r
svmTune = tune(method = svm,y~.,data=train,kernel='linear', type='C-classification', scale=F, ranges = l
svmTune$best.model
```

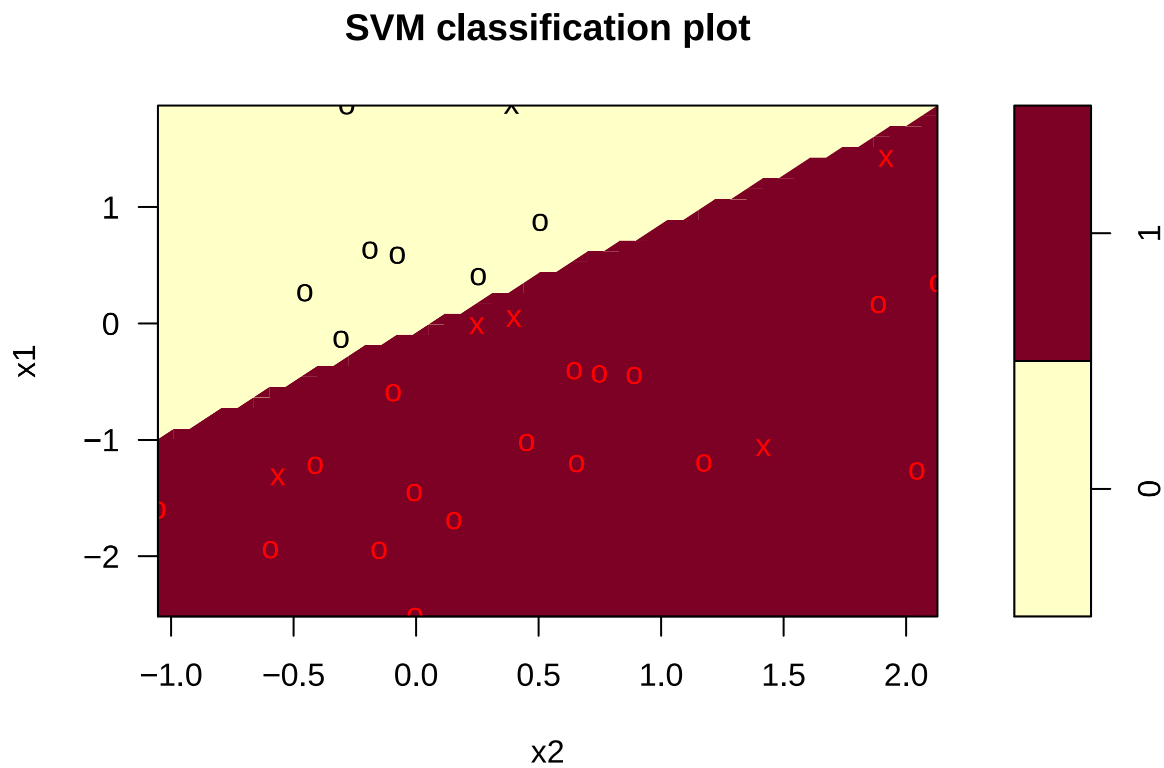
```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 10, 100)), kernel = "linear", type = "C-classification",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  14
```

```r
pred = predict(svmTune$best.model,newdata=test)
table(pred,test$y)
```

```
##
## pred  0  1
##    0  8  0
##    1  0 22
```

Finally, here is a plot of the best model.

```r
plot(svmTune$best.model,test)
```



Note, the examples above looked at binary classification. SVM can easily be extended to more than two categories and can also be used for regression-type problems.

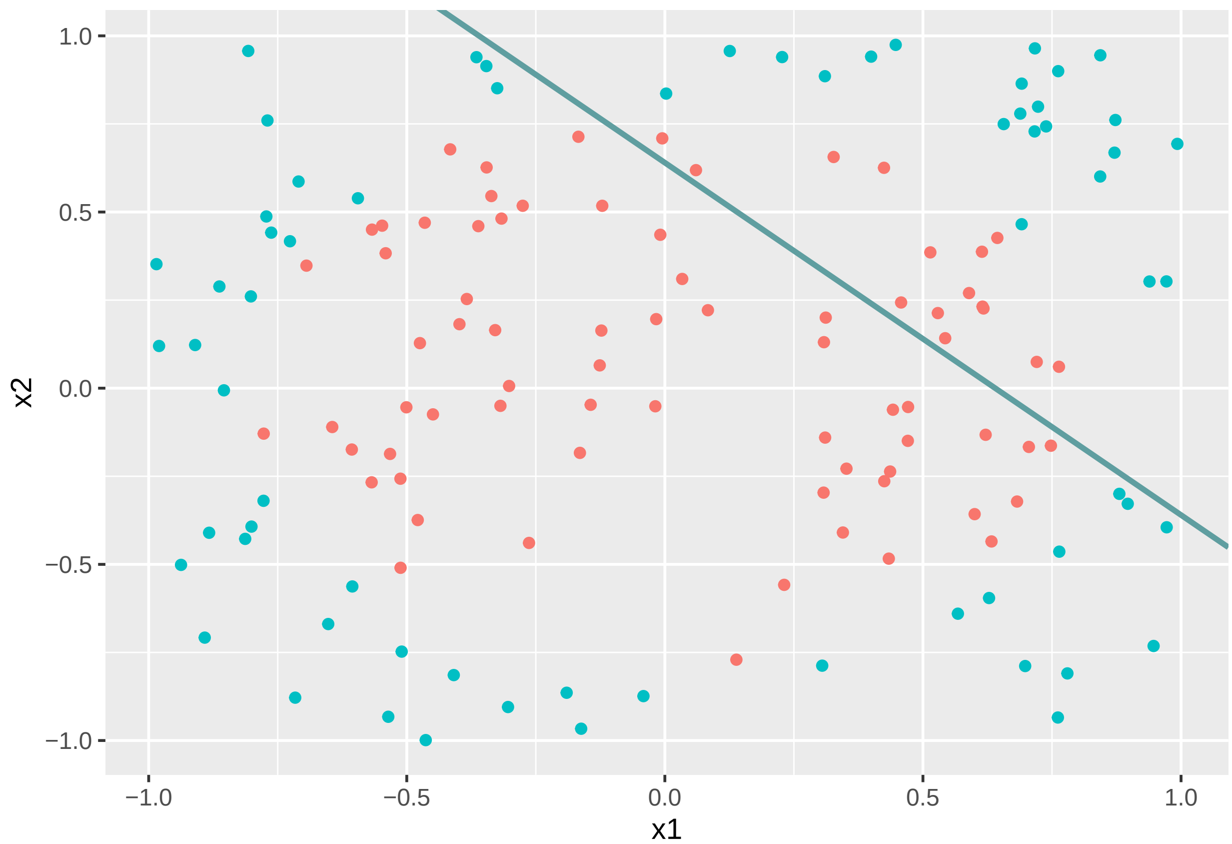
Support Vector Machines - Polynomial

In pracice, a linear boundary may fail, in which case support vector machines are able to function by enriching and enlarging the feature space to make separation possible. This is also known as the "Kernel Trick"

Let us first examine a dataset where a linear classifier is unlikely to succeed.

```
set.seed(0617)
data = data.frame(x1=runif(200,-1,1),x2=runif(200,-1,1))
radius = .8
radius_squared = radius^2
data$y <- factor(ifelse(data$x1^2+data$x2^2<radius_squared, 0, 1))
split = sample(1:nrow(data),0.7*nrow(data))
train = data[split,]
test = data[-split,]

ggplot(train,aes(x=x1,y=x2,color=y))+
  geom_point()+
  guides(color=F)+
  geom_abline(slope = -1,intercept = 0.64,color='cadetblue', size=1)
```



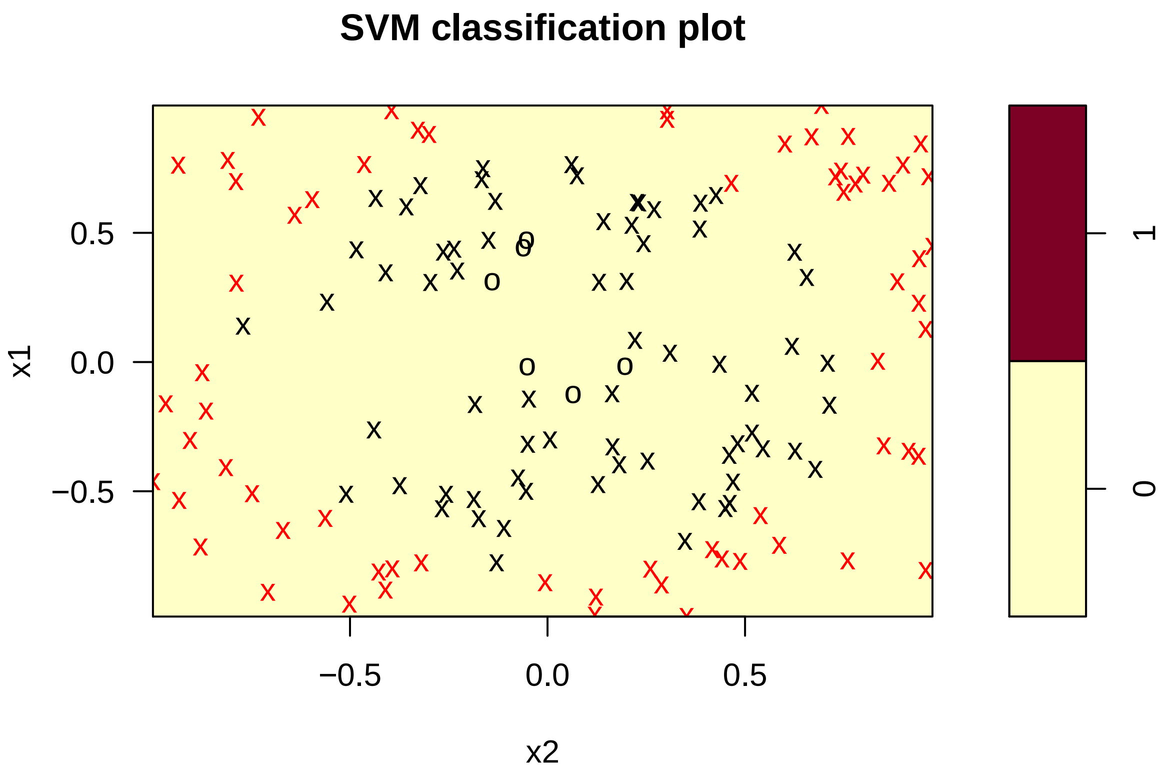

```r
library(e1071)
svmLinear = svm(y~.,data = train, kernel='linear',scale=F,type='C-classification')
pred = predict(svmLinear)
mean(pred==train$y)
```

```
## [1] 0.5285714
```

```r
pred = predict(svmLinear,newdata=test)
mean(pred==test$y)
```

```
## [1] 0.45
```

```r
plot(svmLinear,train)
```



Now, let us try a SVM with a polynomial kernel.

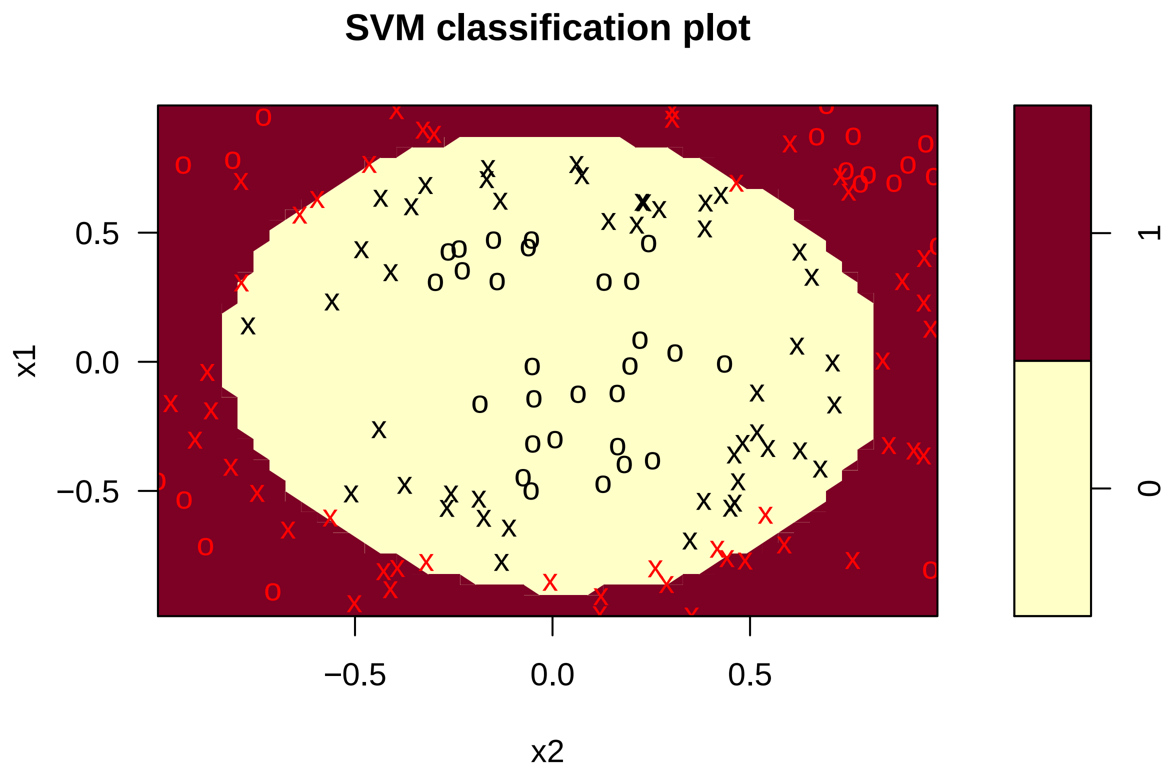
```
svmPoly = svm(y~.,data = train, kernel='polynomial',scale=F,type='C-classification',degree=2)
pred = predict(svmPoly)
mean(pred==train$y)
```

```
## [1] 0.9642857
```

```
pred = predict(svmPoly,newdata=test)
mean(pred==test$y)
```

```
## [1] 0.9166667
```

```
plot(svmPoly,train)
```



But, it is possible that the default parameters chosen aren't optimal, so let us tune this model

```
tune_svmPoly = tune(method = svm,y~.,data = train,kernel='polynomial',
                    ranges= list(degree=c(2,3), cost = c(0.01, 0.1, 1), gamma=c(0,1,10), coef0=c(0,0.1,
summary(tune_svmPoly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree cost gamma coef0
##       2    1    10   0.1
##
## - best performance: 0.01428571
##
## - Detailed performance results:
##    degree cost gamma coef0      error dispersion
## 1       2 0.01     0   0.0 0.47142857 0.12688488
## 2       3 0.01     0   0.0 0.47142857 0.12688488
## 3       2 0.10     0   0.0 0.47142857 0.12688488
## 4       3 0.10     0   0.0 0.47142857 0.12688488
## 5       2 1.00     0   0.0 0.47142857 0.12688488
## 6       3 1.00     0   0.0 0.47142857 0.12688488
## 7       2 0.01     1   0.0 0.16428571 0.11688512
```

```
## 8        3 0.01     1    0.0 0.47857143 0.12621294
## 9        2 0.10     1    0.0 0.05714286 0.05634362
## 10       3 0.10     1    0.0 0.31428571 0.10753895
## 11       2 1.00     1    0.0 0.05000000 0.04821061
## 12       3 1.00     1    0.0 0.30714286 0.09553525
## 13       2 0.01    10    0.0 0.05000000 0.04821061
## 14       3 0.01    10    0.0 0.30714286 0.09553525
## 15       2 0.10    10    0.0 0.05714286 0.05634362
## 16       3 0.10    10    0.0 0.30714286 0.09553525
## 17       2 1.00    10    0.0 0.05714286 0.05634362
## 18       3 1.00    10    0.0 0.29285714 0.09788002
## 19       2 0.01     0    0.1 0.47142857 0.12688488
## 20       3 0.01     0    0.1 0.47142857 0.12688488
## 21       2 0.10     0    0.1 0.47142857 0.12688488
## 22       3 0.10     0    0.1 0.47142857 0.12688488
## 23       2 1.00     0    0.1 0.47142857 0.12688488
## 24       3 1.00     0    0.1 0.47142857 0.12688488
## 25       2 0.01     1    0.1 0.16428571 0.11688512
## 26       3 0.01     1    0.1 0.39285714 0.14384579
## 27       2 0.10     1    0.1 0.03571429 0.03764616
## 28       3 0.10     1    0.1 0.10714286 0.09066397
## 29       2 1.00     1    0.1 0.03571429 0.03764616
## 30       3 1.00     1    0.1 0.03571429 0.03764616
## 31       2 0.01    10    0.1 0.04285714 0.04994328
## 32       3 0.01    10    0.1 0.03571429 0.03764616
## 33       2 0.10    10    0.1 0.04285714 0.03688556
## 34       3 0.10    10    0.1 0.04285714 0.03688556
## 35       2 1.00    10    0.1 0.01428571 0.03011693
## 36       3 1.00    10    0.1 0.04285714 0.03688556
## 37       2 0.01     0    1.0 0.47142857 0.12688488
## 38       3 0.01     0    1.0 0.47142857 0.12688488
## 39       2 0.10     0    1.0 0.47142857 0.12688488
## 40       3 0.10     0    1.0 0.47142857 0.12688488
## 41       2 1.00     0    1.0 0.47142857 0.12688488
## 42       3 1.00     0    1.0 0.47142857 0.12688488
## 43       2 0.01     1    1.0 0.17857143 0.12256703
## 44       3 0.01     1    1.0 0.11428571 0.09035079
## 45       2 0.10     1    1.0 0.04285714 0.03688556
## 46       3 0.10     1    1.0 0.03571429 0.03764616
## 47       2 1.00     1    1.0 0.03571429 0.03764616
## 48       3 1.00     1    1.0 0.04285714 0.03688556
## 49       2 0.01    10    1.0 0.03571429 0.03764616
## 50       3 0.01    10    1.0 0.04285714 0.03688556
## 51       2 0.10    10    1.0 0.02142857 0.03450328
## 52       3 0.10    10    1.0 0.04285714 0.03688556
## 53       2 1.00    10    1.0 0.02857143 0.03688556
## 54       3 1.00    10    1.0 0.05000000 0.04821061
## 55       2 0.01     0   10.0 0.47142857 0.12688488
## 56       3 0.01     0   10.0 0.47142857 0.12688488
## 57       2 0.10     0   10.0 0.47142857 0.12688488
## 58       3 0.10     0   10.0 0.47142857 0.12688488
## 59       2 1.00     0   10.0 0.47142857 0.12688488
## 60       3 1.00     0   10.0 0.47142857 0.12688488
## 61       2 0.01     1   10.0 0.20714286 0.12348860
```

```
## 62      3 0.01     1  10.0 0.04285714 0.03688556
## 63      2 0.10     1  10.0 0.04285714 0.04994328
## 64      3 0.10     1  10.0 0.04285714 0.03688556
## 65      2 1.00     1  10.0 0.03571429 0.03764616
## 66      3 1.00     1  10.0 0.03571429 0.03764616
## 67      2 0.01    10  10.0 0.03571429 0.03764616
## 68      3 0.01    10  10.0 0.03571429 0.03764616
## 69      2 0.10    10  10.0 0.01428571 0.03011693
## 70      3 0.10    10  10.0 0.03571429 0.03764616
## 71      2 1.00    10  10.0 0.03571429 0.03764616
## 72      3 1.00    10  10.0 0.03571429 0.03764616
```

```
tune_svmPoly$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, ranges = list(degree = c(2,
##     3), cost = c(0.01, 0.1, 1), gamma = c(0, 1, 10), coef0 = c(0,
##     0.1, 1, 10)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0.1
##
## Number of Support Vectors:  9
```

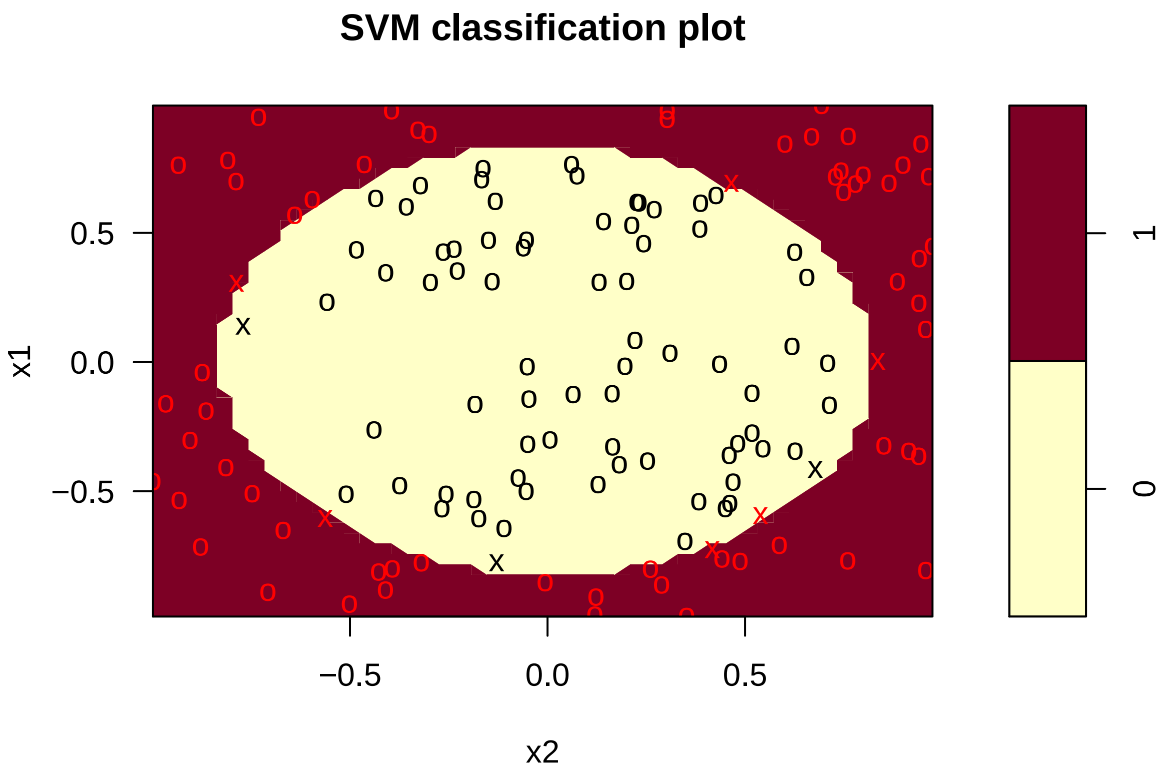
```
pred = predict(tune_svmPoly$best.model)
mean(pred==train$y)
```

```
## [1] 0.9928571
```

```
pred = predict(tune_svmPoly$best.model,newdata=test)
mean(pred==test$y)
```

```
## [1] 0.9666667
```

```
plot(tune_svmPoly$best.model,train)
```



Support Vector Machines - Radial Basis Function

In practice, the Radial Basis function performs better than either Linear or Polynomial kernels as it can fit a variety of decision boundaries.

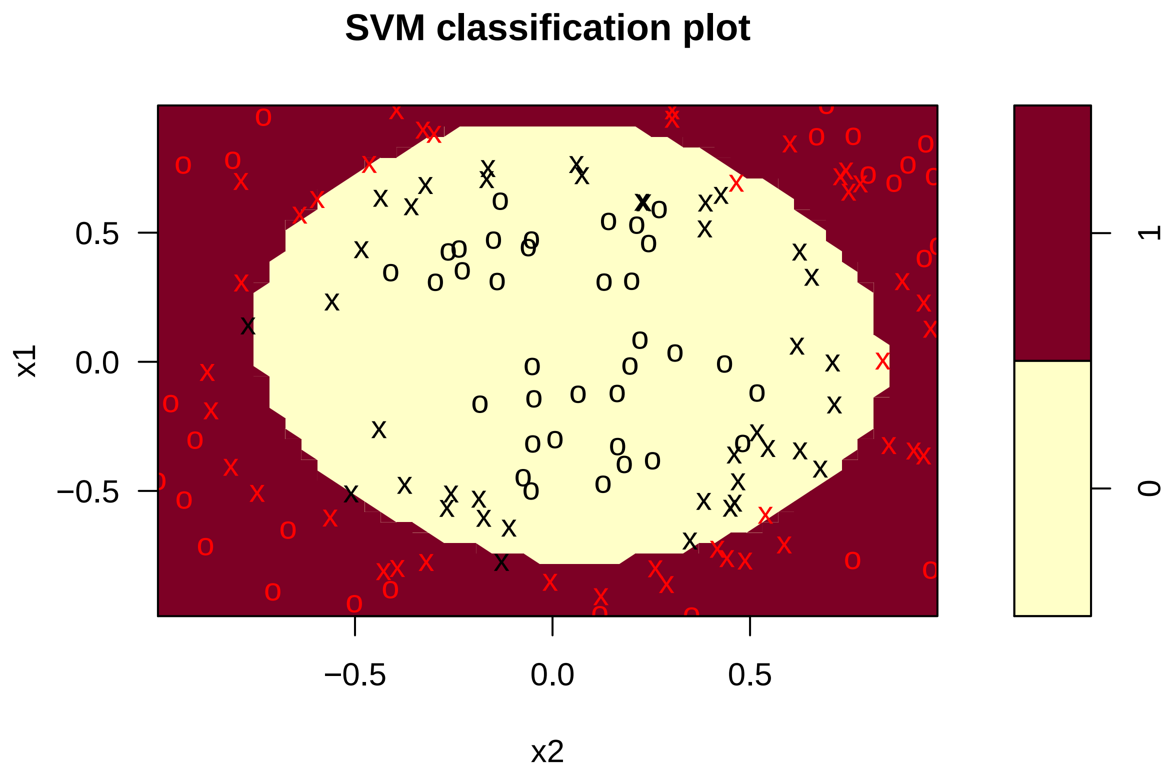
```
svmRadial = svm(y~.,data = train, kernel='radial',scale=F,type='C-classification')
pred = predict(svmRadial)
mean(pred==train$y)
```

```
## [1] 0.9642857
```

```
pred = predict(svmRadial,newdata=test)
mean(pred==test$y)
```

```
## [1] 0.9166667
```

```
plot(svmRadial,train)
```



As before, let's tune the model

```
tune_svmRadial = tune(method='svm',y~.,data=train,kernel='radial', type='C-classification',
                      ranges=list(cost=c(0.1,10,100), gamma=c(1,10), coef0 = c(0.1,1,10)))
summary(tune_svmRadial$best.model)
```

```
##
## Call:
## best.tune(method = "svm", train.x = y ~ ., data = train, ranges = list(cost = c(0.1,
##     10, 100), gamma = c(1, 10), coef0 = c(0.1, 1, 10)), kernel = "radial",
##     type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  27
##
##  ( 12 15 )
##
##
## Number of Classes:  2
##
## Levels:
```

```
##  0 1
```

```r
pred = predict(tune_svmRadial$best.model)
mean(pred==train$y)
```

```
## [1] 0.9642857
```

```r
pred = predict(tune_svmRadial$best.model,newdata=test)
mean(pred==test$y)
```

```
## [1] 0.9666667
```

```r
plot(tune_svmRadial$best.model,test)
```

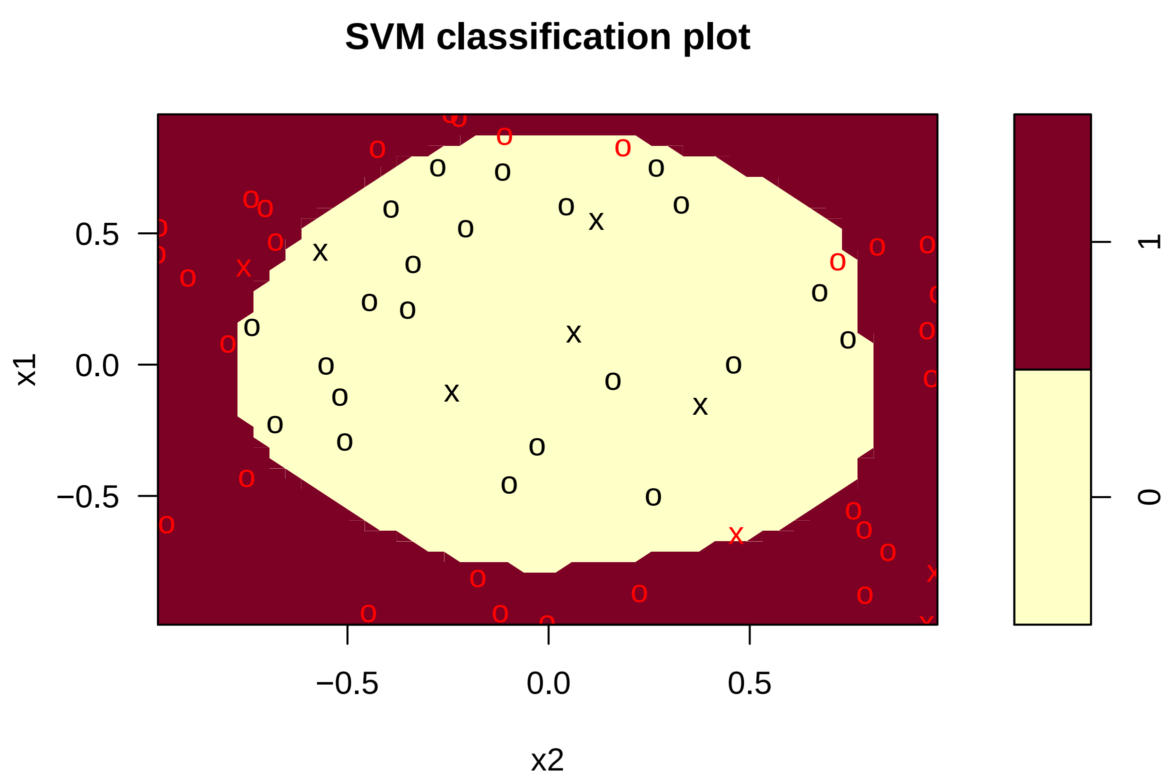


Illustration with wine dataset

Now, let's use SVM to predict wine quality. First, we are going to convert the quality variable from the Wine data into a binary variable.

Read in the wine quality data using the following code: data = read.csv('winequality-white.csv,sep=';')

```r
data$quality = factor(ifelse(data$quality>mean(data$quality), 1, 0),labels = c('high','low'))
```

```r
library(caTools)
set.seed(1706)
split = sample.split(data$quality,SplitRatio = 0.7)
train = data[split,]
test = data[!split,]
```

We are now going to compare performance of Tree and a set of SVMs to predict wine quality using only alcohol and volatile.acidity

Tree Model

```r
library(rpart)
tree = rpart(quality~alcohol+volatile.acidity,train,method='class')
pred = predict(tree,newdata=test,type = 'class')
table(pred,test$quality)
```

```
##
## pred   high low
##   high  263 156
##   low   229 821
```

```r
mean(pred==test$quality)
```

```
## [1] 0.737917
```

Linear SVM

```r
svmLinear = svm(quality~alcohol+volatile.acidity,data = train,kernel='linear',type='C-classification')
summary(svmLinear)
```

```
##
## Call:
## svm(formula = quality ~ alcohol + volatile.acidity, data = train,
##     kernel = "linear", type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  2072
##
##  ( 1037 1035 )
##
##
## Number of Classes:  2
##
## Levels:
##  high low
```

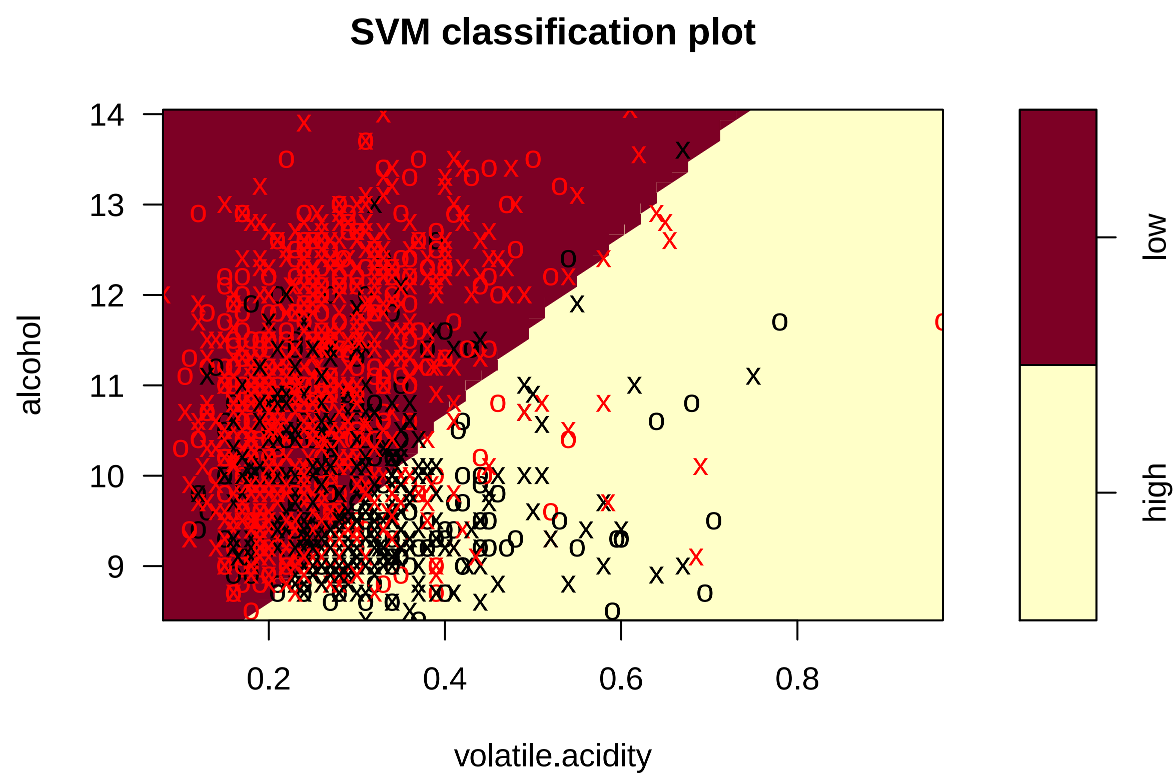
```r
pred = predict(svmLinear,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high  207  98
##   low   285 879
```

```r
mean(pred==test$quality)
```

```
## [1] 0.7392784
```

```r
plot(svmLinear,test[,c('quality','alcohol','volatile.acidity')]) # decision boundary looks non-linear b
```



Linear SVM - Tuned

```r
svmLinearTune = tune(method = svm,quality~alcohol+volatile.acidity,data=train,kernel='linear',type='C-c
summary(svmLinearTune$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = quality ~ alcohol + volatile.acidity,
##     data = train, ranges = list(cost = c(0.01, 0.1, 1, 10, 100)),
```

```
##     kernel = "linear", type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  2079
##
##  ( 1040 1039 )
##
##
## Number of Classes:  2
##
## Levels:
##  high low
```

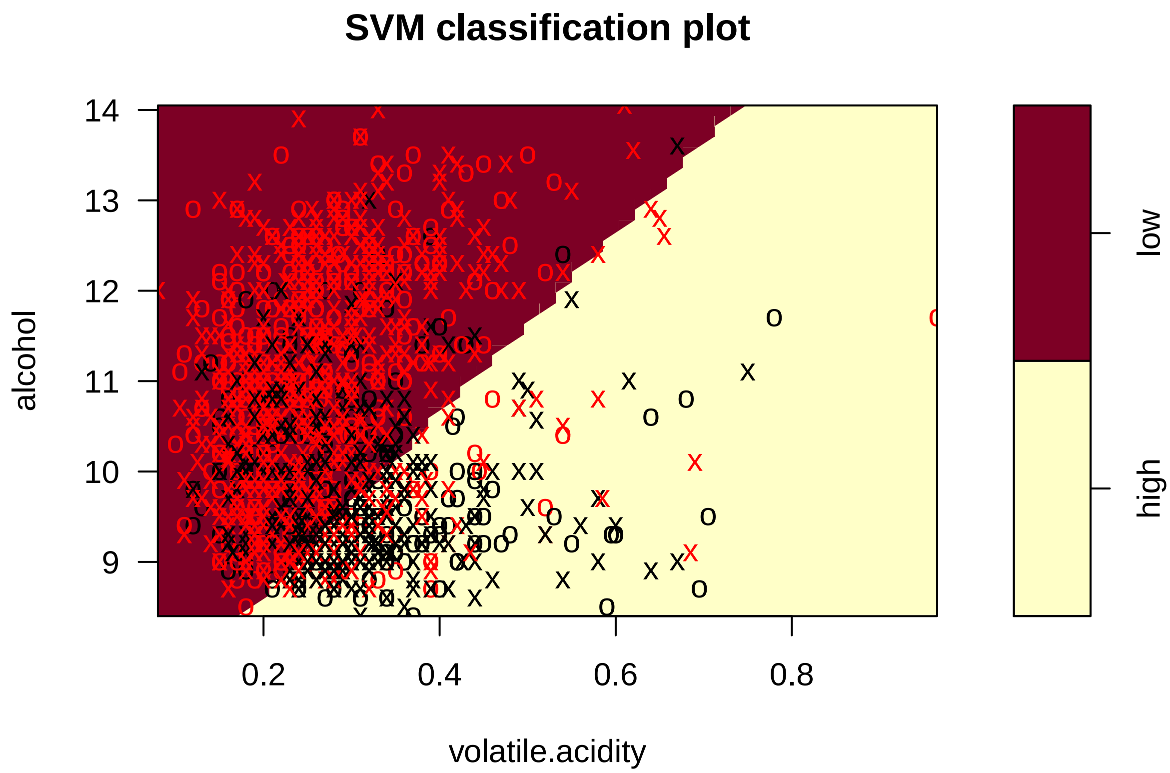
```r
pred = predict(svmLinearTune$best.model,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high  207  98
##   low   285 879
```

```r
mean(pred==test$quality)
```

```
## [1] 0.7392784
```

```r
plot(svmLinearTune$best.model,test[,c('quality','alcohol','volatile.acidity')])
```



Polynomial SVM

```
svmPolynomial = svm(quality~alcohol+volatile.acidity,data = train,kernel='polynomial',degree=2,type='C-c
summary(svmPolynomial)
```

```
##
## Call:
## svm(formula = quality ~ alcohol + volatile.acidity, data = train,
##     kernel = "polynomial", degree = 2, type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  2289
##
##  ( 1146 1143 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  high low
```

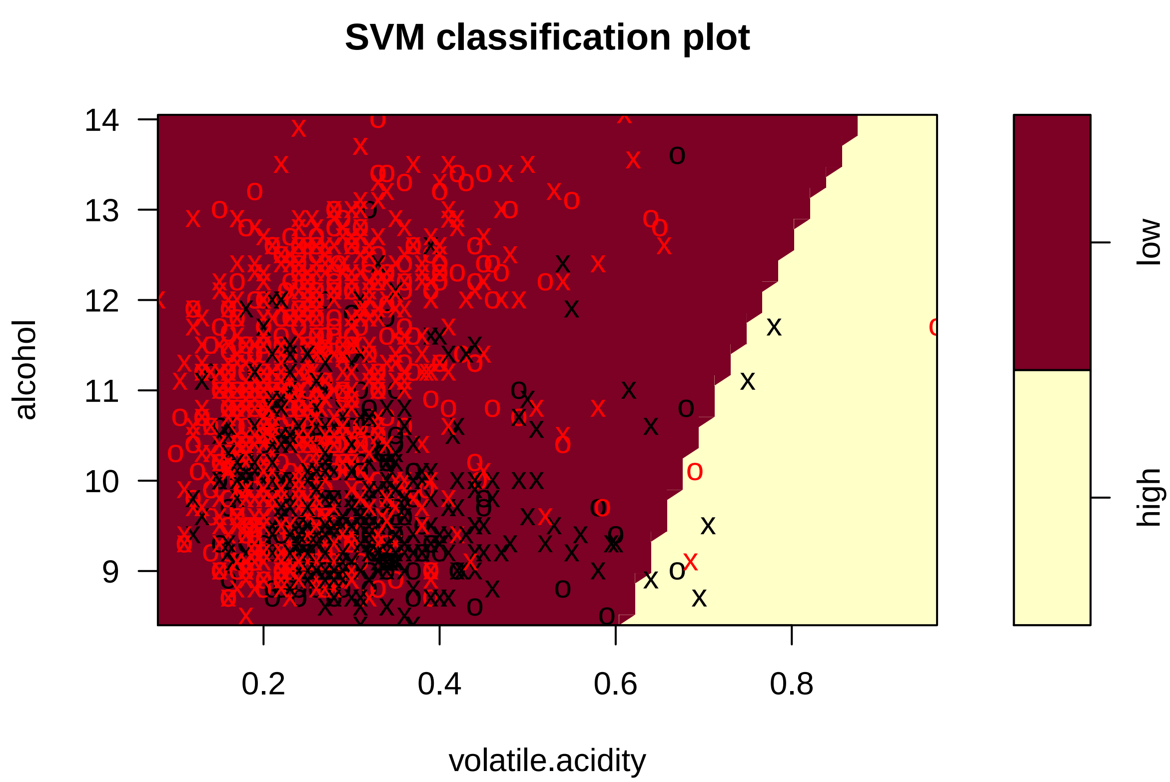
```
pred = predict(svmPolynomial,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high    6   3
##   low   486 974
```

```
mean(pred==test$quality)
```

```
## [1] 0.6671205
```

```
plot(svmPolynomial,test[,c('quality','alcohol','volatile.acidity')])
```



Polynomial SVM - Tuned

```
svmPolynomialTune = tune(method = svm,quality~alcohol+volatile.acidity,data=train,kernel='polynomial',
                         ranges=list(cost=c(0.01,1,100),degree=c(2,3)))
summary(svmPolynomialTune$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = quality ~ alcohol + volatile.acidity,
##     data = train, ranges = list(cost = c(0.01, 1, 100), degree = c(2,
##         3)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  2091
##
##  ( 1046 1045 )
##
##
## Number of Classes:  2
##
## Levels:
##  high low
```

```
svmPolynomialTune$best.parameters
```

```
##   cost degree
## 5    1      3
```

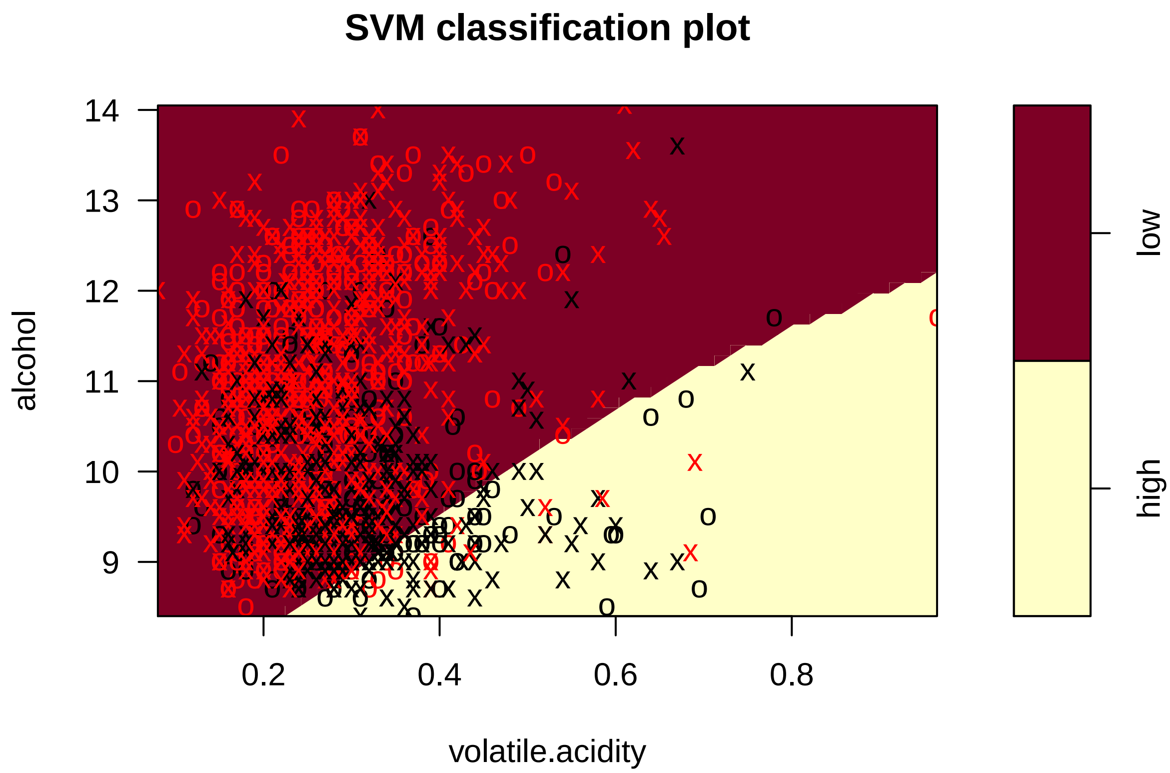
```
pred = predict(svmPolynomialTune$best.model,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high   85  19
##   low   407 958
```

```
mean(pred==test$quality)
```

```
## [1] 0.7100068
```

```
plot(svmPolynomialTune$best.model,test[,c('quality','alcohol','volatile.acidity')])
```



Radial SVM

```
svmRadial = svm(quality~alcohol+volatile.acidity,data = train,kernel='radial',type='C-classification')
summary(svmRadial)
```

```
##
## Call:
## svm(formula = quality ~ alcohol + volatile.acidity, data = train,
##     kernel = "radial", type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  1835
##
##  ( 925 910 )
##
##
## Number of Classes:  2
##
## Levels:
##  high low
```

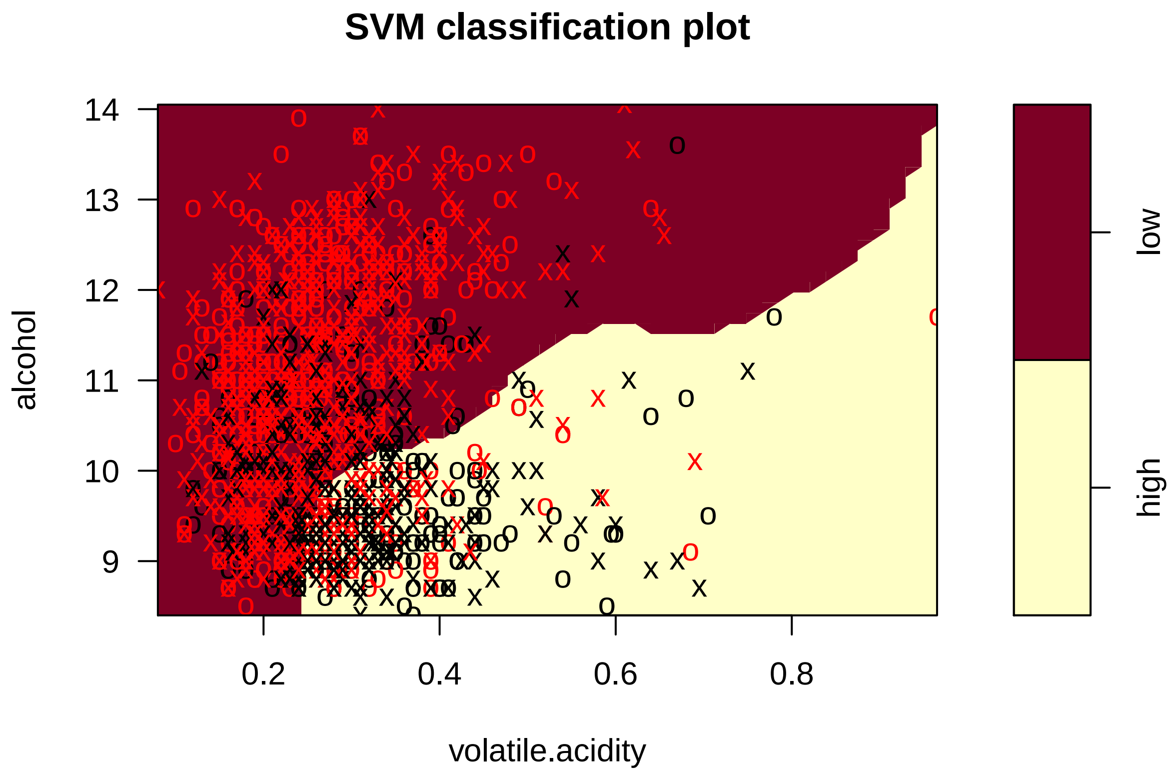
```r
pred = predict(svmRadial,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high  232 124
##   low   260 853
```

```r
mean(pred==test$quality)
```

```
## [1] 0.7385977
```

```r
plot(svmRadial,test[,c('quality','alcohol','volatile.acidity')])
```



Radial SVM - Tuned

```r
svmRadialTune = tune(method=svm,quality~alcohol+volatile.acidity,data=train,kernel='radial',type='C-clas
                     ranges = list(cost=c(0.1,10,100), gamma=c(1,10), coef0 = c(0.1,1,10)))
summary(svmRadialTune$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = quality ~ alcohol + volatile.acidity,
```

```
##     data = train, ranges = list(cost = c(0.1, 10, 100), gamma = c(1,
##         10), coef0 = c(0.1, 1, 10)), kernel = "radial", type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  1825
##
##  ( 936 889 )
##
##
## Number of Classes:  2
##
## Levels:
##  high low
```

```r
svmRadialTune$best.parameters
```

```
##   cost gamma coef0
## 2   10     1   0.1
```

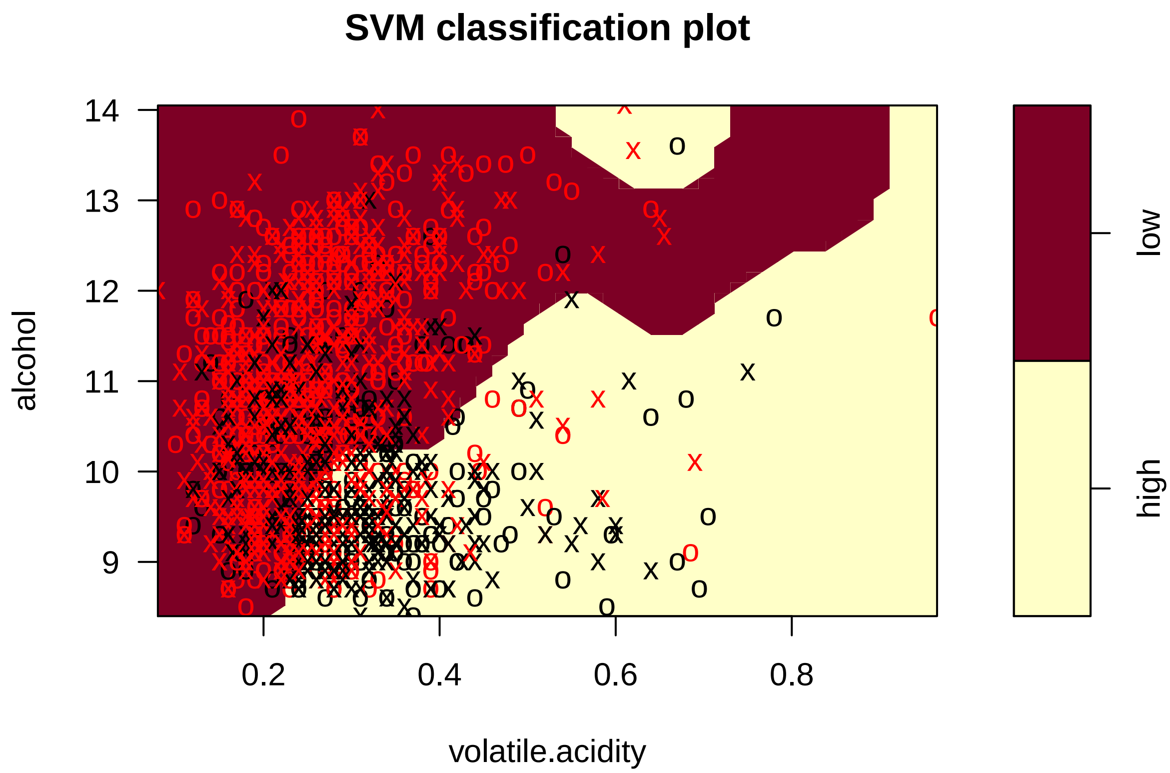
```r
pred = predict(svmRadialTune$best.model,newdata=test)
table(pred,test$quality)
```

```
##
## pred   high low
##   high  255 133
##   low   237 844
```

```r
mean(pred==test$quality)
```

```
## [1] 0.748128
```

```r
plot(svmRadialTune$best.model,test[,c('quality','alcohol','volatile.acidity')])
```



Hopefully, the simulated data and the wine example help illustrate the use of Support Vector Machines for classification.