9 SVM

By: Udit (based on ISLR)

Setup

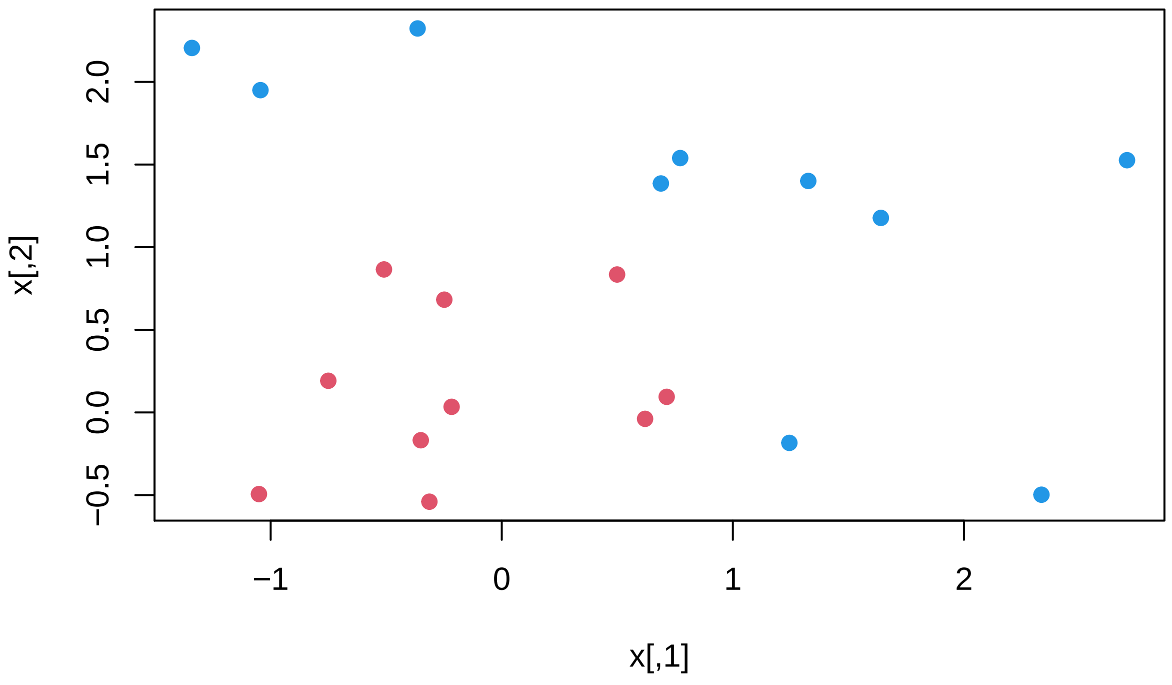
The way **Cost** is implemented in model fitting, is different from how it is explained in the book. In the text $Cost = \sum e_i$ implying higher margin as cost **increases**. However, in code it is a regularization term similar to ridge or lasso regressions: $Cost * \sum e_i$, implying higher margin as cost **decreases**. * StackExchange link

```
library(ISLR2)
library(e1071) # for SVM
library(ROCR)  # for ROC curves
```

Linear SVM Classifier - classes are NOT linearly separable

SVM fit also provides support points under **$index**. **Tune()** function allows us to run cross-validation.
SVM does not readily provide coefficients from the fit.

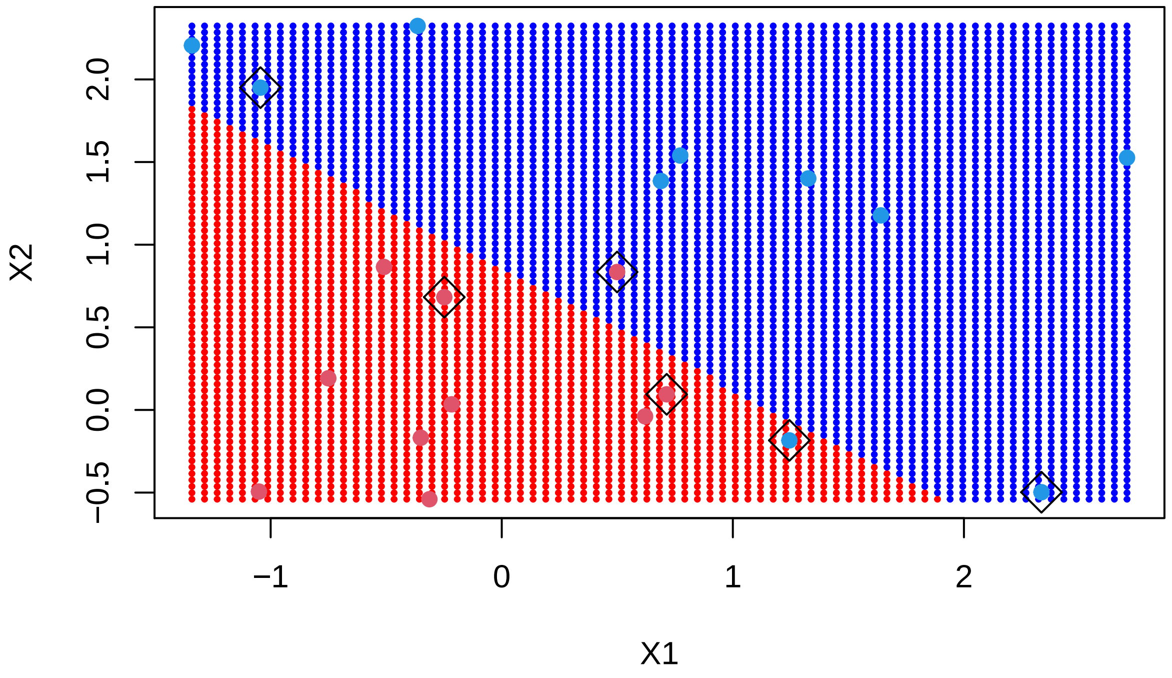
```
# Function to get grid for plotting
make.grid = function(x,n=75){
  xrange = apply(x,2, range)
  x1=seq(from=xrange[1,1], to=xrange[2,1],length=n)
  x2=seq(from=xrange[1,2], to=xrange[2,2],length=n)
  expand.grid(X1=x1, X2=x2)
}

# Simulated data
set.seed(10111)
x=matrix(rnorm(40),20,2)
y=rep(c(-1,1),c(10,10))

# Shifting 'means'
x[y==1,] = x[y==1,]+1

# Classes are NOT linearly separable
plot(x,col=y+3,pch=19)
```



```r
# Fitting Linear SVM with Cost
dat = data.frame(x,y=as.factor(y))
fit.svm = svm(y~., data=dat, kernel="linear", cost=10, scale=FALSE)
print(fit.svm)
```

```
##
## 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:  6
```

```r
# plot(fit.svm, dat) # produces an ugly looking plot

# Plot result
x.grid = make.grid(x)
y.grid = predict(fit.svm, x.grid)
plot(x.grid, col=c("red","blue")[as.numeric(y.grid)], pch=20, cex=.5)
points(x, col=y+3, pch=19)
points(x[fit.svm$index,], pch=5, cex=2)
```



```r
# Result using another value for cost function
summary(svm(y~., data=dat, kernel="linear", cost=1, scale=FALSE))
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  10
##
##  ( 5 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
# Cross Validation on "Cost" parameter using tune()
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
##   100
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.15  0.2415229
## 4 1e+00  0.20  0.2581989
## 5 5e+00  0.20  0.2581989
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.10  0.2108185
```

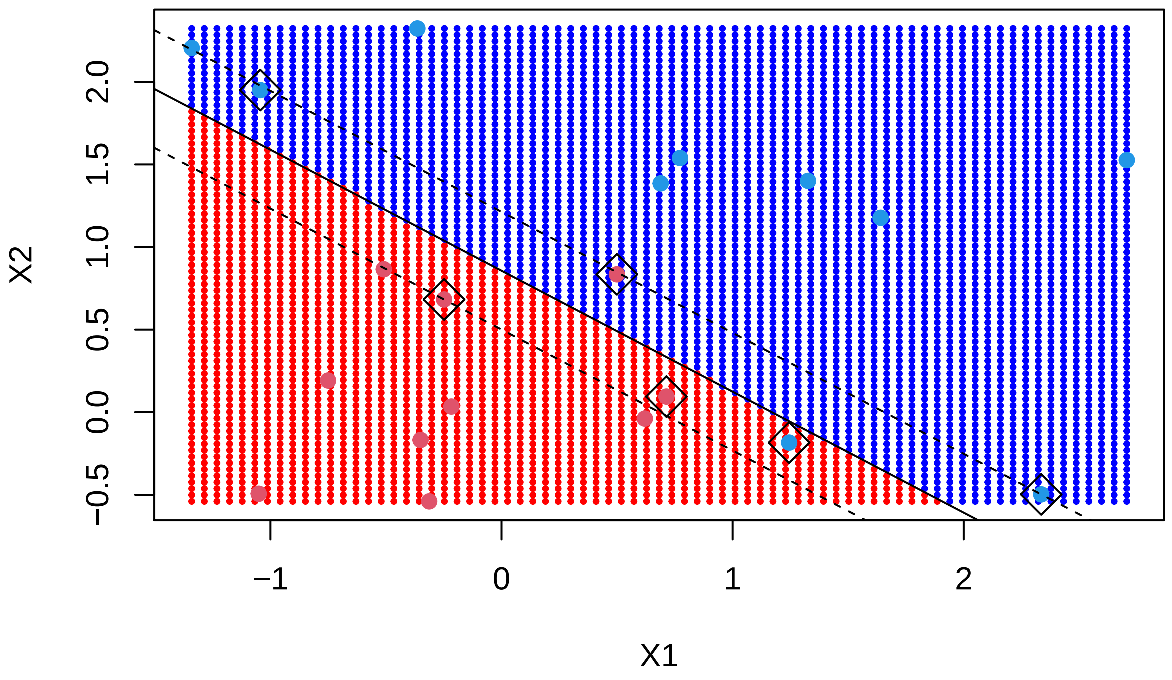
```
# Tune provides best fit model
best.mod = tune.out$best.model
summary(best.mod)
```

```
##
## 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:  100
##
## Number of Support Vectors:  5
##
##  ( 2 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
# Making predictions
x.test = matrix(rnorm(40),20,2)
y.test = rep(c(-1,1),c(10,10))
x.test[y.test==1,] = x.test[y.test==1,]+1
dat.test = data.frame(X1 = x.test[,1], X2 = x.test[,2], y = as.factor(y.test))
y.pred = predict(best.mod, dat.test)
table(pred=y.pred, truth = dat.test$y)
```

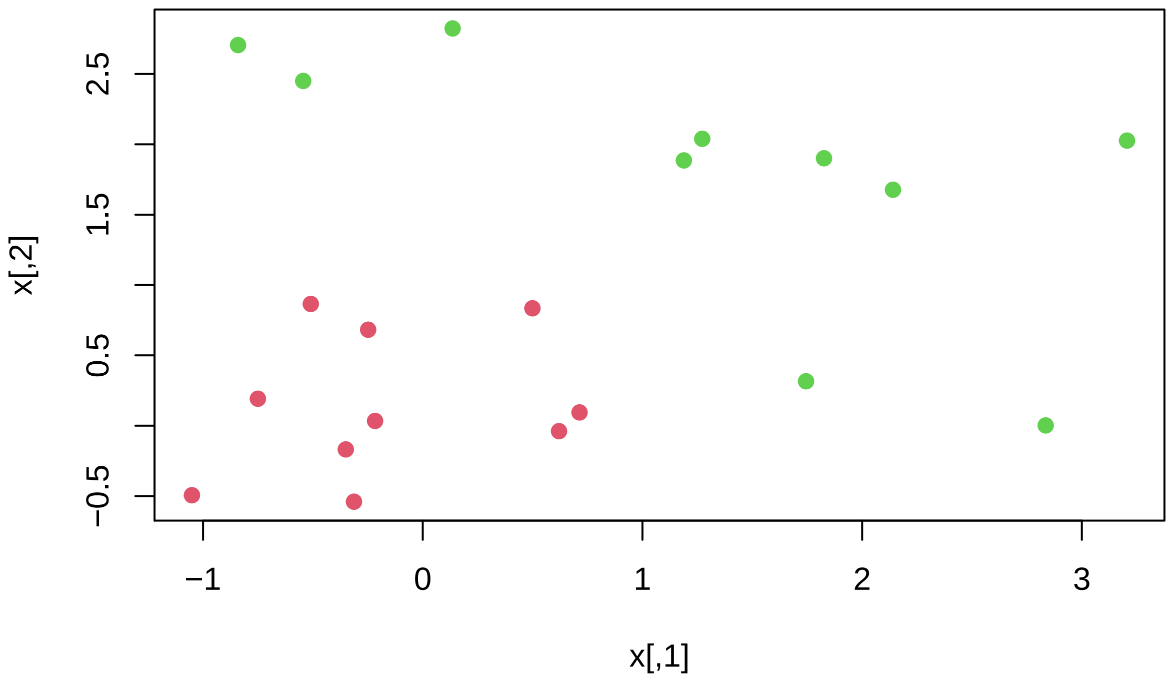
```
##     truth
## pred -1  1
##   -1 10  2
##   1   0  8
```

```r
# Extracting Coefficients
beta = drop(t(fit.svm$coefs)%*%x[fit.svm$index,])
beta0 = fit.svm$rho
plot(x.grid, col=c("red","blue")[as.numeric(y.grid)], pch=20, cex=0.5)
points(x, col=y+3, pch=19)
abline(beta0/beta[2], -beta[1]/beta[2])  # intercept and slope
abline((beta0-1)/beta[2], -beta[1]/beta[2], lty=2)
abline((beta0+1)/beta[2], -beta[1]/beta[2], lty=2)
points(x[fit.svm$index,], pch=5, cex=2)
```



Linear SVM Classifier - classes ARE linearly separable

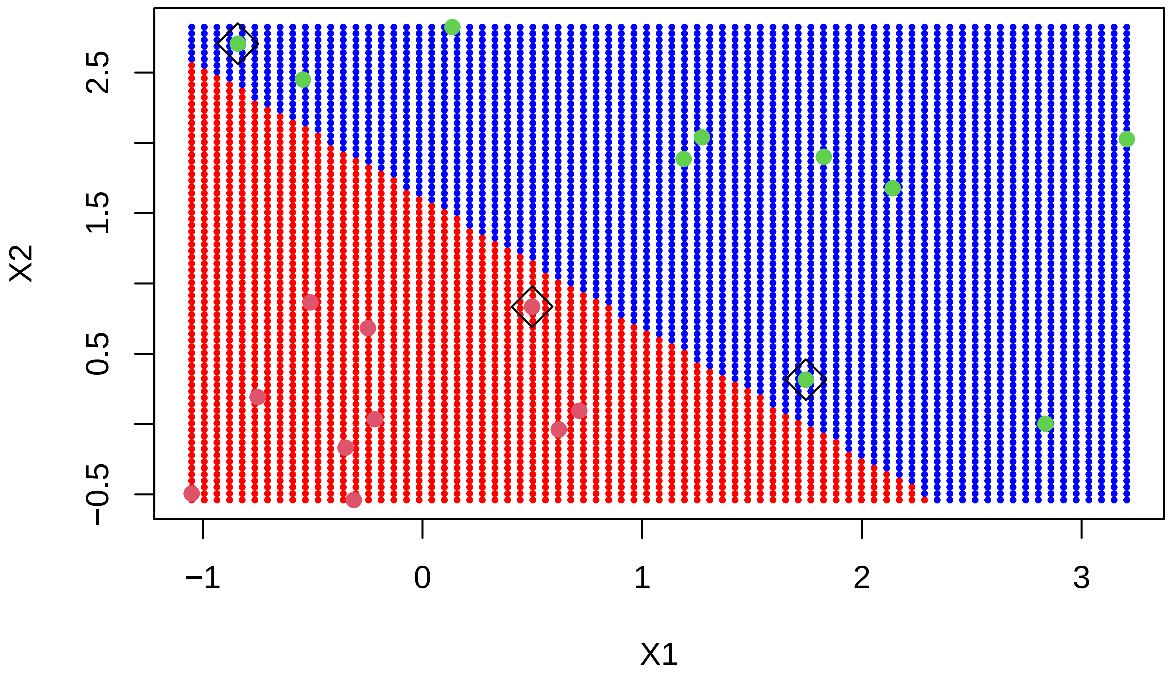
```r
# Shifting 'means' a little more to make classes separable
x[y==1,] = x[y==1,]+.5

# Classes ARE linearly separable now
plot(x, col= (y+5)/2, pch=19)
```



```
# Fitting Linear SVM
dat.sep = data.frame(X1=x[,1], X2 = x[,2], y = as.factor(y))
fit.svm.sep = svm(y~., data=dat.sep, kernel = "linear", cost = 1e5)
summary(fit.svm.sep)
```

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

```
# Plot
x.grid = make.grid(x)
y.grid = predict(fit.svm.sep, x.grid)
plot(x.grid, col=c("red","blue")[as.numeric(y.grid)], pch=20, cex=.5)
points(x, col=(y+5)/2, pch=19)
points(x[fit.svm.sep$index,], pch=5, cex=2)
```



While the classes are linearly separable, we get a fit with small margin, which is highly dependent on position for those 3 support vectors - and by extension has high variance, and possibly won't do as well on test data.

SVM - Non-linear Kernel

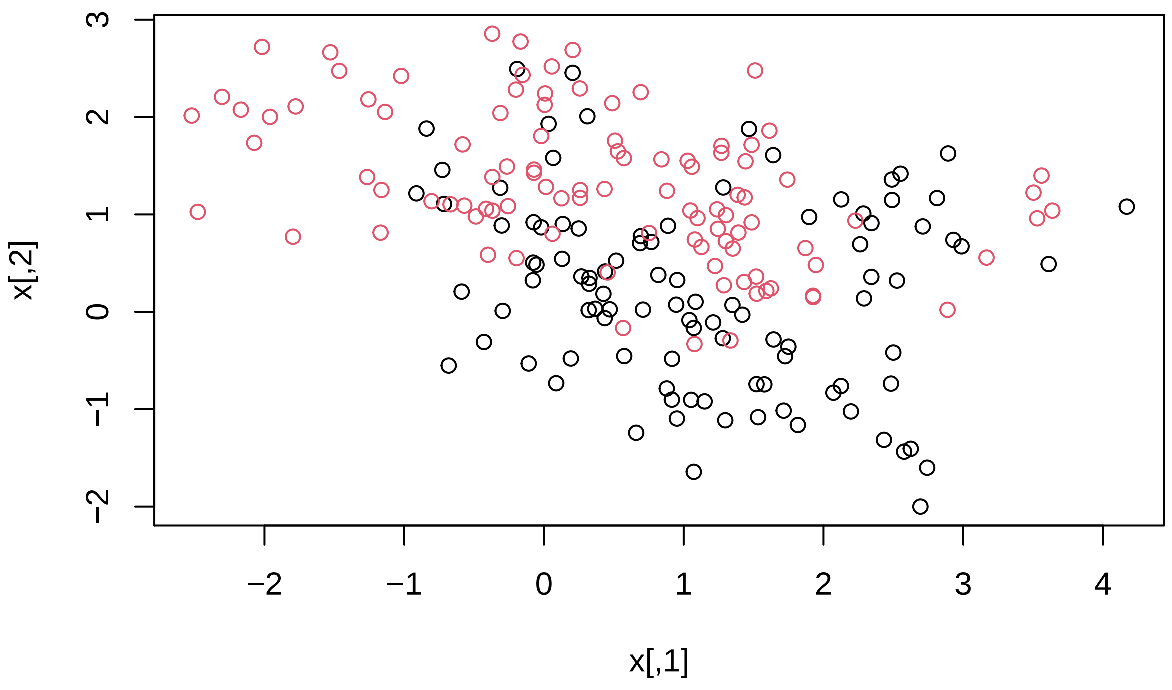
Example 1

Decision boundary comes in pretty close to the **true** boundary, esp. in the the regions where data exists.

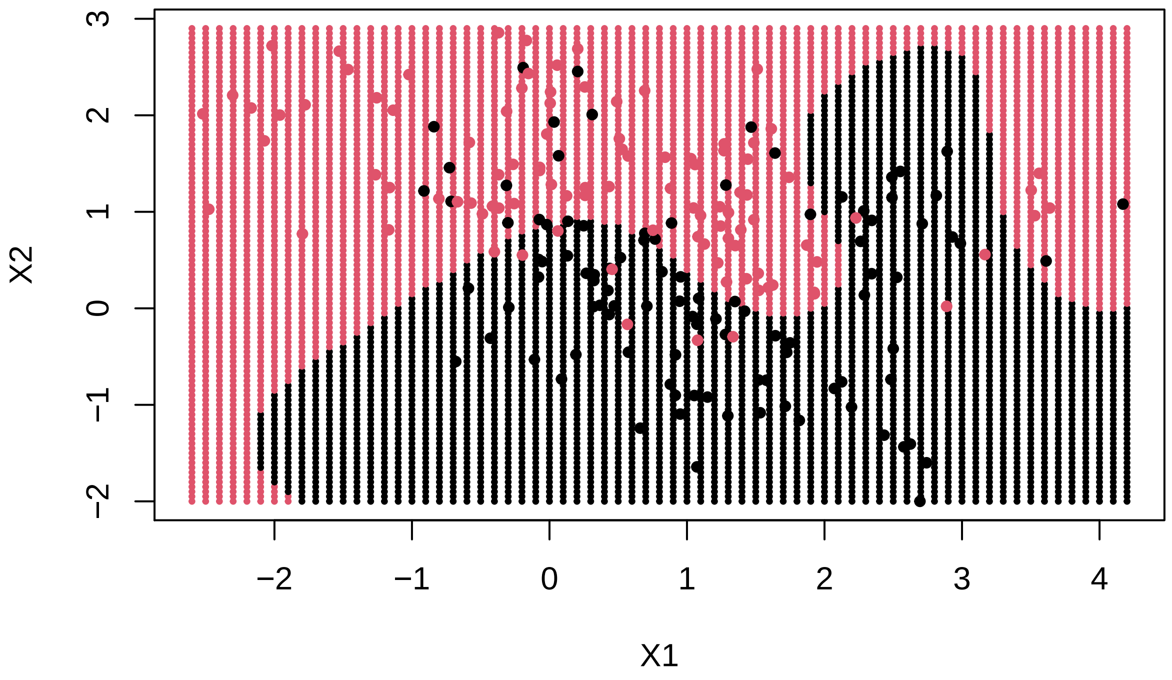
```
load(url("http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/ESL.mixture.rda"))
names(ESL.mixture)
```

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

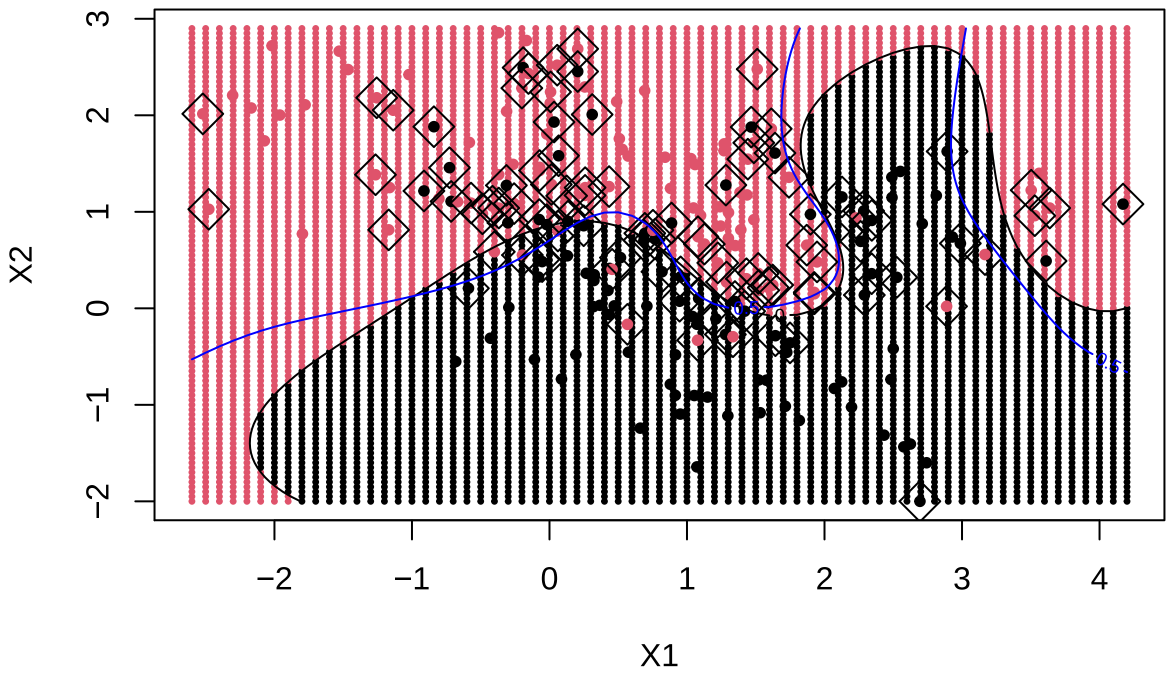
```
rm(x,y); attach(ESL.mixture)

# Fitting non-linear SVM
plot(x,col=y+1)
```



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

# Plotting
x.grid = expand.grid(X1=px1, X2=px2)  # px1, px2 are part of ESL.mixture
y.grid = predict(fit.nl, x.grid)
plot(x.grid, col=as.numeric(y.grid), pch=20, cex=0.5)
points(x, col=y+1, pch=20)
```

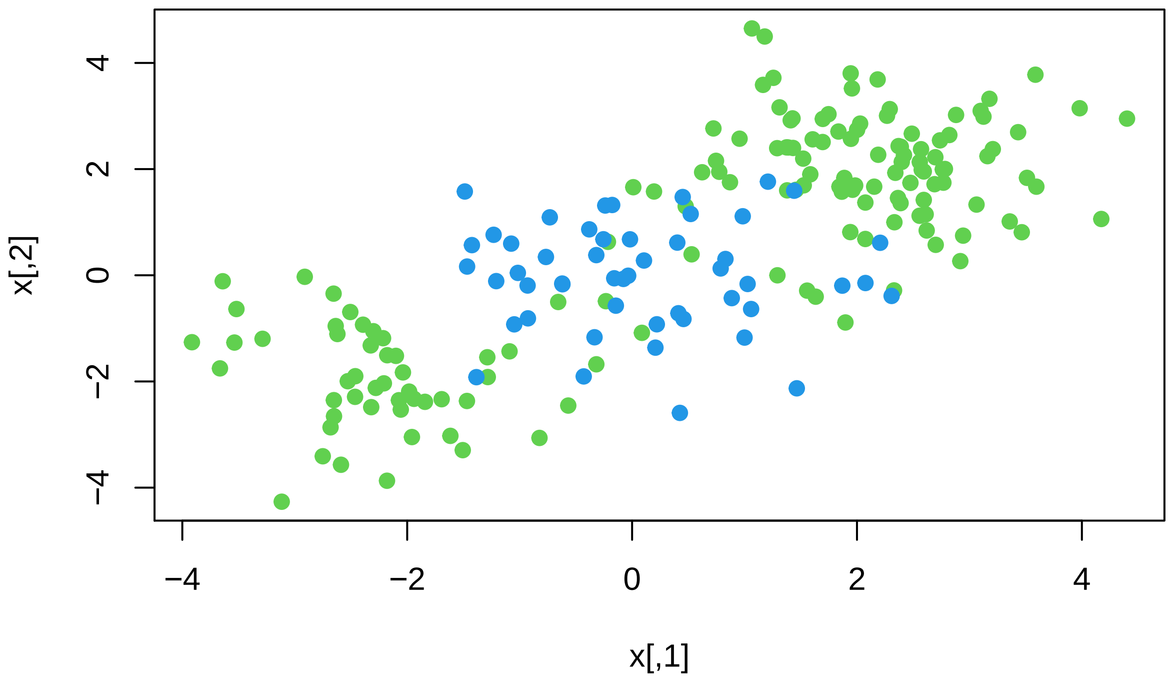


```
pred.f = predict(fit.nl, x.grid, decision.values=TRUE) # function value, not class
pred.f = attributes(pred.f)$decision #function value is returned as an "attribute"

plot(x.grid, col=as.numeric(y.grid), pch=20, cex=0.5)
points(x, col=y+1, pch=20)
points(x[fit.nl$index,], pch=5, cex=2)
contour(px1,px2, matrix(pred.f,69,99), level=0, add=TRUE) # 0 is the threshold
contour(px1,px2, matrix(prob,69,99), level=0.5, add=TRUE, col="blue") #truth
```



Example 2

```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))
dat = data.frame(X1 = x[,1], X2 = x[,2], y = as.factor(y))

plot(x, col=y+2, pch=19)
```



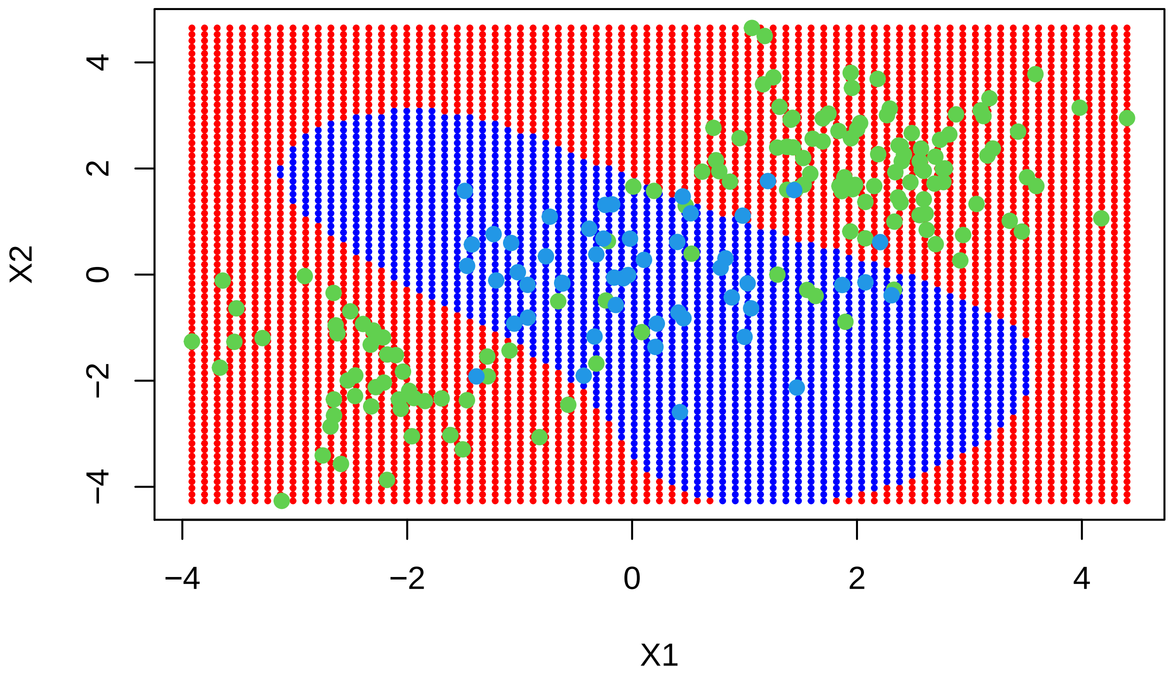
```
train = sample(200,100)
svm.radial = svm(y~., data=dat[train,], kernel="radial", gamma=1, cost=1)
summary(svm.radial)
```

```
##
## 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
```

```
table(dat$y[svm.radial$index])
```

```
##
##  1  2
## 31  0
```

```
#plot(svm.radial, dat[train,])

x.grid = make.grid(x)
y.grid = predict(svm.radial, x.grid)
plot(x.grid, col=c("red","blue")[as.numeric(y.grid)], pch=20, cex=.5)
points(x, col=y+2, pch=19)
```

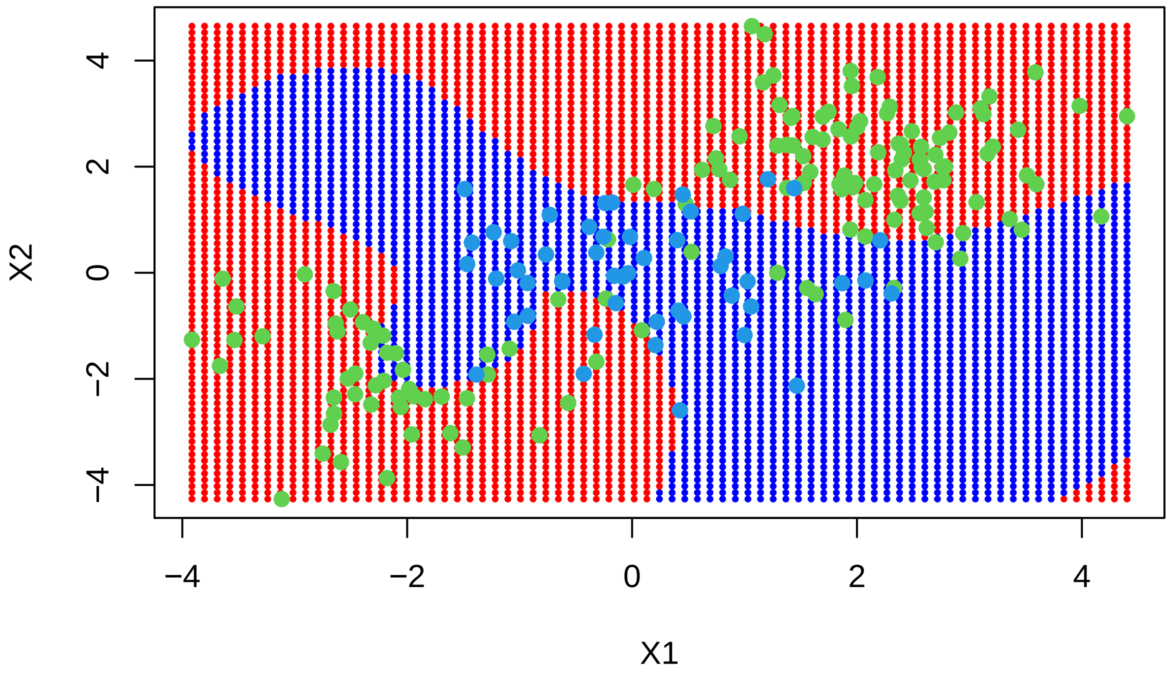


```
#points(x[svm.radial$index,], pch=5, cex=2)

# Increasing Cost to reduce training error - more irregular boundary
svm.radial.2 = svm(y~., data=dat[train,], kernel="radial", gamma=1, cost=1e5)
summary(svm.radial.2)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1e+05
##
## Number of Support Vectors:  16
##
##  ( 7 9 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  1 2
```

```
y.grid.2 = predict(svm.radial.2, x.grid)
plot(x.grid, col=c("red","blue")[as.numeric(y.grid.2)], pch=20, cex=.5)
points(x, col=y+2, pch=19)
```



```
# Cross-validation to calibrate 'Gamma' and 'Cost'
svm.tune = 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(svm.tune)
```

```
##
## 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
```

```
# Predict using CV selected model
table(
  true = dat[-train,"y"],
  pred = predict(svm.tune$best.model, newdata=dat[-train,])
)
```

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

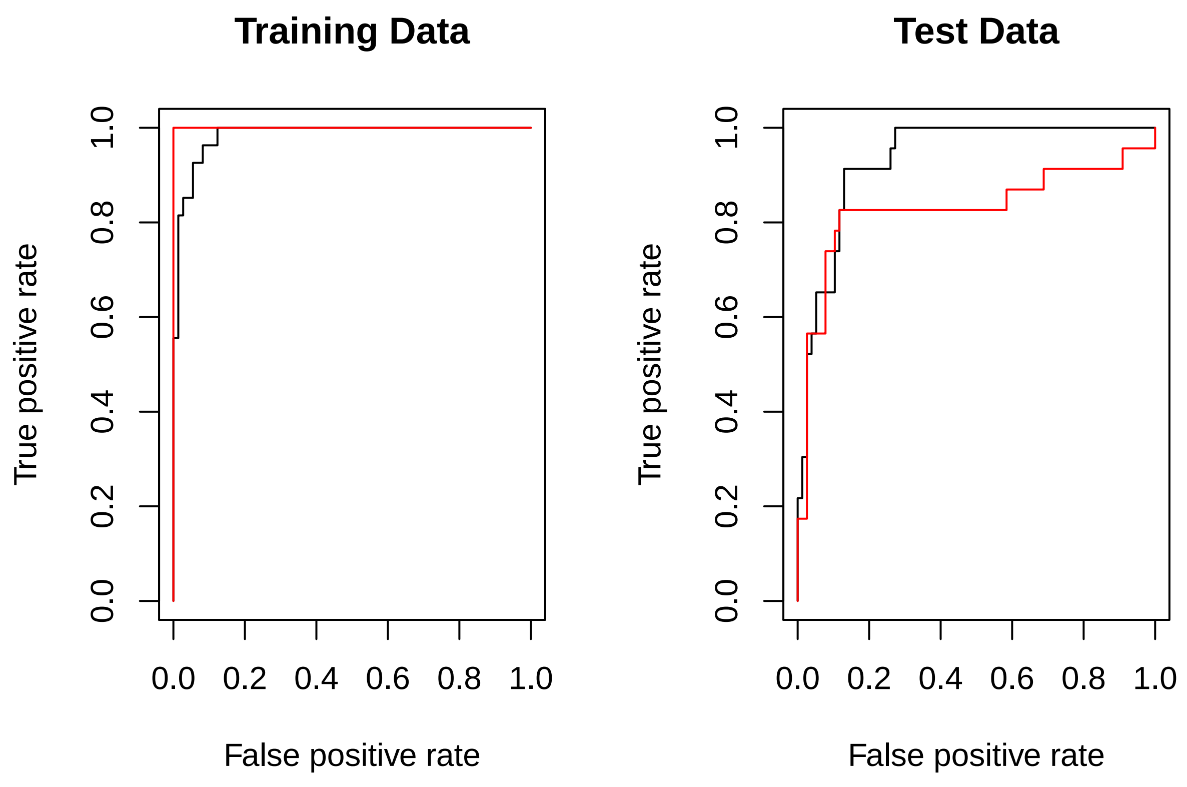
```
# Error of ~11%
1-88/(77+22)
```

```
## [1] 0.1111111
```

ROC Curves

performance() - all kinds of predictor evaluations are performed using this.
prediction() - Function to create prediction objects

```
rocplot = function(pred, truth,...){
  pred.obj = prediction(pred, truth)
  perf     = performance(pred.obj, "tpr", "fpr")
  plot(perf, ...)
}

par(mfrow=c(1,2))

# Fitting & Prediction on Training Data
svmfit.opt = svm(y~., data=dat[train,], kernel="radial",
                 gamma=0.5, cost=1, decision.values=TRUE)
fitted = attributes(
         predict(svmfit.opt, dat[train,], decision.values=TRUE))$decision.values
rocplot(-fitted, dat[train,"y"], main= "Training Data")
#So that -ve values correspond to Class 1 and +ve values to Class 2

# Fitting with higher 'Gamma' for more flexible fit
svmfit.flex = svm(y~., data=dat[train,], kernel="radial",
                 gamma=50, cost=1, decision.values=TRUE)
fitted.flex = attributes(
         predict(svmfit.flex, dat[train,], decision.values=TRUE))$decision.values
rocplot(-fitted.flex, dat[train,"y"], add=T, col="red")

# Fitting on Test data
fitted.test = attributes(
         predict(svmfit.opt, dat[-train,], decision.values=TRUE))$decision.values
rocplot(-fitted.test, dat[-train,"y"], main="Test Data")

fitted.test.flex = attributes(
         predict(svmfit.flex, dat[-train,], decision.values=TRUE))$decision.values
rocplot(-fitted.test.flex, dat[-train,"y"], add=T, col="red")
```

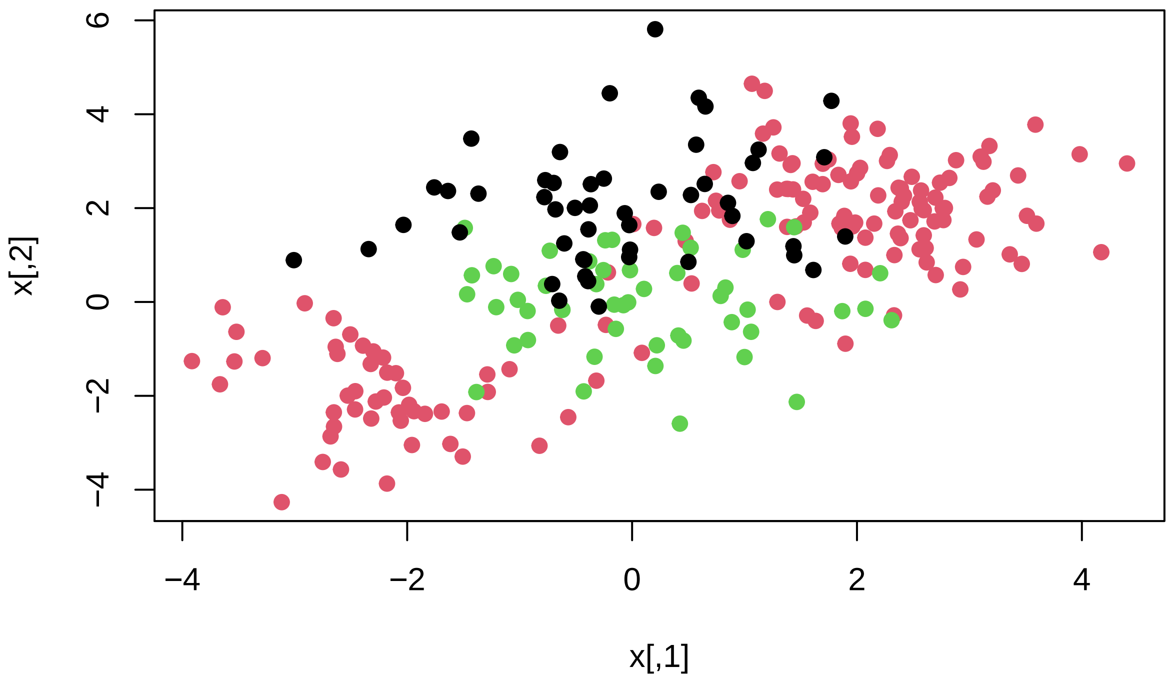


SVM - Multiple Class Classification

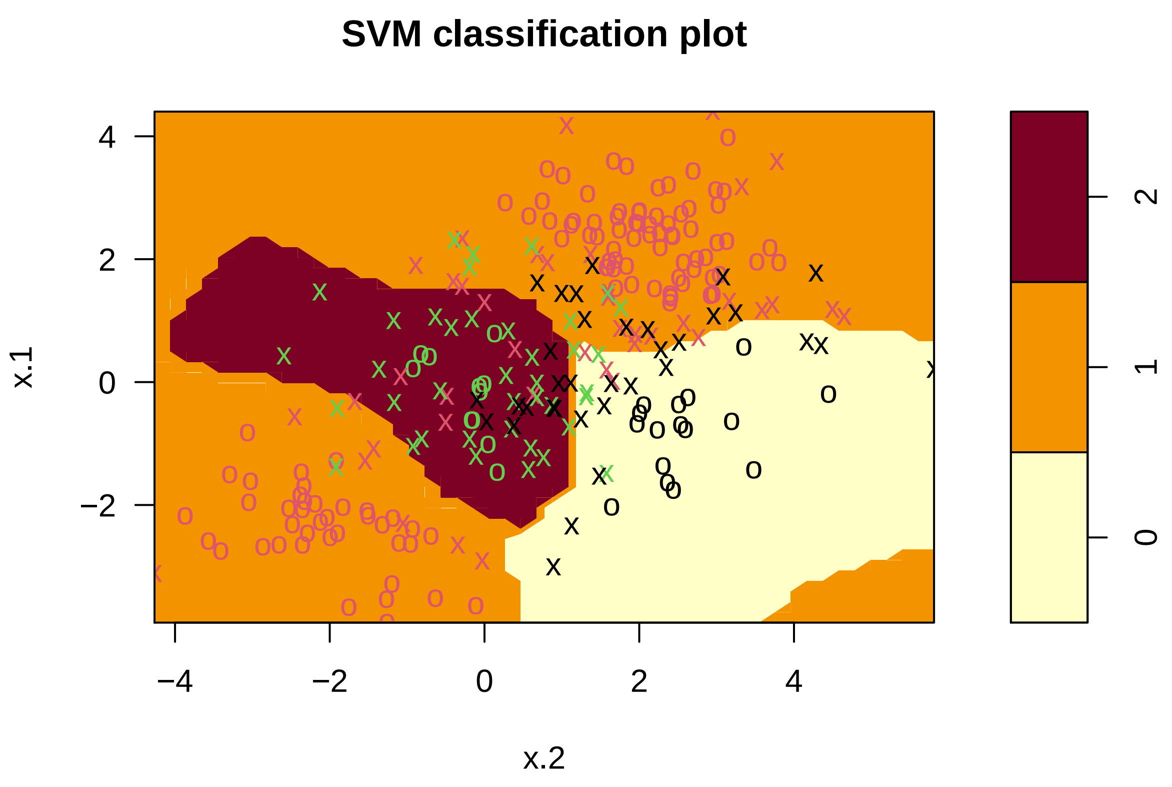
Either one-vs-one approach or one-vs-all approach can be used.

```
set.seed(1)
x = matrix(rnorm(200*2), ncol=2)
x[1:100,] = x[1:100,] + 2
```

```
x[101:150,] = x[101:150,] - 2
x = rbind(x, matrix(rnorm(50*2), ncol=2))
y = c(rep(1,150), rep(2,50), 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, pch=19)
```



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



## Application to Gene Expression Data

```
names(Khan)
```

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

```
dim(Khan$xtrain); dim(Khan$xtest)
```

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

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

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

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

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

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

```
# Due to 'p' >> 'n', we select linear kernel to avoid allowing more flexibility
dat = data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))
svm = svm(y~., data=dat, kernel="linear", cost=1e5)
summary(svm)
```

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

```r
table(svm$fitted, dat$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
```

```r
# Test data
pred = predict(svm, data.frame(x = Khan$xtest, y = as.factor(Khan$ytest)))
table(pred=pred, true= Khan$ytest)
```

```
##     true
## pred 1 2 3 4
##    1 3 0 0 0
##    2 0 6 2 0
##    3 0 0 4 0
##    4 0 0 0 5
```

Quiz question

```r
library(MASS) # for multivariate normal distribution
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```r
set.seed(42)
error.svm = double(50)
error.log = double(50)
```

```r
for(i in 1:50){

# 50 obs for TRAINING, 200 for TESTING
x0 = mvrnorm(n=250, mu=rep(0, 10),          Sigma = diag(10))
x1 = mvrnorm(n=250, mu=rep(c(1,0),each=5), Sigma = diag(10))
y0 = rep(0,250)
y1 = rep(1,250)

y.train = c(y0[1:50],y1[1:50])
x.train = rbind(x0[1:50,],x1[1:50,])
dat.train = data.frame(y = y.train, x = x.train)
dat.test  = data.frame(y = c(y0[51:250],y1[51:250]),
                      x = rbind(x0[51:250,],x1[51:250,]))

# fitting SVM - radial or linear
fit.svm  = svm(factor(y)~., data=dat.train, kernel="linear")
pred.svm = predict(fit.svm, dat.test)

# fitting logistic
fit.log  = glm(factor(y)~., data=dat.train, family="binomial")
pred.log = predict(fit.log, newdata=dat.test, type="response")
pred.log = ifelse(pred.log>0.5,1,0)

error.svm[i] = 1-mean(c(y0[51:250],y1[51:250])==pred.svm)
error.log[i] = 1-mean(c(y0[51:250],y1[51:250])==pred.log)
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
mean(error.svm)   #0.16 for radial, 0.15 for linear
```

```
## [1] 0.1634
```

```r
mean(error.log)   #0.15 .. logistic is similar to SVM with linear kernel
```

```
## [1] 0.1597
```