

Support Vector Machine

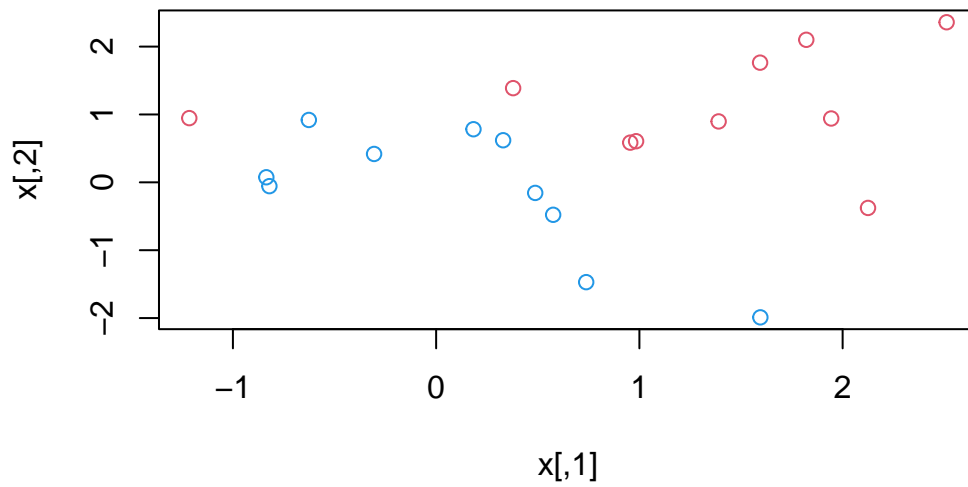
Madhur

Support Vector Classifier

```
library(e1071)
```

Create a Dataset

```
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))
```



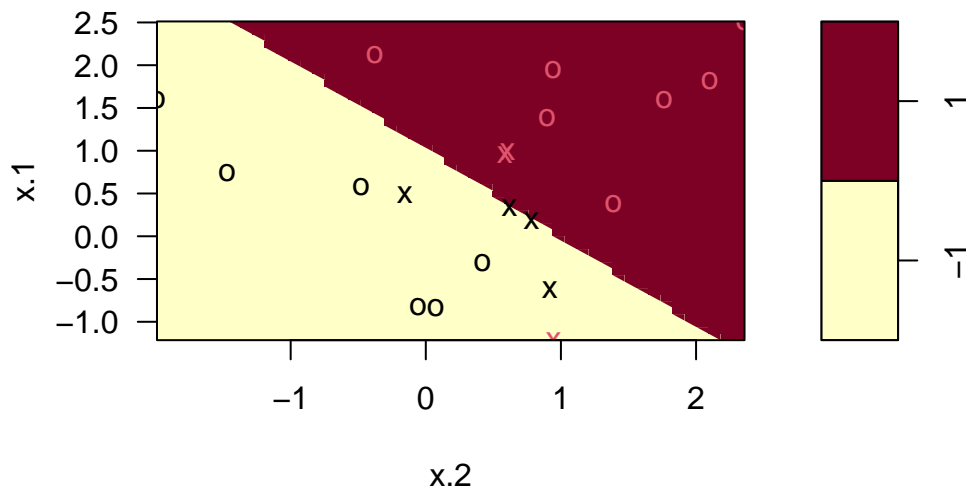
Nont linearly seperable two classes.

```
dat <- data.frame(x = x, y = as.factor(y))
```

Fitting a Linear SVM model with cost 10

```
svm.fit <- svm(y~. , data = dat, kernel = "linear",  
              cost = 10, scale = FALSE)  
plot(svm.fit, data = dat)
```

SVM classification plot



- The decision boundary is linear.
- The support vectors are plotted as crosses and remaining observations as circles. We can count the support vectors.

```
svm.fit$index
```

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

```
summary(svm.fit)
```

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

Fitting a Linear SVM model with cost 0.1

```
svm.fit <- svm(y~. , data = dat, kernel = "linear",  
              cost = 0.1, scale = FALSE)  
summary(svm.fit)
```

Call:

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

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

Cross Validation to select optimum cost

```
## Cross Validation to select optimum cost  
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

```
best.model <- tune.out$best.model
summary(best.model)
```

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

create 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(best.model, testdat)
table(predict = ypred, truth = testdat$y)
```

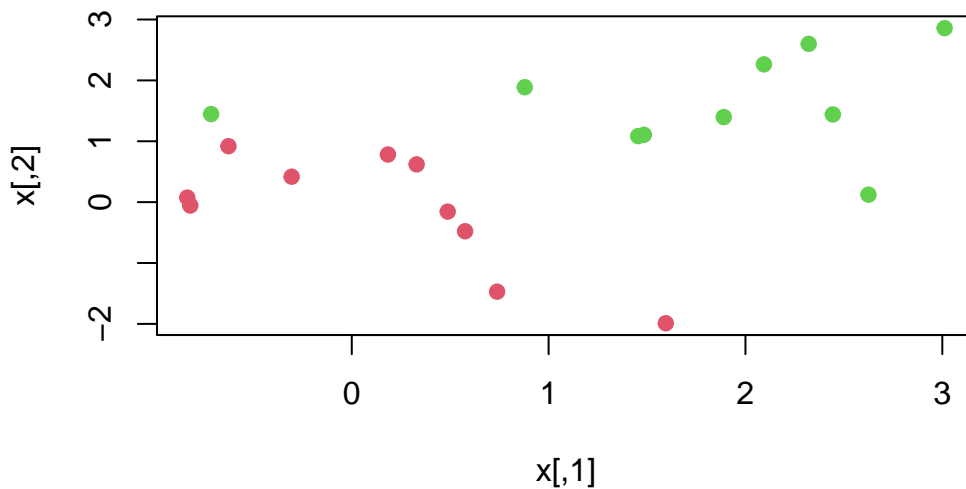
	truth
predict -1	1
-1	9 1
1	2 8

```
# try to build a model with 0.01 cost
```

```
svm.fit2 <- svm(y ~ ., data = dat, kernel = "linear",
               cost = 0.01, scale = FALSE)
y_pred2 <- predict(svm.fit2, testdat)
table(predict = y_pred2, truth = testdat$y)
```

	truth
predict -1	1
-1	11 6
1	0 3

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



```
# We fit the support vector classifier and plot the resulting hyperplane,
# using a very large value of cost so that no observations are misclassified.

dat <- data.frame(x = x, y = as.factor(y))

svm.fit <- svm(y ~ ., data = dat, kernel = "linear",
  cost = 1e5)

summary(svm.fit)
```

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
```

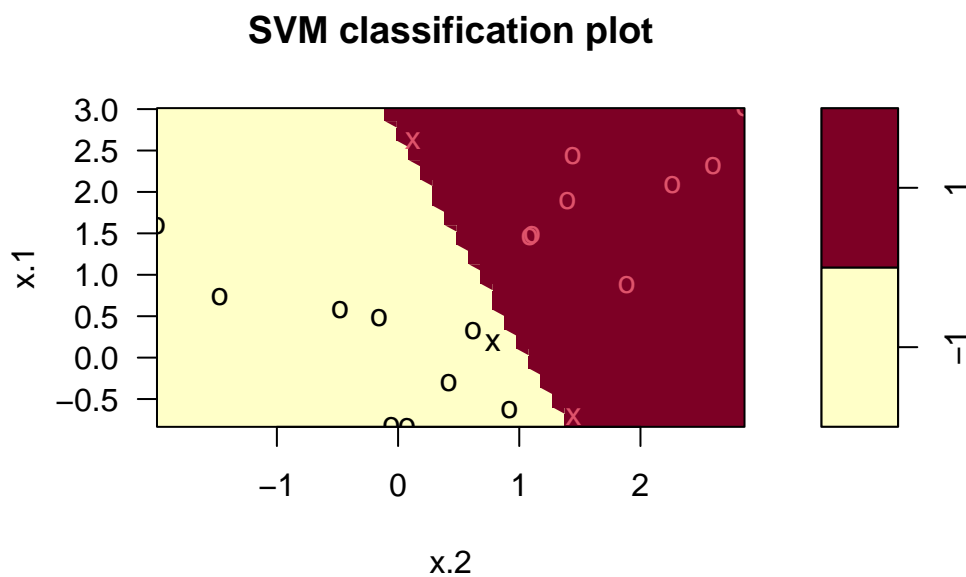
```
( 1 2 )
```

```
Number of Classes: 2
```

```
Levels:
```

```
-1 1
```

```
plot(svm.fit, dat)
```



```
# decrease the cost
```

```
svm.fit2 <- svm(y~. , data = dat, kernel = "linear",  
               cost = 1)
```

```
summary(svm.fit2)
```


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

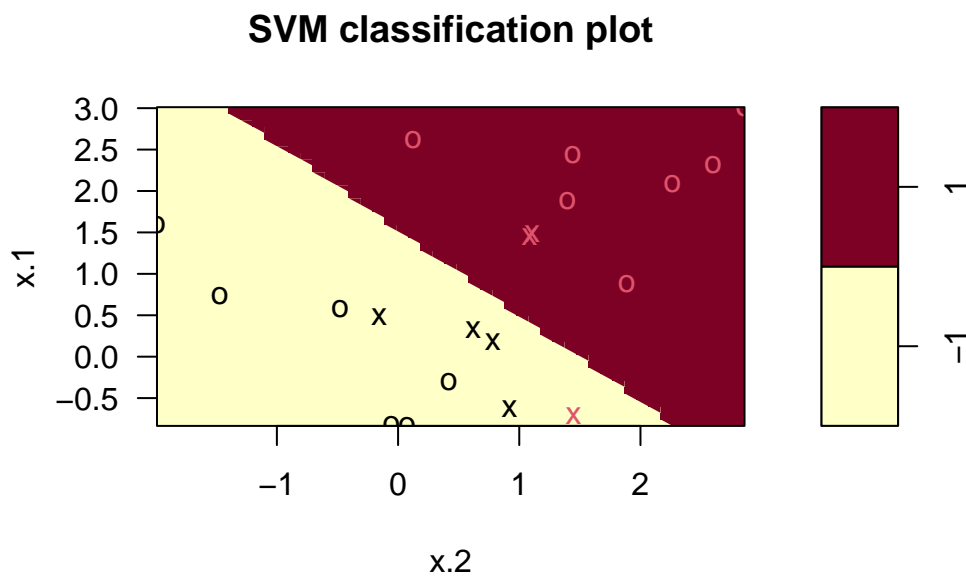
(4 3)

Number of Classes: 2

Levels:

-1 1

```
plot(svm.fit2, dat)
```



```
# Using cost = 1, we misclassify a training observation, but we also obtain a much wider margin
# It seems likely that this model will perform better on test data than the model with cost =
```

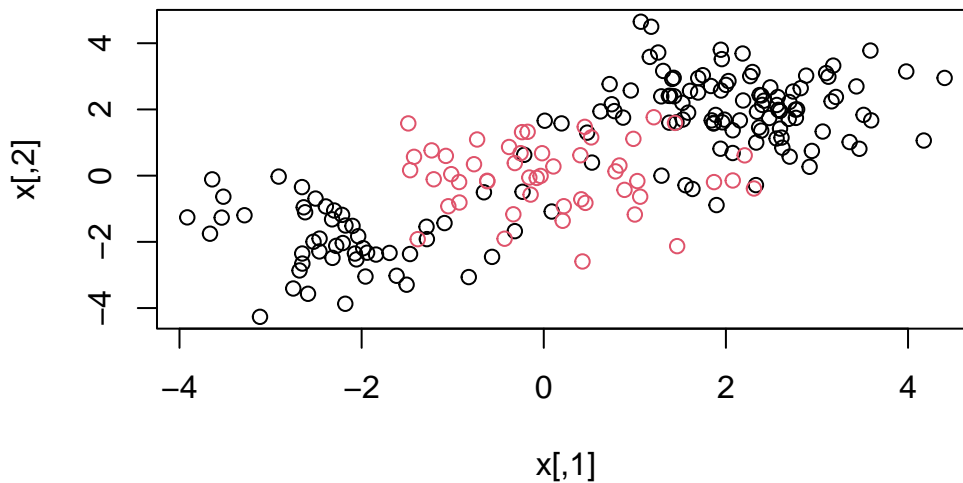
SVM - Non Linear Kernel -

1 polynomial - use degree argument

2 radial - use gamma

```
# generate data
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)
```

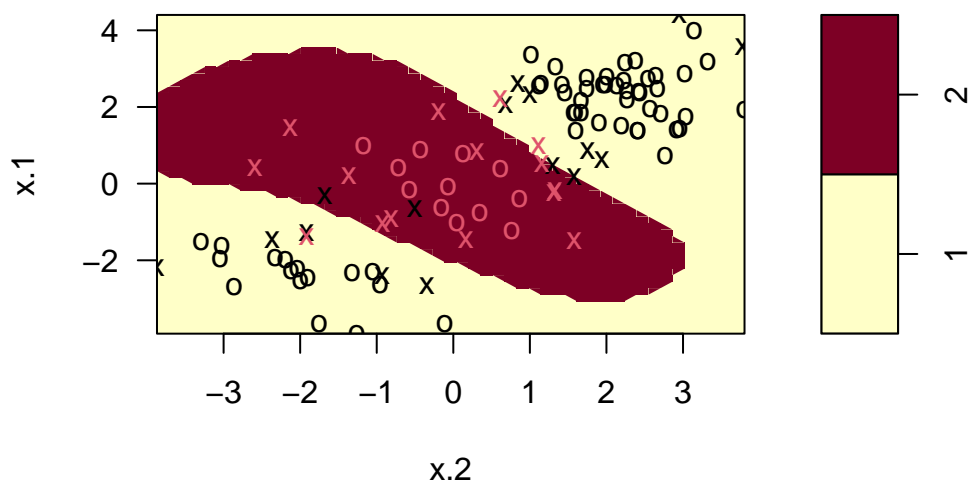


split data

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

plot(svm.fit, dat[train, ])
```

SVM classification plot



```
summary(svm.fit)
```

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:

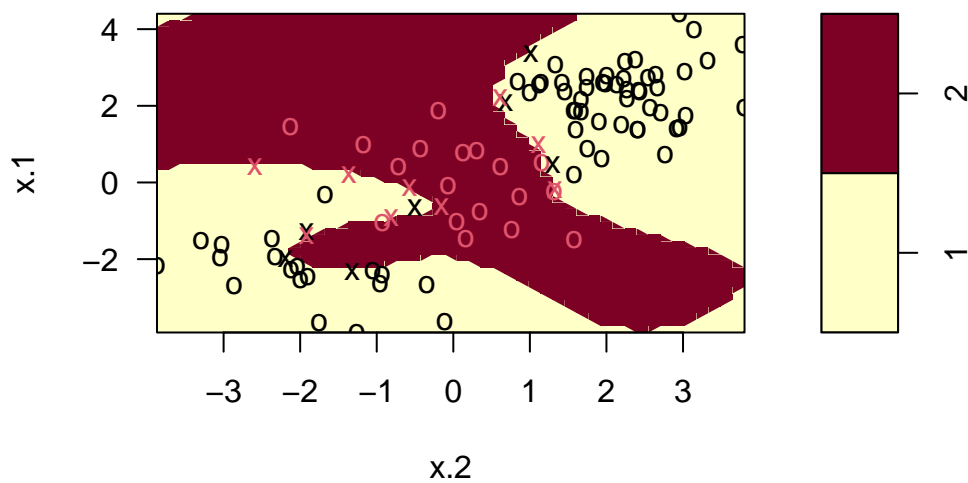
1 2

```
# increase cost
# increase cost : reduce misclassification errors
# but irregular decision boundary increase risk of overfitting

svm.fit2 <- svm(y ~., data = dat[train, ], kernel = "radial",
  gamma = 1, cost = 1e5)

plot(svm.fit2, dat[train, ])
```

SVM classification plot



```
summary(svm.fit2)
```

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
```

```
# cross-validation using tune() to select the best choice of gamma and cost for an SVM with a radial kernel
```

```
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

```

```

# predict using best model

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

```

```

      pred
true  1  2
     1 67 10
     2  2 21

```

ROC Curves

```

library(ROCR)

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

svm.fit.opt <- svm(y~., data = dat[train, ],
                  kernel = "radial", gamma = 2, cost = 1,
                  decision.values = T)

fitted <- attributes(
  predict(svm.fit.opt, dat[train, ], decision.values = TRUE)
)$decision.values

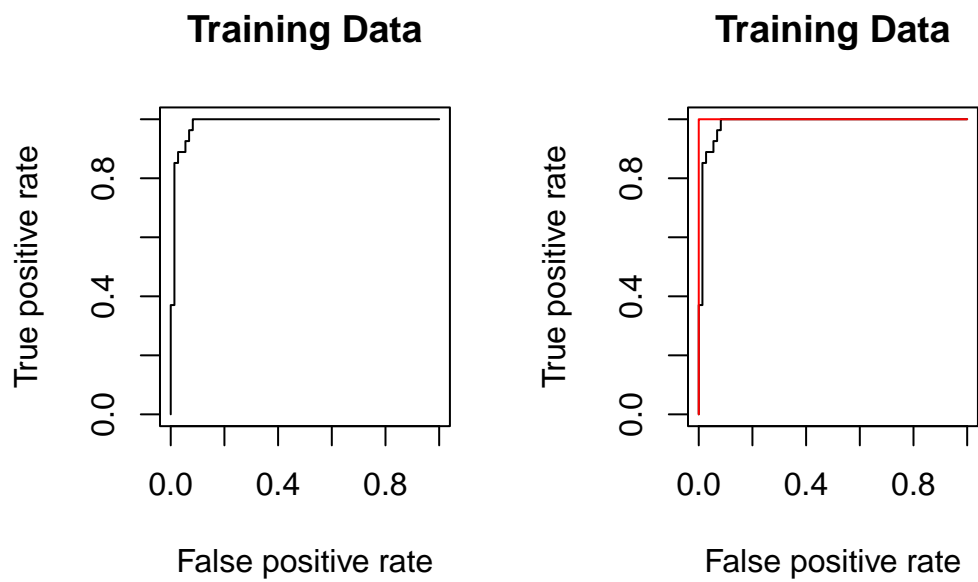
par(mfrow = c(1, 2))

rocplot(-fitted, dat[train, "y"], main = "Training Data")

# increase gamma

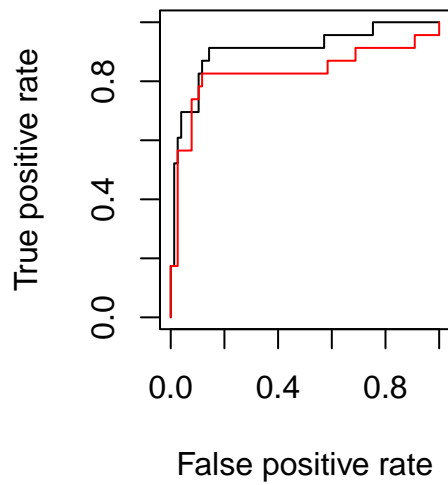
rocplot(-fitted, dat[train, "y"], main = "Training Data")
svmfit.flex <- svm(y ~ ., data = dat[train, ],
                  kernel = "radial", gamma = 50, cost = 1,
                  decision.values = T)
fitted <- attributes(
  predict(svmfit.flex, dat[train, ], decision.values = T)
)$decision.values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

```



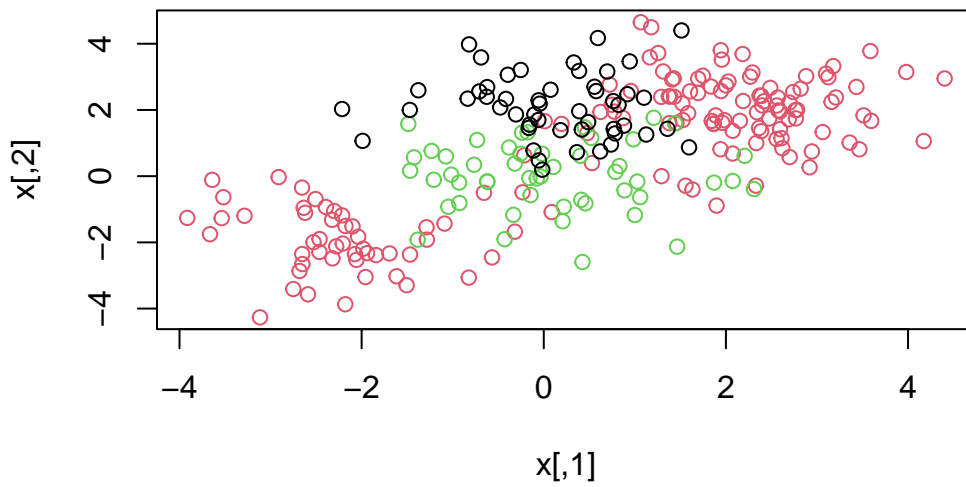
```
fitted <- attributes(
  predict(svm.fit.opt, dat[-train, ], decision.values = T)
)$decision.values
rocplot(-fitted, dat[-train, "y"], main = "Test Data")
fitted <- attributes(
  predict(svmfit.flex, dat[-train, ], decision.values = T)
)$decision.values
rocplot(-fitted, dat[-train, "y"], add = T,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


Test Data



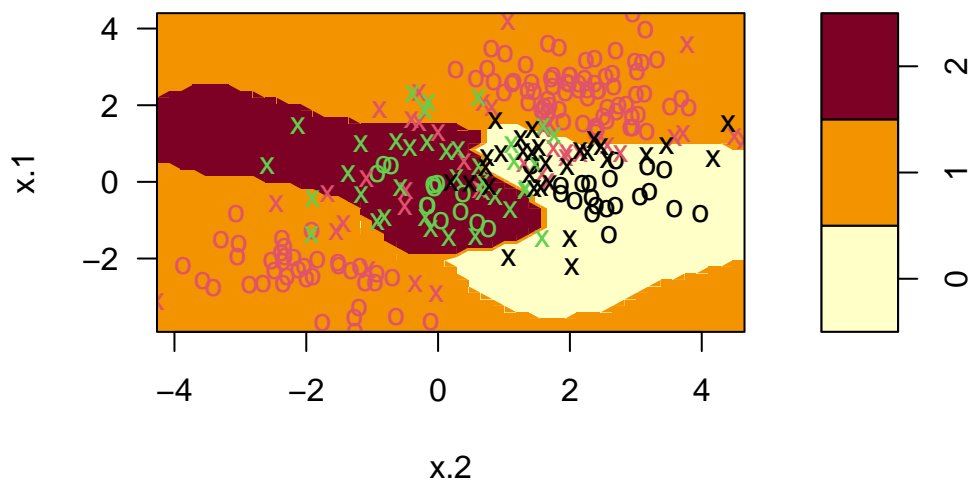
SVM with Multiple Classes

```
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))
```



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

SVM classification plot



Application to Gene Expression Data

```
library(ISLR2)
names(Khan)
```

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

```
# 20 observation ans 2308 genes
```

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

```
[1] 63 2308
```

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

```
[1] 20 2308
```

```
# We will use a support vector approach to predict cancer subtype using gene expression meas
# In this data set, there are a very large number of features relative to the number of obser
# This suggests that we should use a linear kernel,
# because the additional flexibility that will result from using a polynomial or radial kernel
```

```
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
```

```
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

```
# no training errors : large number of variables relative to the number of
#observations implies that it is easy to find hyperplanes that fully separate the classes.
```

```
# testing data
```

```
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