Untitled

```r
install.packages("e1071")
```

```
Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
(as 'lib' is unspecified)
```

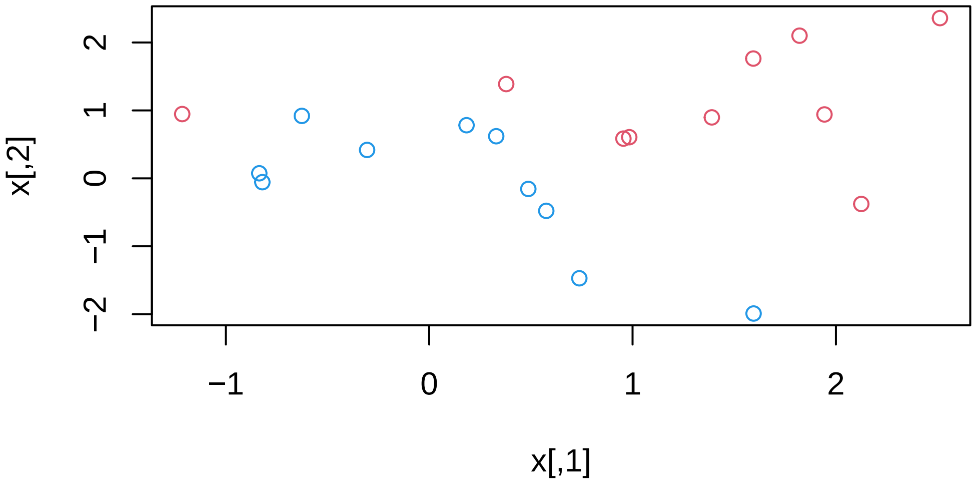
```r
library(e1071)
```

#########labs

9.6.1 Support Vector Classifer

#checking whether the classes are linearly separable

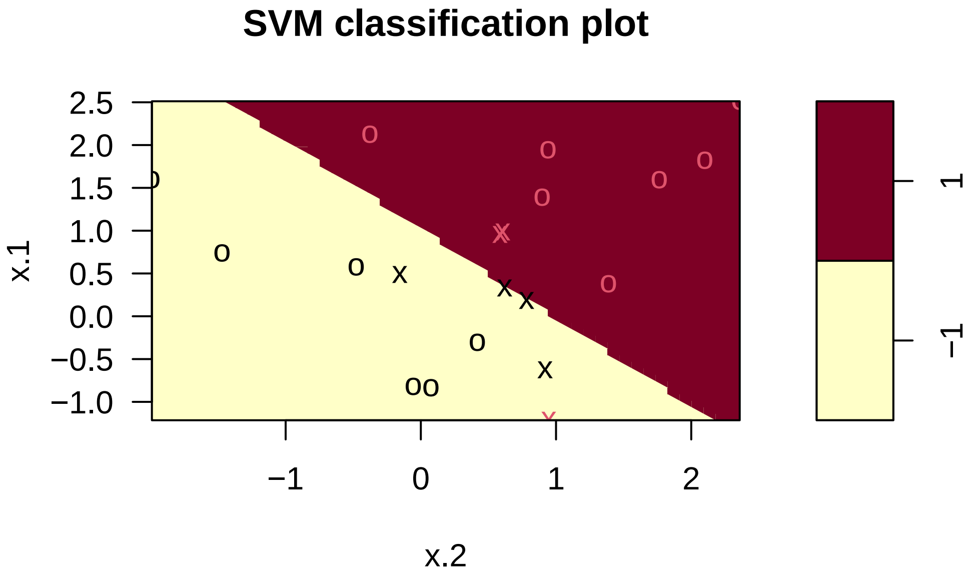
```r
set.seed(1)
x <- matrix(rnorm(20 * 2), ncol = 2)
y <- c(rep(-1, 10), rep(1, 10))
x[y == 1, ] <- x[y == 1, ] + 1
plot(x, col = (3 - y))
```



##fit the classifer

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

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



```
svmfit$index
```

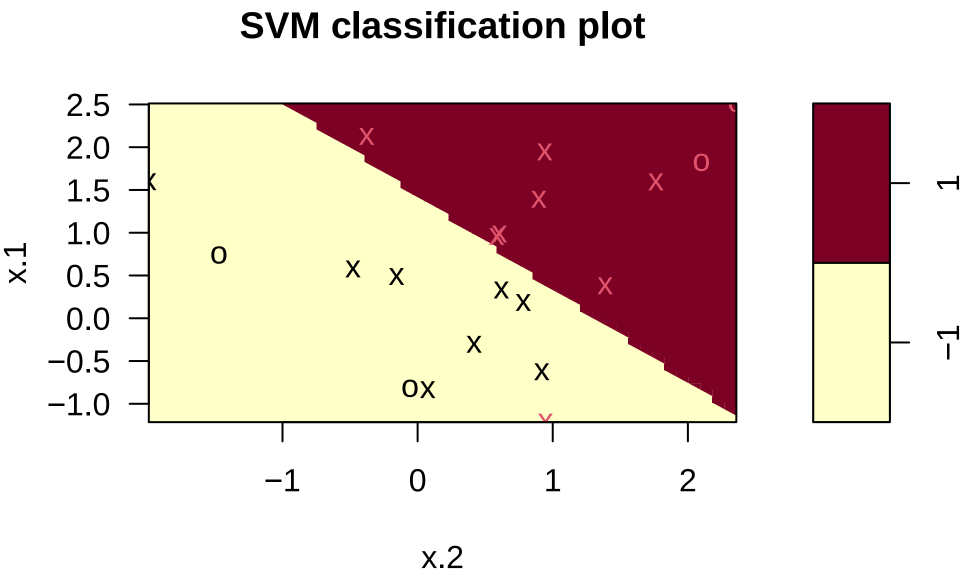
```
[1]  1  2  5  7 14 16 17
```

#there are seven support vectors

```
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
```

```
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
```

#Now that a smaller value of the cost parameter is being used, we obtain a larger number of support vectors, because the margin is now wider.

#perform cross validation

```
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
```

#cost = 0.1 results in the lowest cross-validation error rate

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

```


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
```

#generating a test data set.

```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))
```

predict the class labels of these test observations

```r
ypred <- predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
```

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

What if we had instead used cost = 0.01?

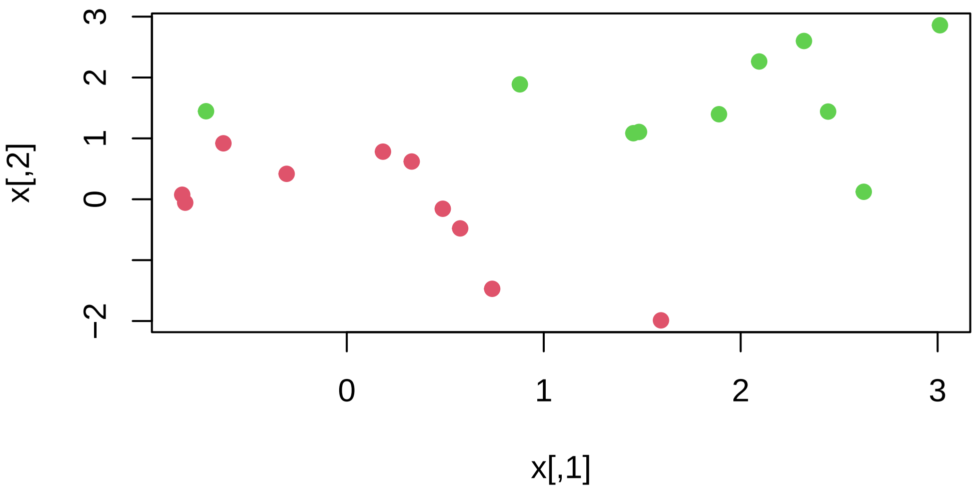
```r
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
              cost = .01, scale = FALSE)
ypred <- predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)
```

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

three additional observations are misclassifed

further separate the two classes in our simulated data so that they are linearly separable

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



```
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
```

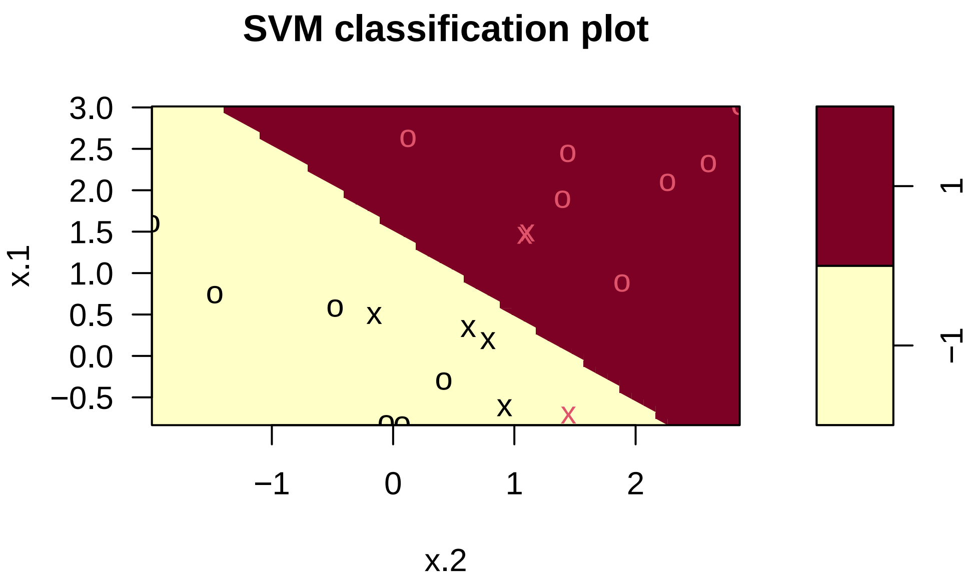
#only three support vectors were used

#It seems likely that this model will perform poorly on test data.

```
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
```

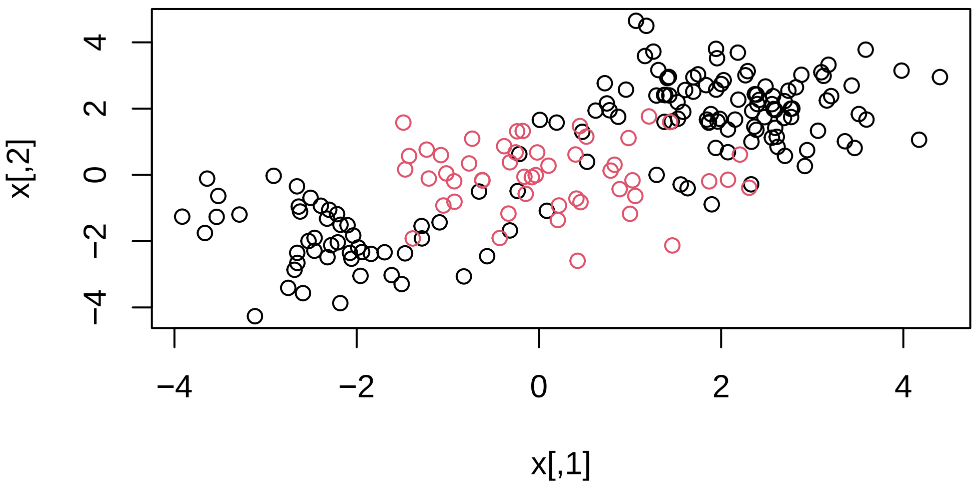
```
plot(svmfit, dat)
```



#9.6.2 Support Vector Machine

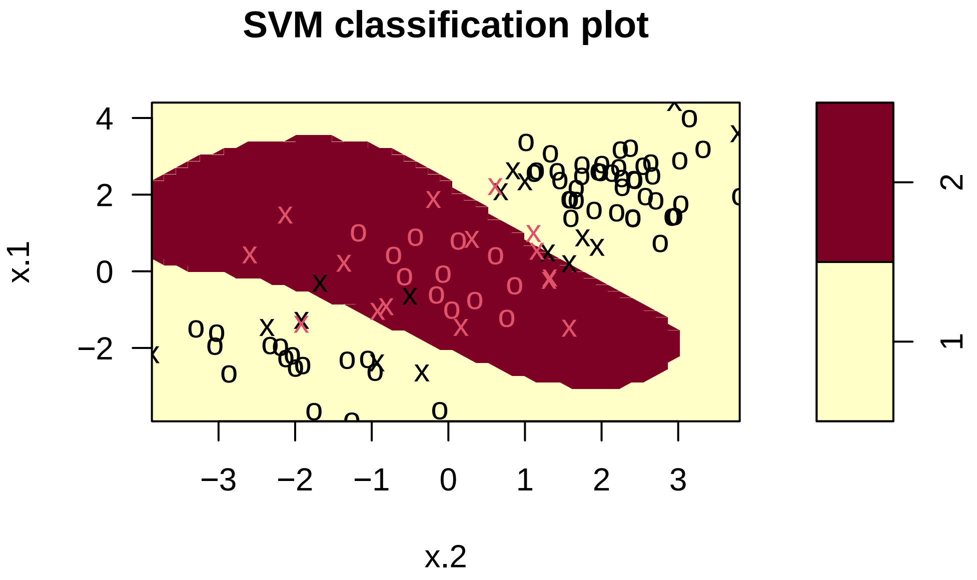
```
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
```

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



fit the training data using the svm() function with a radial kernel and = 1:

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



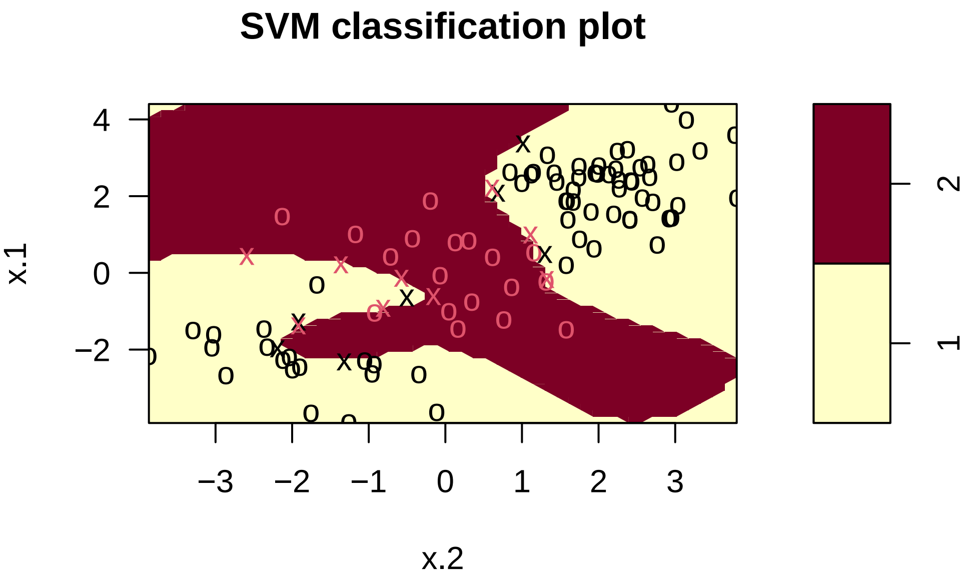
```
summary(svmfit)
```

```


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
```

seems to be at risk of overfitting the data.

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



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

```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
```

```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
```

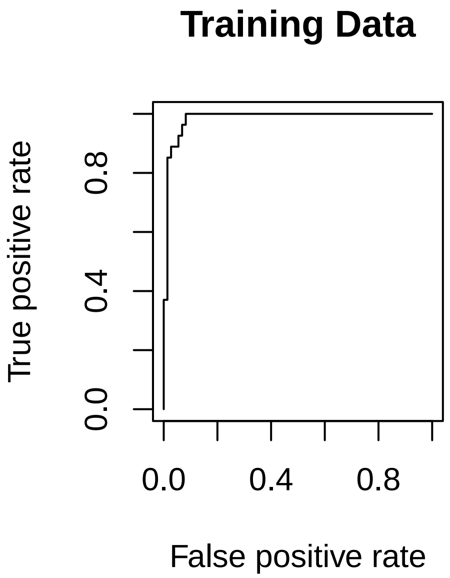
#12 % of test observations are misclassifed by this SVM

#9.6.3 ROC Curves

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

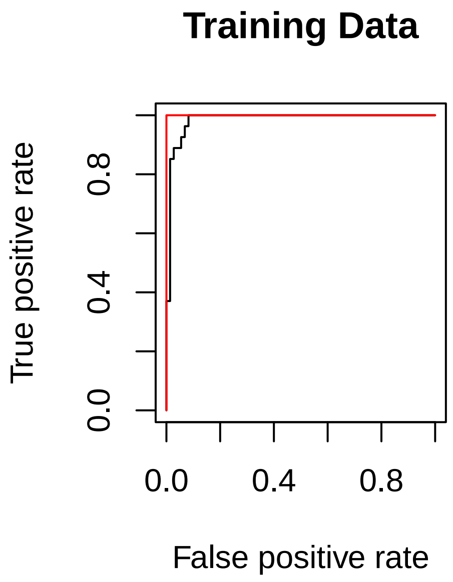
```
svmfit.opt <- svm(y ~ ., data = dat[train, ],
                  kernel = "radial", gamma = 2, cost = 1,
                  decision.values = T)
fitted <- attributes(
  predict(svmfit.opt, dat[train, ], decision.values = TRUE)
  )$decision.values
```

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



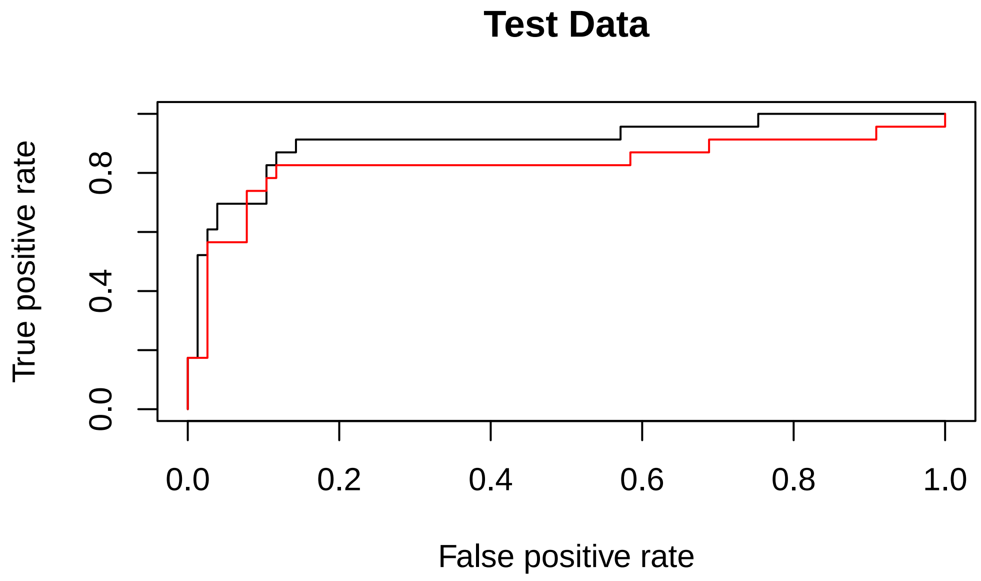
#SVM appears to be producing accurate predictions. By increasing we can produce a more flexible fit and generate further improvements in accuracy

```r
par(mfrow = c(1, 2))
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")
```



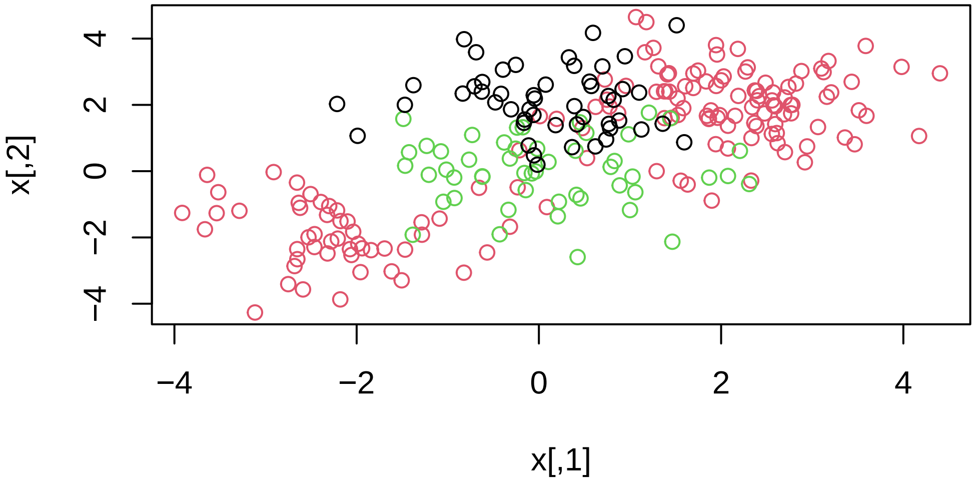
#We are really more interested in the level of prediction accuracy on the test data. #model with = 2 appears to provide the most accurate results.

```
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")
```

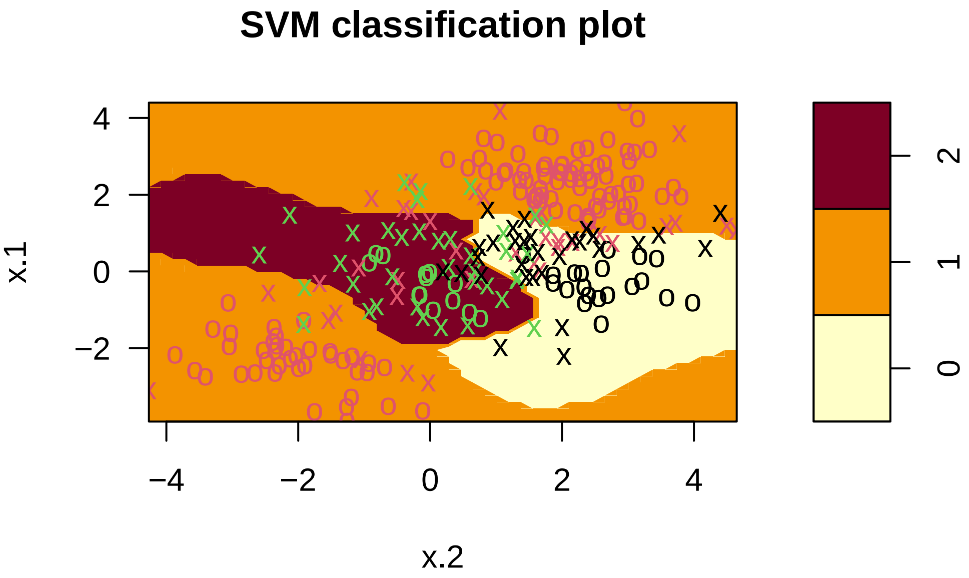


#9.6.4 SVM with Multiple Classes

```
set.seed(1)
x <- rbind(x, matrix(rnorm(50 * 2), ncol = 2)) #entries for 50 observations, each with two fe
y <- c(y, rep(0, 50)) #creates a vector of 50 zeros
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)
```



#9.6.5 Application to Gene Expression Data

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

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

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

```
[1]   63 2308
```

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

```
[1]   20 2308
```

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

```
[1] 63
```

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

```
[1] 20
```

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

```

 1  2  3  4 
 8 23 12 20 
```

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

```

1 2 3 4 
3 6 6 5 
```

```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
```

```
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
```

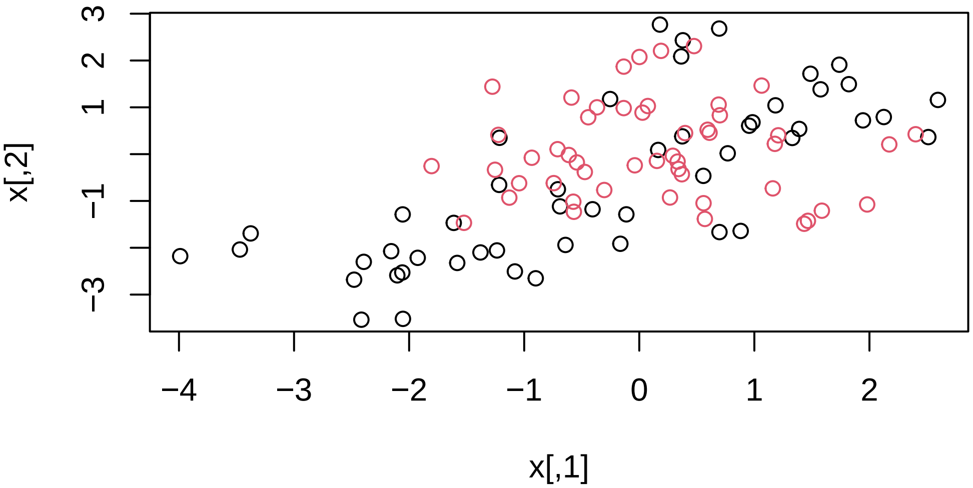
```
dat.te <- data.frame(
  x = Khan$xtest,
  y = as.factor(Khan$ytest))

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
```

#9.7 Applied Exercises #4.

```
set.seed(1)
x <- matrix(rnorm(100 * 2), ncol = 2)
x[1:20, ] <- x[1:20, ] + 1
x[20:40, ] <- x[20:40, ] - 2
y <- c(rep(1, 50), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
```

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

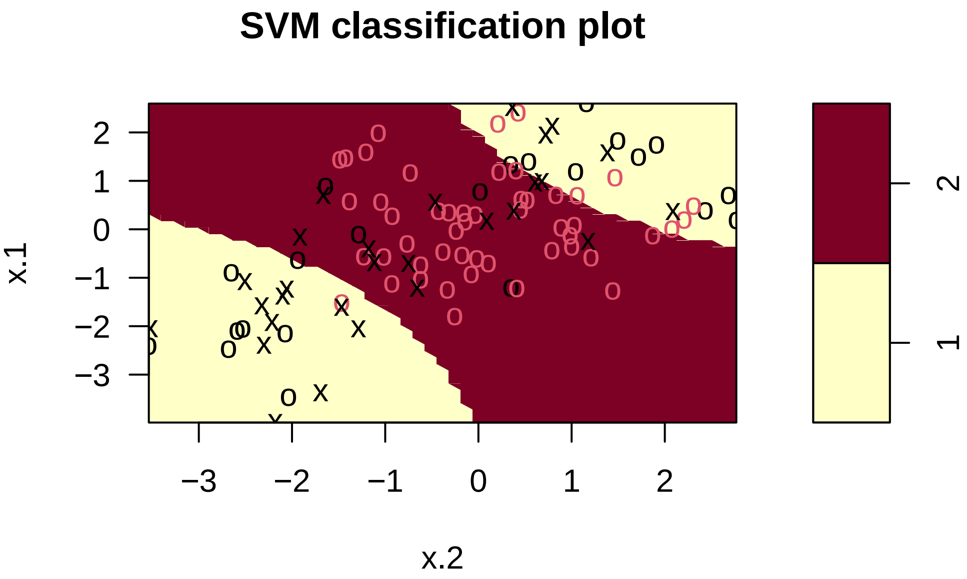


#a polynomial kernel

```
train <- sample(100, 50)
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "poly",
              degree = 2)

set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ],
                 kernel = "poly",
                 ranges = list(degree = c(1, 2, 3, 4)
) )
summary(tune.out)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 degree
      4

- best performance: 0.22

- Detailed performance results:
  degree error dispersion
1      1  0.26  0.2836273
2      2  0.24  0.2065591
3      3  0.36  0.2270585
4      4  0.22  0.1988858
```

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



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

```
    pred
true  1  2
   1 18 10
   2  4 18
```

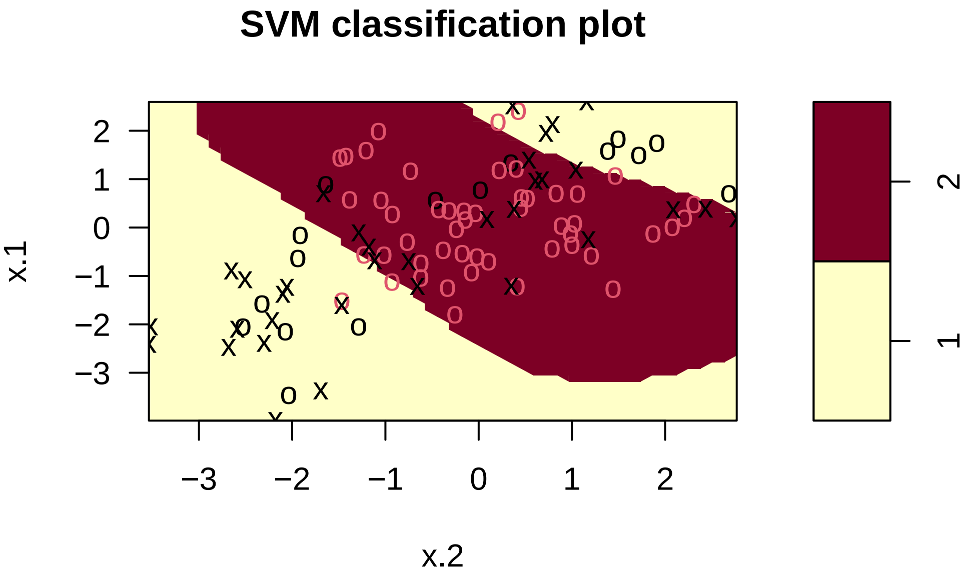
#a radial kernel

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

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: 0.24

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.44  0.2796824
2  1e+00   0.5  0.24  0.2458545
3  1e+01   0.5  0.26  0.2319004
4  1e+02   0.5  0.30  0.2538591
5  1e+03   0.5  0.36  0.2458545
6  1e-01   1.0  0.44  0.2796824
7  1e+00   1.0  0.24  0.2458545
8  1e+01   1.0  0.30  0.2538591
9  1e+02   1.0  0.32  0.2347576
10 1e+03   1.0  0.38  0.2740641
11 1e-01   2.0  0.44  0.2796824
12 1e+00   2.0  0.26  0.2319004
13 1e+01   2.0  0.32  0.2347576
14 1e+02   2.0  0.40  0.2981424
15 1e+03   2.0  0.48  0.2347576
16 1e-01   3.0  0.44  0.2796824
17 1e+00   3.0  0.30  0.2160247
18 1e+01   3.0  0.36  0.2796824
19 1e+02   3.0  0.42  0.2740641
20 1e+03   3.0  0.46  0.2836273
21 1e-01   4.0  0.44  0.2796824
22 1e+00   4.0  0.30  0.2357023
23 1e+01   4.0  0.38  0.3047768
```

```
24 1e+02   4.0  0.42  0.2898275
25 1e+03   4.0  0.46  0.2503331
```

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



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

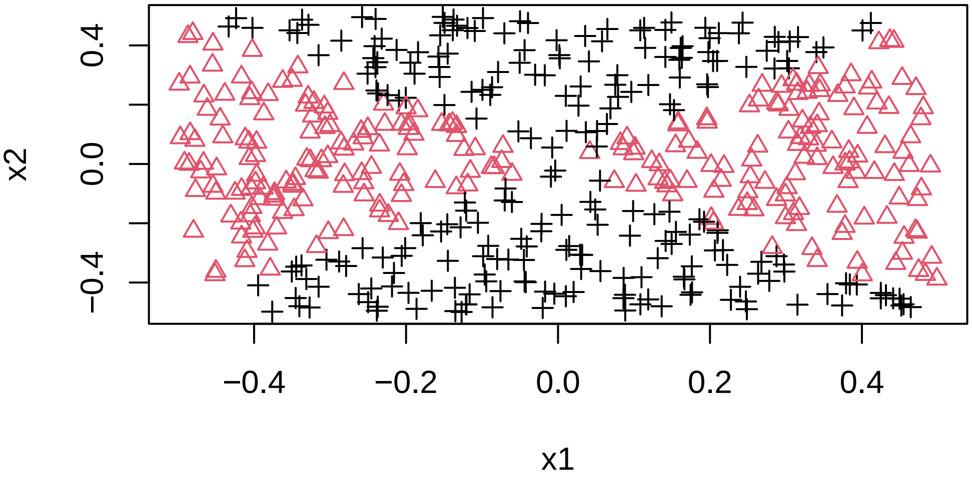
```
    pred
true  1  2
   1 17 11
   2  3 19
```

techniques performs almost the same

#5.

```
set.seed(12)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
```

```r
par(mfrow = c(1, 1))
plot(x1, x2, col =(y+1) , pch = 3-y)
```



#c) Fit a logistic regression model to the data, using X1 and X2 as predictors

```r
glm.fits <- glm(y ~ x1 + x2,family = binomial)
summary(glm.fits)
```

```
Call:
glm(formula = y ~ x1 + x2, family = binomial)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.04927    0.08978   0.549    0.583
x1          -0.23002    0.31534  -0.729    0.466
x2           0.51072    0.31560   1.618    0.106

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 692.86  on 499  degrees of freedom
Residual deviance: 689.58  on 497  degrees of freedom
AIC: 695.58

Number of Fisher Scoring iterations: 3
```

#d) Apply this model to the training data

```
glm.probs <- predict(glm.fits, type = "response")
glm.pred <- rep(0, 500)
glm.pred[glm.probs > .5] = 1
table(glm.pred, y)
```

```
        y
glm.pred   0   1
       0 125  73
       1 119 183
```

#e) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors

```
glm.fits1 <- glm(y ~ x1 + I(x1^2) + x2, family = binomial)
summary(glm.fits1)
```

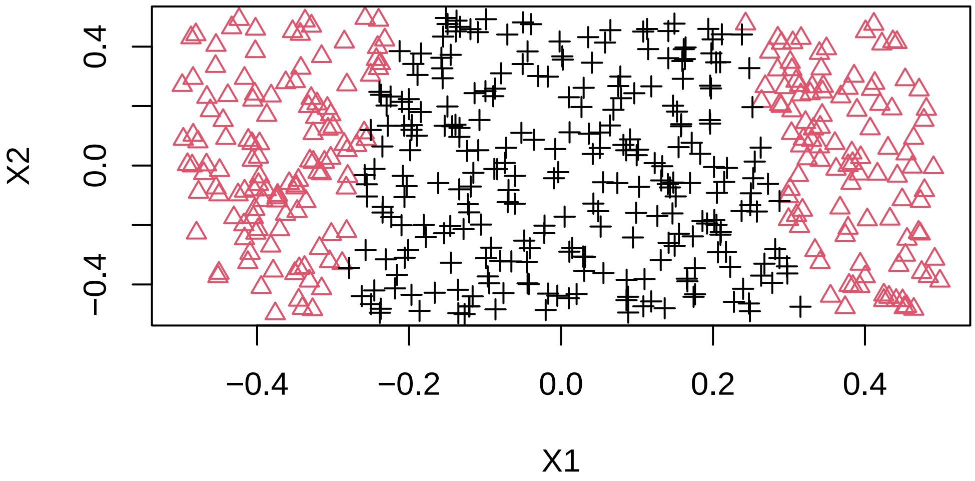
```

Call:
glm(formula = y ~ x1 + I(x1^2) + x2, family = binomial)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept)  -1.3955     0.1661  -8.404   <2e-16 ***
x1           -0.4005     0.4036  -0.992   0.3211    
I(x1^2)      19.1102     1.8827  10.151   <2e-16 ***
x2            0.8015     0.3696   2.169   0.0301 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 692.86  on 499  degrees of freedom
Residual deviance: 539.84  on 496  degrees of freedom
AIC: 547.84

Number of Fisher Scoring iterations: 4
```

#f) Apply this model to the training data in order to obtain a predicted class label for each training observation.

```
glm.probs1 <- predict(glm.fits1, type = "response")
glm.pred1 <- rep(0, 500)
glm.pred1[glm.probs1 > .5] = 1
table(glm.pred1, y)
```

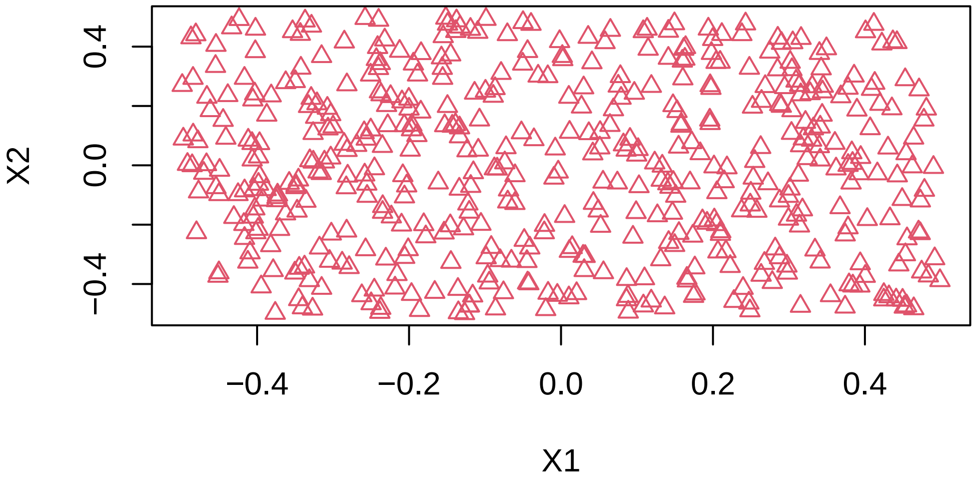
```
         y
glm.pred1   0   1
        0 189  78
        1  55 178
```

```
data <- data.frame(x1 = x1, x2 = x2, y = y)
plot(data[glm.pred1 == 1, ]$x1, data[glm.pred1 == 1, ]$x2, col = (1 + 1), pch = (3 - 1), xlab
points(data[glm.pred1 == 0, ]$x1, data[glm.pred1 == 0, ]$x2, col = (1 + 0), pch = (3 - 0))
```



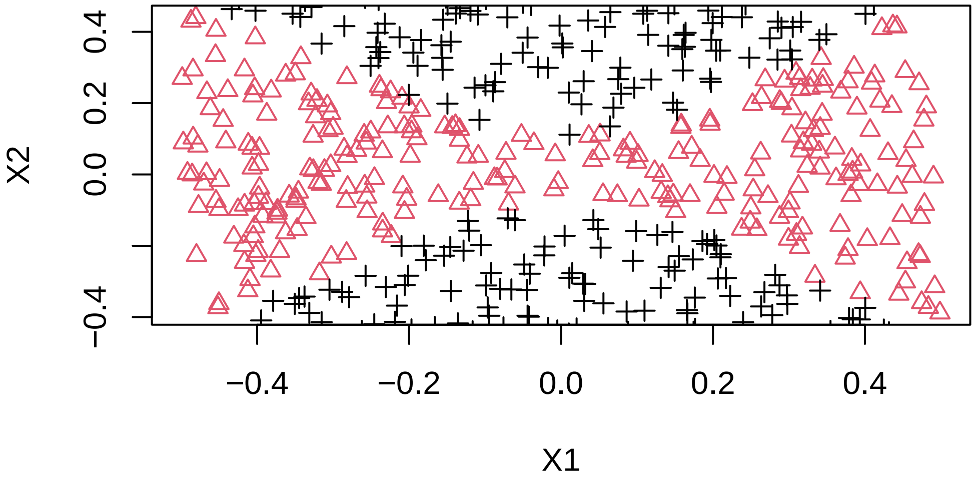
#g) Fit a support vector classifer to the data with X1 and X2 as predictors.

```
data$y <- as.factor(data$y)
svmfit <- svm(y ~ x1 + x2, data = data, kernel = "linear",
              cost = 0.01, scale = FALSE)
preds <- predict(svmfit, data)
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (1 + 1), pch = (3 - 1), xlab = "X1"
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (1 + 0), pch = (3 - 0))
```



#h) Fit a SVM using a non-linear kernel to the data

```
svmfit <- svm(y ~ x1 + x2, data = data, kernel = "radial",
              gamma = 1, cost = 1)
preds <- predict(svmfit, data)
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (1 + 1), pch = (3 - 1), xlab = "X1"
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (1 + 0), pch = (3 - 0))
```

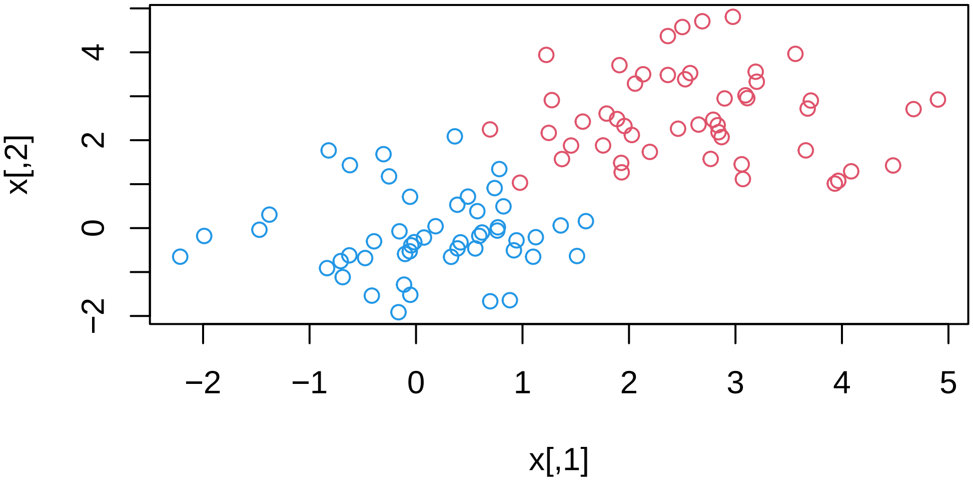


#i) Comment on your results. #fitting a SVM seems best

#6.

#a) Generate two-class data with p = 2

```
set.seed(1)
x <- matrix(rnorm(100 * 2), ncol = 2)
y <- c(rep(-1, 50), rep(1, 50))
x[y == 1, ] <- x[y == 1, ] + 2.5
plot(x, col = (3 - y))
```



#b) Compute the cross-validation error rates for support vector classifers with a range of cost values

```
dat <- data.frame(x = x, y = y)
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: 0.2168627
- Detailed performance results:
   cost     error dispersion
1 1e-03 1.2669256 0.45202618
2 1e-02 0.2492452 0.05693377
3 1e-01 0.2168627 0.04430729
4 1e+00 0.2210400 0.04191765
5 5e+00 0.2209562 0.04155216
6 1e+01 0.2208784 0.04159409
7 1e+02 0.2208707 0.04156451
```

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

```


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:  eps-regression
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  87
```

#Number of Support Vectors: 87

```r
preds <- predict(bestmod, dat)
pred1 <- rep(0, 100)
pred1[preds > .5] = 1
table(pred1, y)
```

```
     y
pred1 -1  1
    0 50 12
    1  0 38
```

#c) Generate an appropriate test data set

```r
set.seed(12)
xtest <- matrix(rnorm(50 * 2), ncol = 2)
ytest <- c(rep(-1, 25), rep(1, 25))
xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 2.5
testdat <- data.frame(x = xtest, y = as.factor(ytest))
```

```r
costs <- c(0.01, 0.1, 1, 5, 10)
test.err <- rep(NA, length(costs))
data.test <- data.frame(x = xtest, z = as.factor(ytest))
for (i in 1:length(costs)) {
    svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = costs[i])
    pred <- predict(svmfit, testdat)
    pred1 <- rep(0, 50)
    pred1[pred > .5] = 1
    test.err[i] <- sum(pred1 != testdat$y)
}
data.frame(cost = costs, misclass = test.err)
```

```
   cost misclass
1  0.01       35
2  0.10       31
3  1.00       31
4  5.00       31
5 10.00       31
```

#0.1 #7.

```r
Auto <- na.omit(Auto)
Auto$mpg_b <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpg_b <-  as.factor(Auto$mpg_b )
```

```r
tune.out <- tune(svm, mpg_b ~ .-mpg, data = Auto, 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
 0.01

- best performance: 0.08935897

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.13025641 0.03349293
2 1e-02 0.08935897 0.03478144
3 1e-01 0.09185897 0.03843785
4 1e+00 0.09961538 0.02845044
5 5e+00 0.11750000 0.03895464
6 1e+01 0.11500000 0.04437783
7 1e+02 0.12519231 0.02882961
```

#cost = 1

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

```
Call:
best.tune(METHOD = svm, train.x = mpg_b ~ . - mpg, data = Auto, 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:  0.01

Number of Support Vectors:  168

 ( 84 84 )


Number of Classes:  2

Levels:
 0 1
```

```
ypred <- predict(bestmod, Auto)

table(predict = ypred, truth = Auto$mpg_b)
```

```
       truth
predict   0   1
      0 170   8
      1  26 188
```

#c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels,

```
tune.out <- tune(svm, mpg_b ~ .-mpg, data = Auto, kernel = "radial",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
                               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     1

- best performance: 0.08147436

- Detailed performance results:
    cost gamma      error dispersion
1  1e-03   0.5 0.56358974 0.05130146
2  1e-02   0.5 0.56358974 0.05130146
3  1e-01   0.5 0.08923077 0.04214637
4  1e+00   0.5 0.08410256 0.03993932
5  5e+00   0.5 0.08653846 0.03367809
6  1e+01   0.5 0.08910256 0.03384038
7  1e+02   0.5 0.09166667 0.02914298
8  1e-03   1.0 0.56358974 0.05130146
9  1e-02   1.0 0.56358974 0.05130146
10 1e-01   1.0 0.56358974 0.05130146
```

```
11 1e+00   1.0 0.08147436 0.03917716
12 5e+00   1.0 0.08910256 0.03979296
13 1e+01   1.0 0.08910256 0.03979296
14 1e+02   1.0 0.09166667 0.04154431
15 1e-03   2.0 0.56358974 0.05130146
16 1e-02   2.0 0.56358974 0.05130146
17 1e-01   2.0 0.56358974 0.05130146
18 1e+00   2.0 0.12987179 0.07213189
19 5e+00   2.0 0.13243590 0.06943307
20 1e+01   2.0 0.13243590 0.06943307
21 1e+02   2.0 0.13243590 0.06943307
22 1e-03   3.0 0.56358974 0.05130146
23 1e-02   3.0 0.56358974 0.05130146
24 1e-01   3.0 0.56358974 0.05130146
25 1e+00   3.0 0.36455128 0.18145896
26 5e+00   3.0 0.36711538 0.17204930
27 1e+01   3.0 0.36711538 0.17204930
28 1e+02   3.0 0.36711538 0.17204930
29 1e-03   4.0 0.56358974 0.05130146
30 1e-02   4.0 0.56358974 0.05130146
31 1e-01   4.0 0.56358974 0.05130146
32 1e+00   4.0 0.47423077 0.08585905
33 5e+00   4.0 0.46160256 0.07545274
34 1e+01   4.0 0.46160256 0.07545274
35 1e+02   4.0 0.46160256 0.07545274
```

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

```

Call:
best.tune(METHOD = svm, train.x = mpg_b ~ . - mpg, data = Auto, ranges = list(cost = c(0.001
    0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  376
```

```
 ( 187 189 )


Number of Classes:  2

Levels: 
 0 1
```

```
ypred <- predict(bestmod, Auto)

table(predict = ypred, truth = Auto$mpg_b)
```

```
       truth
predict   0   1
      0 195   2
      1   1 194
```

```
tune.out <- tune(svm, mpg_b ~ .-mpg, data = Auto, kernel = "poly",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
                               degree = c(1, 2, 3, 4)))
summary(tune.out)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation 

- best parameters:
 cost degree
    5      1

- best performance: 0.08935897 

- Detailed performance results:
    cost degree      error dispersion
1  1e-03      1 0.58634615 0.04634541
2  1e-02      1 0.58634615 0.04634541
3  1e-01      1 0.26455128 0.13497715
4  1e+00      1 0.10987179 0.03254492
5  5e+00      1 0.08935897 0.03884801
6  1e+01      1 0.09192308 0.04407224
```

```
7  1e+02      1 0.09192308 0.03264577
8  1e-03      2 0.58634615 0.04634541
9  1e-02      2 0.58634615 0.04634541
10 1e-01      2 0.58634615 0.04634541
11 1e+00      2 0.58634615 0.04634541
12 5e+00      2 0.58634615 0.04634541
13 1e+01      2 0.57352564 0.05444489
14 1e+02      2 0.31134615 0.07418073
15 1e-03      3 0.58634615 0.04634541
16 1e-02      3 0.58634615 0.04634541
17 1e-01      3 0.58634615 0.04634541
18 1e+00      3 0.58634615 0.04634541
19 5e+00      3 0.58634615 0.04634541
20 1e+01      3 0.58634615 0.04634541
21 1e+02      3 0.40237179 0.12105339
22 1e-03      4 0.58634615 0.04634541
23 1e-02      4 0.58634615 0.04634541
24 1e-01      4 0.58634615 0.04634541
25 1e+00      4 0.58634615 0.04634541
26 5e+00      4 0.58634615 0.04634541
27 1e+01      4 0.58634615 0.04634541
28 1e+02      4 0.58634615 0.04634541
```

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

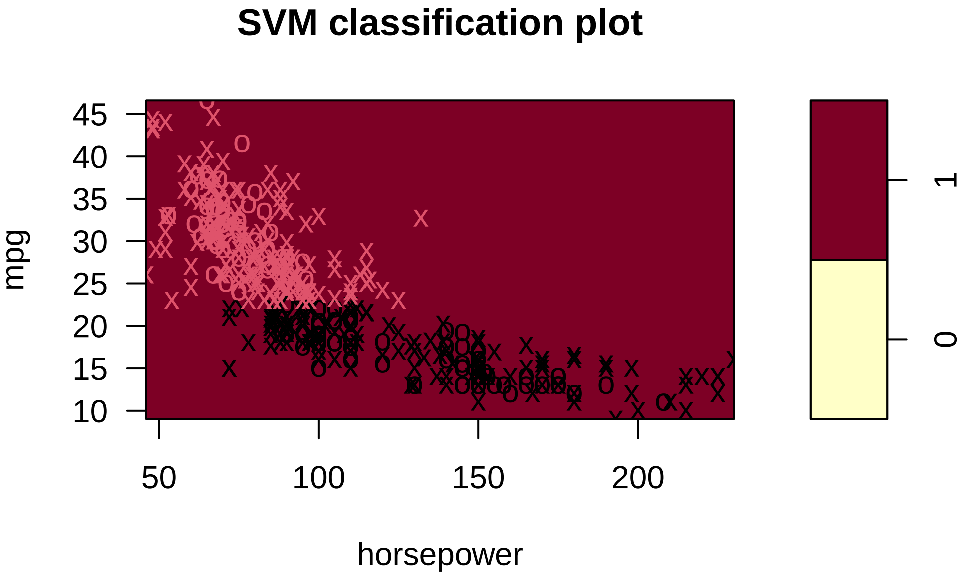
```
Call:
best.tune(METHOD = svm, train.x = mpg_b ~ . - mpg, data = Auto, ranges = list(cost = c(0.001
    0.01, 0.1, 1, 5, 10, 100), degree = c(1, 2, 3, 4)), kernel = "poly")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  5
     degree:  1
     coef.0:  0

Number of Support Vectors:  148

 ( 73 75 )
```

```
Number of Classes:  2

Levels:
 0 1
```

```
ypred <- predict(bestmod, Auto)

table(predict = ypred, truth = Auto$mpg_b)
```

```
       truth
predict   0   1
      0 170   9
      1  26 187
```

```
svm.radial <- svm(mpg_b ~ cylinders + displacement + horsepower + weight + acceleration + yea
                  kernel = "radial", cost = 1, gamma = 0.5)
plot(svm.radial, Auto, mpg ~ horsepower)
```



#8. #a) Create a training set containing a random sample of 800 observations

```
set.seed(1)
indexes <- sample(1:nrow(OJ), 800)
train <- OJ[indexes, ]
test <- OJ[-indexes, ]
```

```r
svmfit <- svm(Purchase ~ ., data = train, kernel = "linear",
              cost = 0.01, scale = FALSE)

summary(svmfit)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  615

 ( 309 306 )


Number of Classes:  2

Levels:
 CH MM
```

```r
ypred <- predict(svmfit, train)
table(predict = ypred, truth = train$Purchase)
```

```
       truth
predict  CH  MM
     CH 420 105
     MM  65 210
```

#(65+105)/800 =0.2125

```r
ypred <- predict(svmfit, test)
table(predict = ypred, truth = test$Purchase)
```

```
       truth
predict  CH  MM
     CH 148  43
     MM  20  59
```

#(20+42)/270 = 0.22962963

#d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```r
set.seed(1)
tune.out <- tune(svm, Purchase ~ ., data = train, 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: 0.1725

- Detailed performance results:
   cost   error dispersion
1 1e-03 0.31250 0.04124790
2 1e-02 0.17625 0.02853482
3 1e-01 0.17250 0.03162278
4 1e+00 0.17500 0.02946278
5 5e+00 0.17250 0.03162278
6 1e+01 0.17375 0.03197764
7 1e+02 0.17500 0.03486083
```

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

```

Call:
```

```
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, 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:  342

 ( 171 171 )


Number of Classes:  2

Levels:
 CH MM
```

```
ypred <- predict(bestmod, train)
table(predict = ypred, truth = train$Purchase)
```

```
       truth
predict  CH  MM
     CH 422  69
     MM  63 246
```

#(69+63)/800 = 0.165

```
ypred <- predict(bestmod, test)
table(predict = ypred, truth = test$Purchase)
```

```
       truth
predict  CH  MM
     CH 155  31
     MM  13  71
```

#(13+31)/270 = 0.162962963

#f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma

```r
set.seed(1)
tune.out <- tune(svm, Purchase ~ ., data = 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
   10   0.5

- best performance: 0.2125

- Detailed performance results:
    cost gamma   error dispersion
1  1e-01   0.5 0.28250 0.05502525
2  1e+00   0.5 0.21375 0.03701070
3  1e+01   0.5 0.21250 0.03632416
4  1e+02   0.5 0.23875 0.04016027
5  1e+03   0.5 0.23875 0.06248611
6  1e-01   1.0 0.34500 0.04937104
7  1e+00   1.0 0.22625 0.04466309
8  1e+01   1.0 0.23000 0.04684490
9  1e+02   1.0 0.24375 0.04973890
10 1e+03   1.0 0.24250 0.05658082
11 1e-01   2.0 0.38625 0.04348132
12 1e+00   2.0 0.22750 0.04281744
13 1e+01   2.0 0.24000 0.04158325
14 1e+02   2.0 0.25875 0.05205833
15 1e+03   2.0 0.26375 0.04910660
16 1e-01   3.0 0.39375 0.04007372
17 1e+00   3.0 0.22625 0.03304563
18 1e+01   3.0 0.25375 0.03335936
19 1e+02   3.0 0.26125 0.03793727
20 1e+03   3.0 0.26375 0.03557562
21 1e-01   4.0 0.39375 0.04007372
```

```
22 1e+00   4.0 0.22750 0.03322900
23 1e+01   4.0 0.25500 0.03496029
24 1e+02   4.0 0.26250 0.03280837
25 1e+03   4.0 0.26750 0.03073181
```

```
table(
  true = train$Purchase,
  pred = predict(
    tune.out$best.model, newdata = train
) )
```

```
    pred
true   CH  MM
  CH 449  36
  MM  49 266
```

#(49+36)/800 = 0.10625

```
table(
  true = test$Purchase,
  pred = predict(
    tune.out$best.model, newdata = test
) )
```

```
    pred
true   CH  MM
  CH 152  16
  MM  37  65
```

#(37+16)/270 =0.196296296

#g)

```
set.seed(1)
tune.out <- tune(svm, Purchase ~ ., data = train,
                 kernel = "poly",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                               degree = c(0.5, 1, 2, 3, 4)
                               ) )
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      1

- best performance: 0.17125

- Detailed performance results:
    cost degree   error dispersion
1  1e-01    0.5 0.39375 0.04007372
2  1e+00    0.5 0.39375 0.04007372
3  1e+01    0.5 0.39375 0.04007372
4  1e+02    0.5 0.39375 0.04007372
5  1e+03    0.5 0.39375 0.04007372
6  1e-01    1.0 0.18000 0.02776389
7  1e+00    1.0 0.17625 0.02853482
8  1e+01    1.0 0.17125 0.02703521
9  1e+02    1.0 0.17375 0.03304563
10 1e+03    1.0 0.17500 0.03486083
11 1e-01    2.0 0.32125 0.05001736
12 1e+00    2.0 0.20250 0.04116363
13 1e+01    2.0 0.18125 0.02779513
14 1e+02    2.0 0.18250 0.02513851
15 1e+03    2.0 0.19125 0.02503470
16 1e-01    3.0 0.28750 0.05068969
17 1e+00    3.0 0.18500 0.02415229
18 1e+01    3.0 0.19500 0.03184162
19 1e+02    3.0 0.22000 0.04609772
20 1e+03    3.0 0.23625 0.04656611
21 1e-01    4.0 0.31875 0.04903584
22 1e+00    4.0 0.23000 0.03016160
23 1e+01    4.0 0.20375 0.02949223
24 1e+02    4.0 0.21375 0.02729087
25 1e+03    4.0 0.23000 0.03016160
```

```r
table(
  true = train$Purchase,
  pred = predict(
    tune.out$best.model, newdata = train
) )
```

```
    pred
true   CH  MM
  CH 424  61
  MM  71 244
```

#(71+61)/800=0.165

```
table(
  true = test$Purchase,
  pred = predict(
    tune.out$best.model, newdata = test
) )
```

```
    pred
true   CH  MM
  CH 155  13
  MM  29  73
```

#(29+13)/270 = 0.155555556