Chapter 9

R Quiz

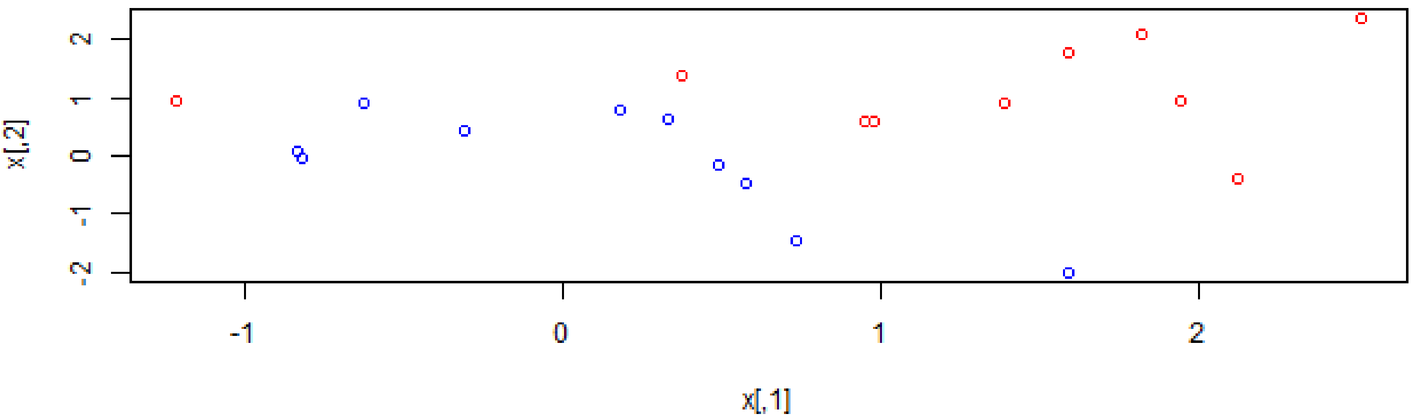
```
set.seed(1001)
counts = 100
errate = rep(0, counts)
for(i in 1:counts){
  x = matrix(rnorm(100 * 10), ncol = 10)
  y = c(rep(0, 50), rep(1, 50))
  x[y == 1, 1:5] = x[y == 1, 1:5] + 1
  dat = data.frame(x = x, y = as.factor(y))
  svm.fit = svm(y ~ ., data = dat, kernel = "linear", cost = 1)


  xtest = matrix(rnorm(100 * 10), ncol = 10)
  ytest = sample(c(0, 1), 100, rep = TRUE)
  xtest[ytest == 1,] = xtest[ytest == 1,] + 1
  testdat = data.frame(x = xtest, y = as.factor(ytest))

  ypred = predict(svm.fit, testdat)
  result = table(predict = ypred, truth = testdat$y)
  errate[i] = 1 - (result[1] + result[4]) / 100
}
mean(errate)
```

```
[1] 0.1673
```

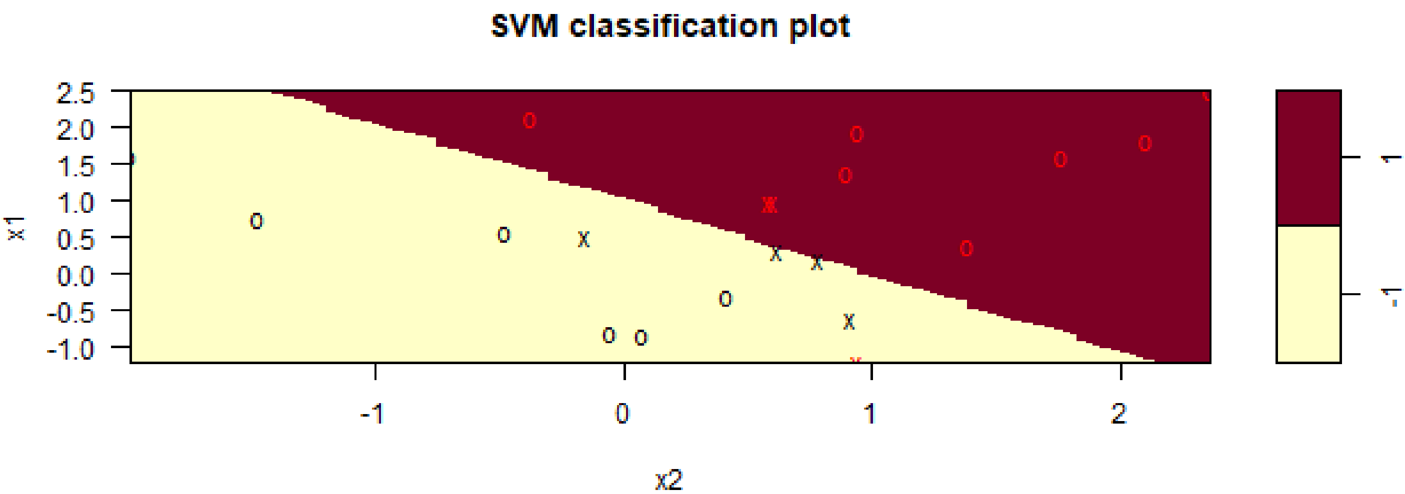
Lab

Support Vector Classifier

```
set.seed(1)

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




```
dat <- data.table(x1 = x[,1], x2 = x[, 2], y = as.factor(y))
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = F)
plot(svmfit, dat)
```




```
svmfit$index
```

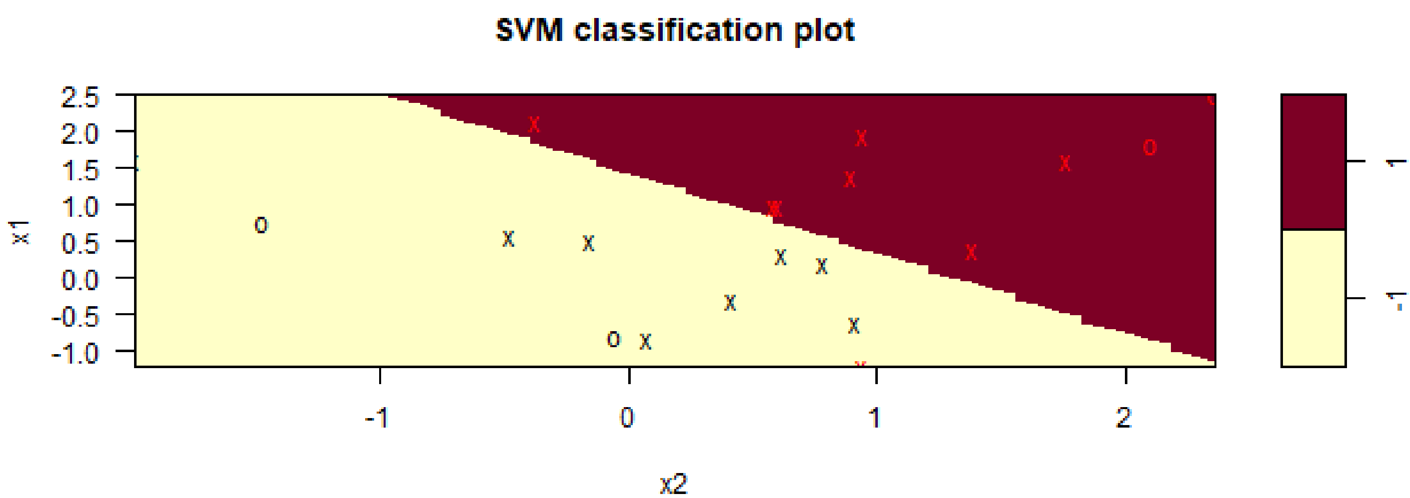
```
[1]  1  2  5  7 14 16 17
```

```
summary(svmfit)
```

```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = F)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

```r
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 0.1, scale = F)
plot(svmfit, dat)
```




```r
svmfit$index
```

```
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

```r
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
```

```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:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

```r
xtest <- matrix(rnorm(20*2), ncol = 2)
ytest <- sample(c(-1, 1), 20, rep = T)
xtest[ytest==1,] = xtest[ytest==1,] + 1
testdata <- data.table(x1 = xtest[, 1], x2 = xtest[, 2], y = as.factor(ytest))
```

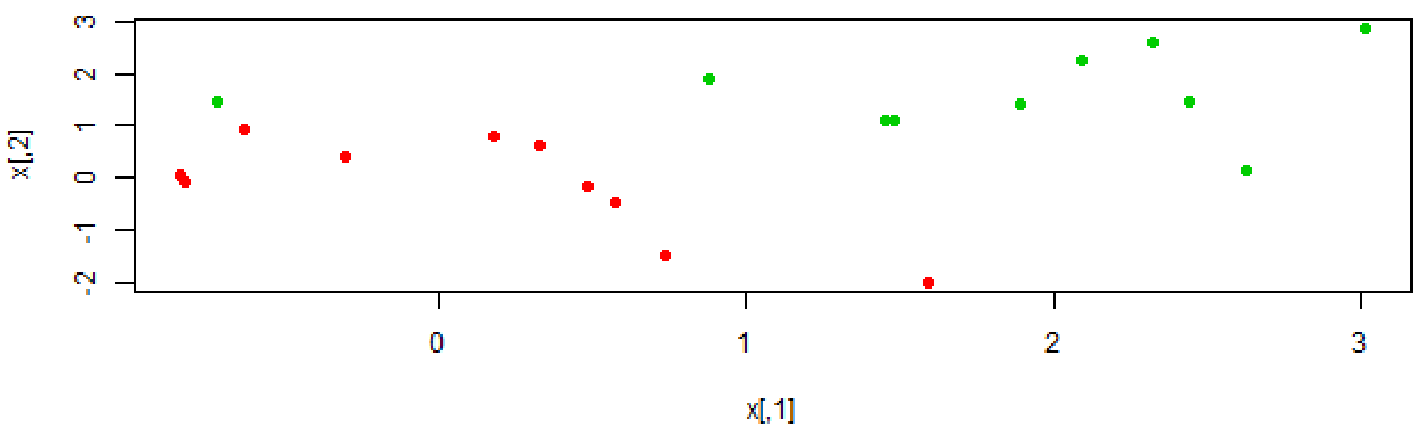
```r
ypred <- predict(bestmod, testdata)
table(predict = ypred, truth = testdata$y)
```

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

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

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

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

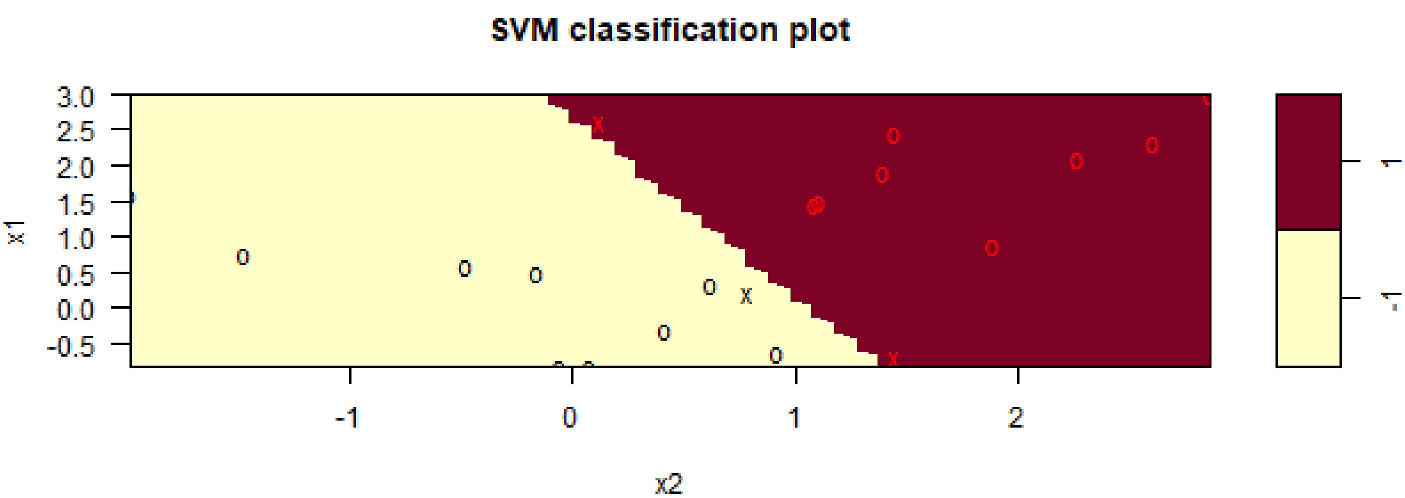



```r
dat <- data.table(x = x, y = as.factor(y))
names(dat)[1:2] <- c("x1", "x2")

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
```

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

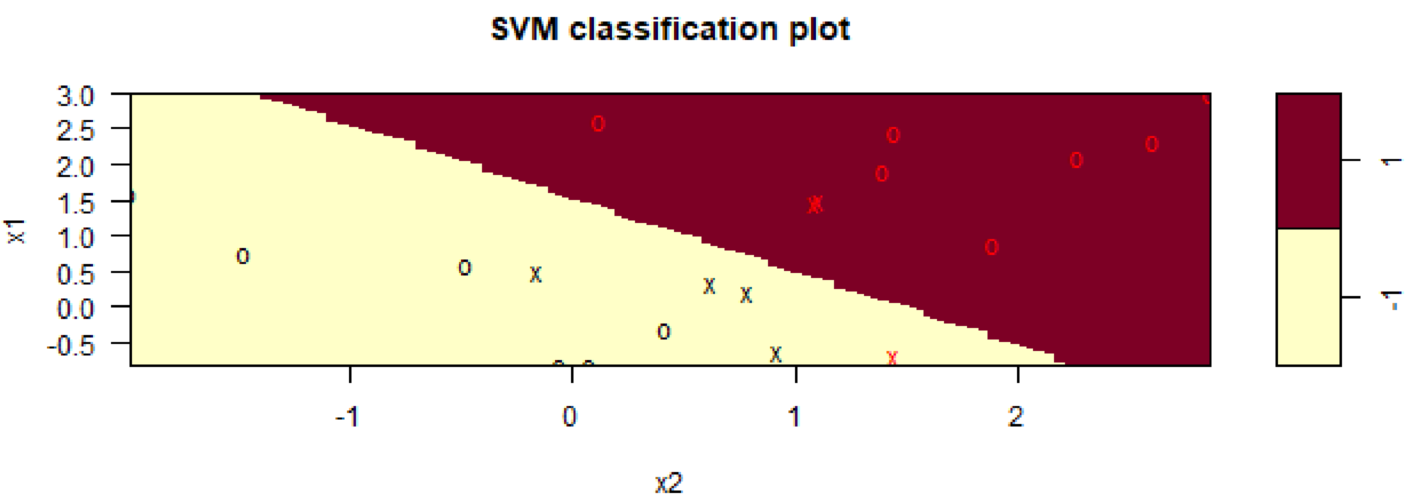



```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 1)
summary(svmfit)
```

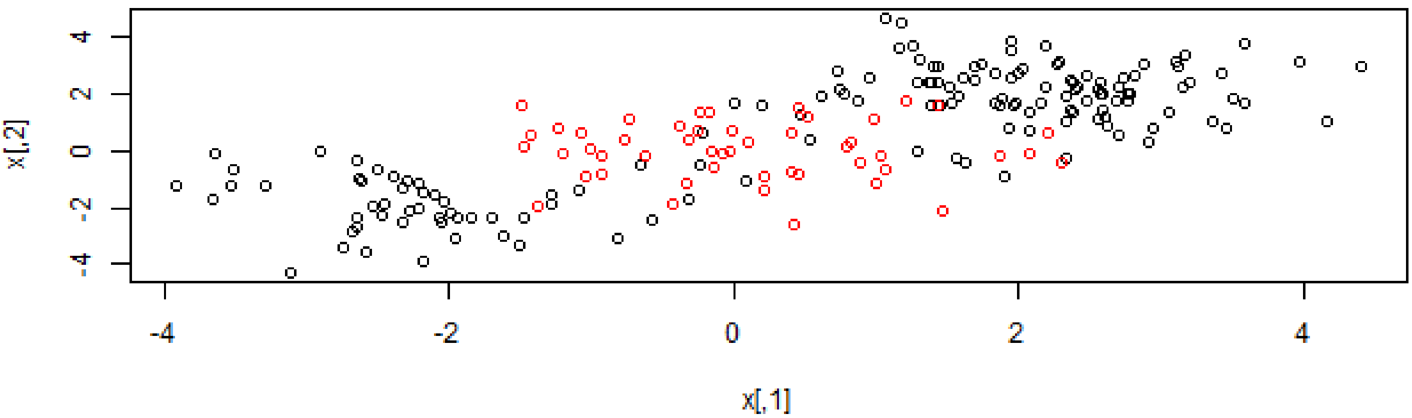
```
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
```

```
Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

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

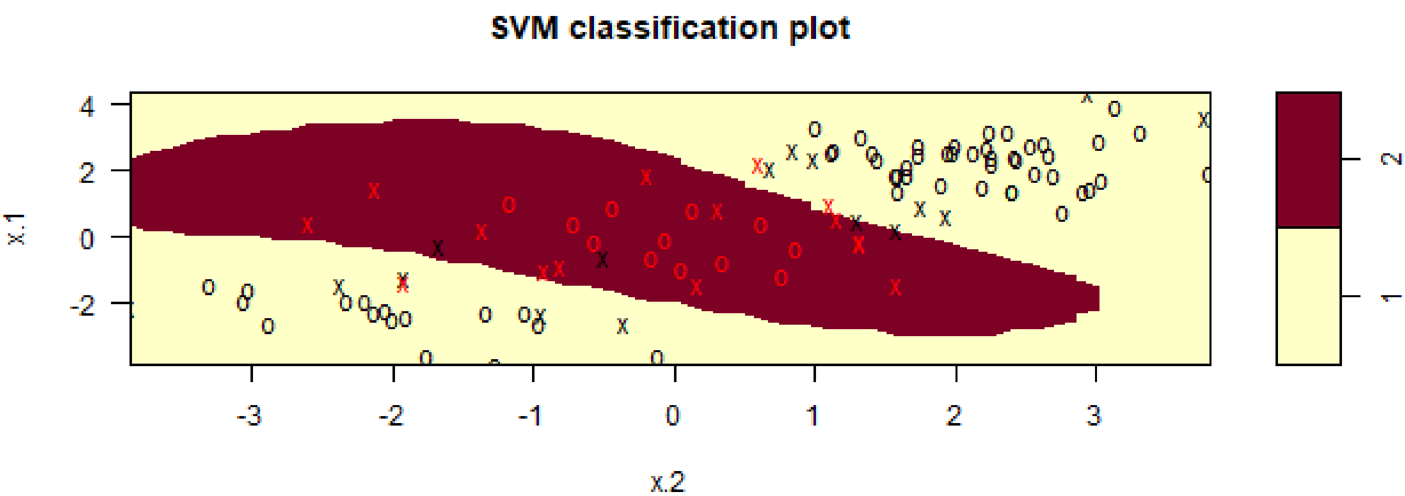



```r
set.seed(1)

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(x = x, y = as.factor(y))

par(mfrow = c(1, 1))
plot(x, col = y)
```



Support Vecctor Machines

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

plot(svmfit, dat[train,])
```

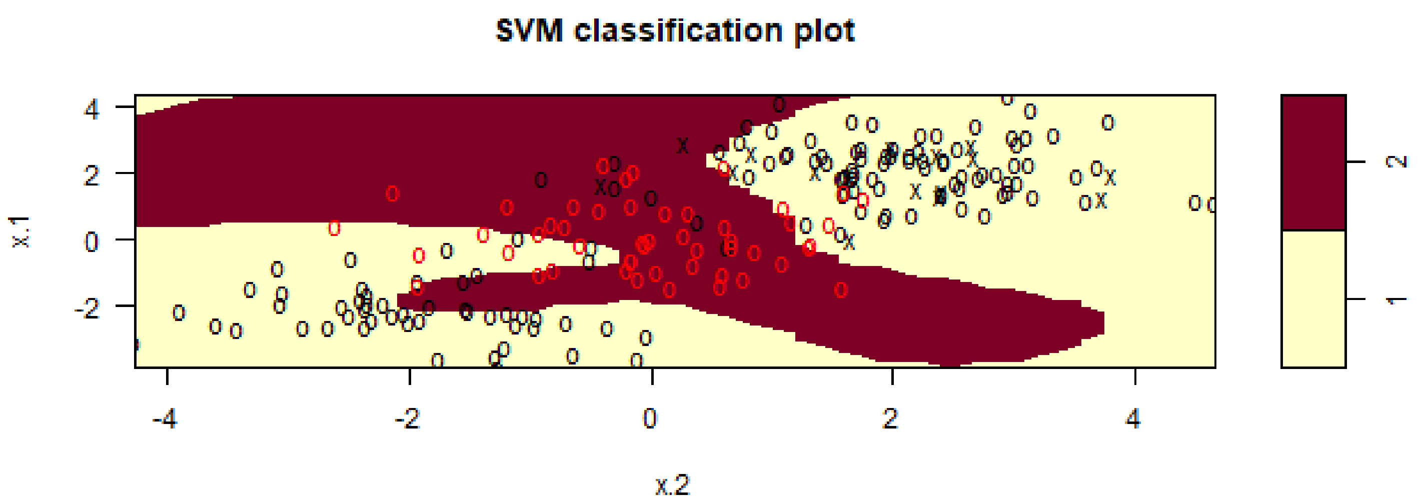



```
summary(svmfit)
```

```
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
```

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




```
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,]))
```

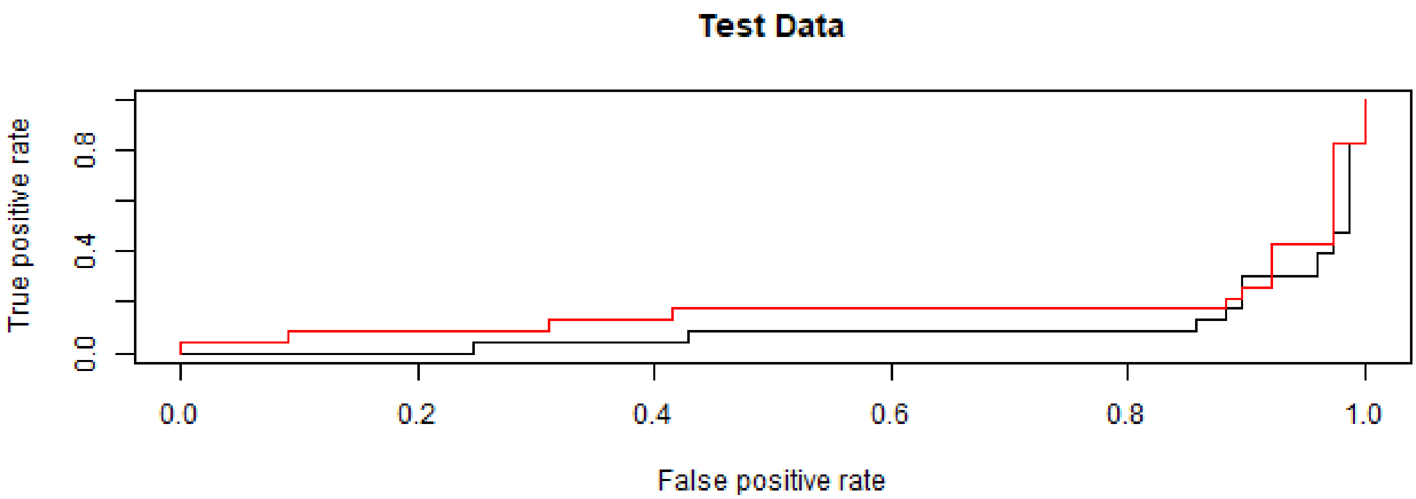
```
    pred
true  1  2
   1 67 10
   2  2 21
```

ROC Curves

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

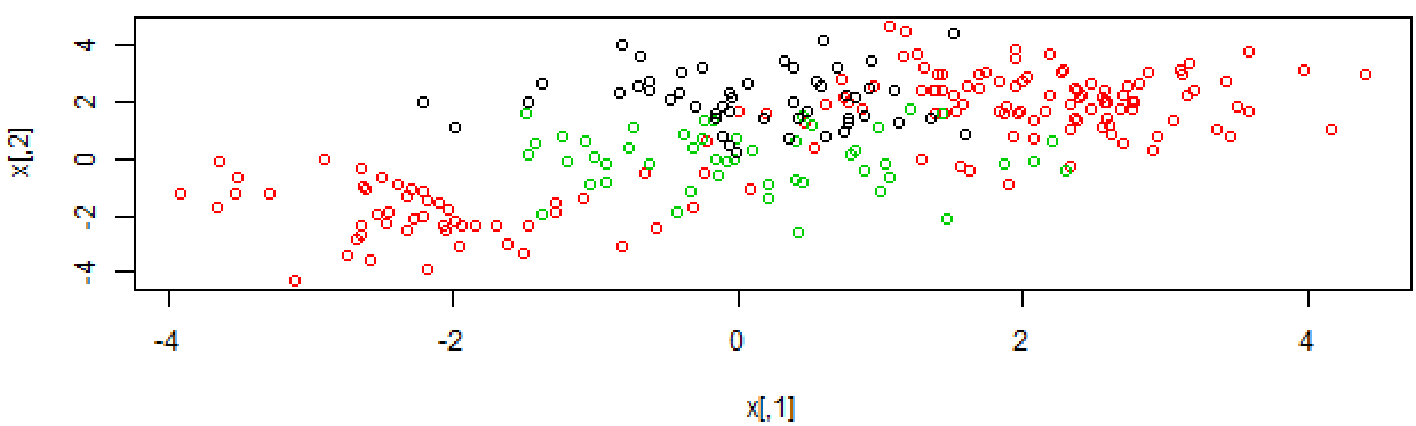
```
svmfit.opt <- svm(y ~ ., data = dat[train,], kernel = "radial",
                  gamma = 2, cost = 1, decision.values = T)

fitted <- attributes(predict(svmfit.opt, dat[train,], decision.values = T))$decision.values

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


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

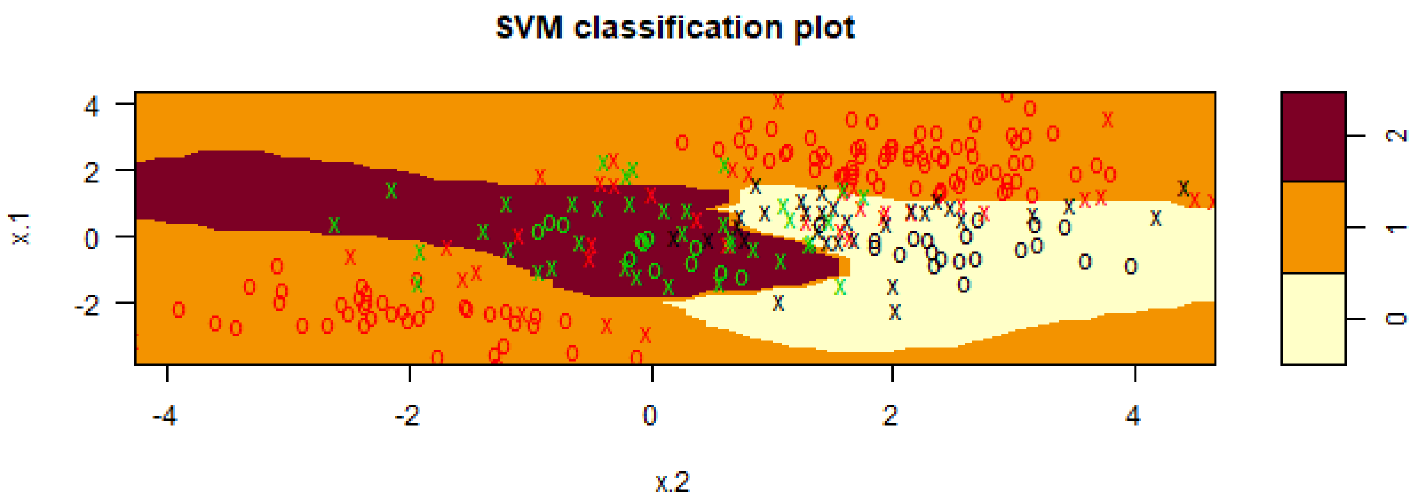


SVM w/ Multiple Classes

```
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))
```



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



Gene Expression Data

```r
khan <- ISLR::Khan
```

```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
```

```r
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
```

```r
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
```

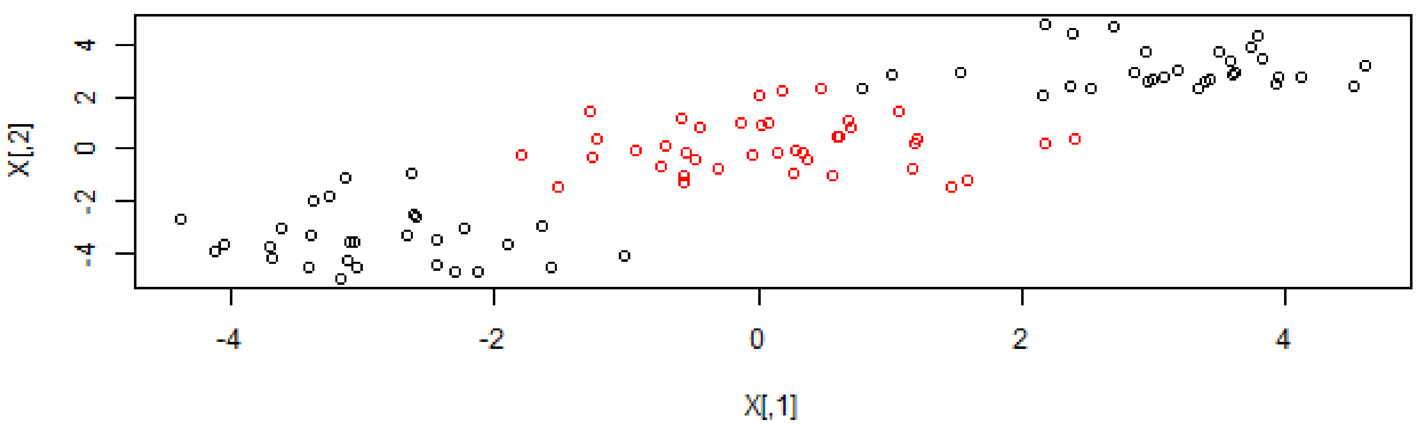
Applied

4.) Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

Generate data and plot:

```r
set.seed(1)
```

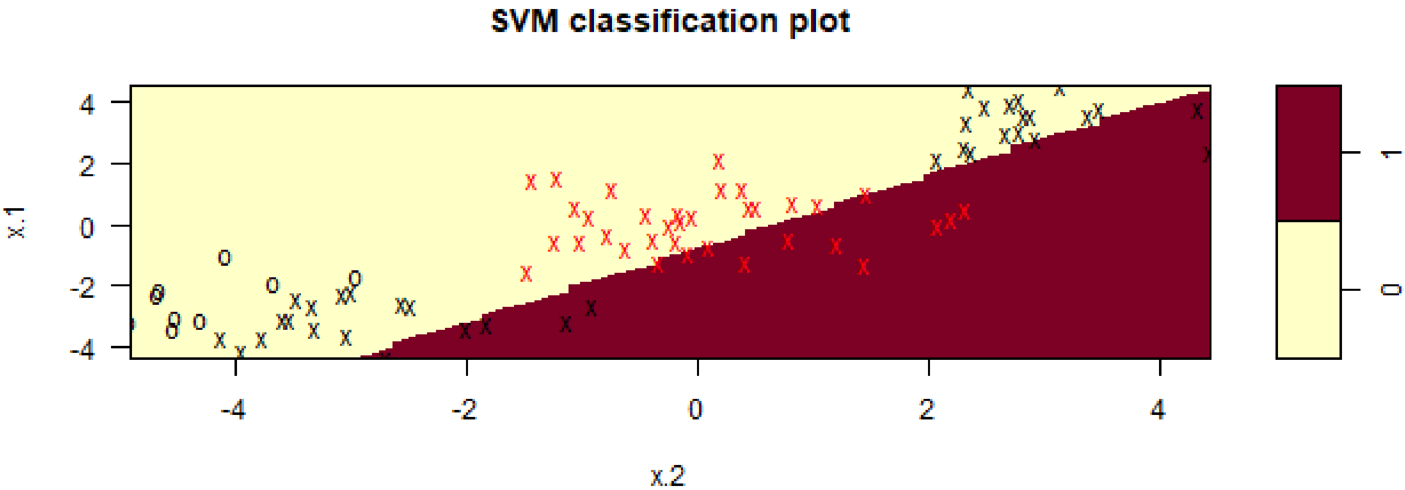
```r
transl <- 3
X <- matrix(rnorm(100 * 2), ncol = 2)
X[1:30, ] <- X[1:30, ] + transl
X[31:60, ] <- X[31:60, ] - transl
y <- c(rep(0, 60), rep(1, 40))
dat <- data.frame(x = X, y = as.factor(y))

par(mfrow = c(1,1))
plot(X, col = y + 1)
```



Split to training and test set:

```r
train <- sample(100, 80)
dat.train <- dat[train, ]
dat.test <- dat[-train, ]
```

```r
svm.lin <- svm(y ~ ., data = dat.train, kernel = 'linear', scale = FALSE)
plot(svm.lin, data = dat.train)
```




```r
summary(svm.lin)
```

```
Call:
svm(formula = y ~ ., data = dat.train, kernel = "linear", scale = FALSE)


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

Number of Support Vectors:  71

 ( 36 35 )


Number of Classes:  2

Levels:
 0 1
```

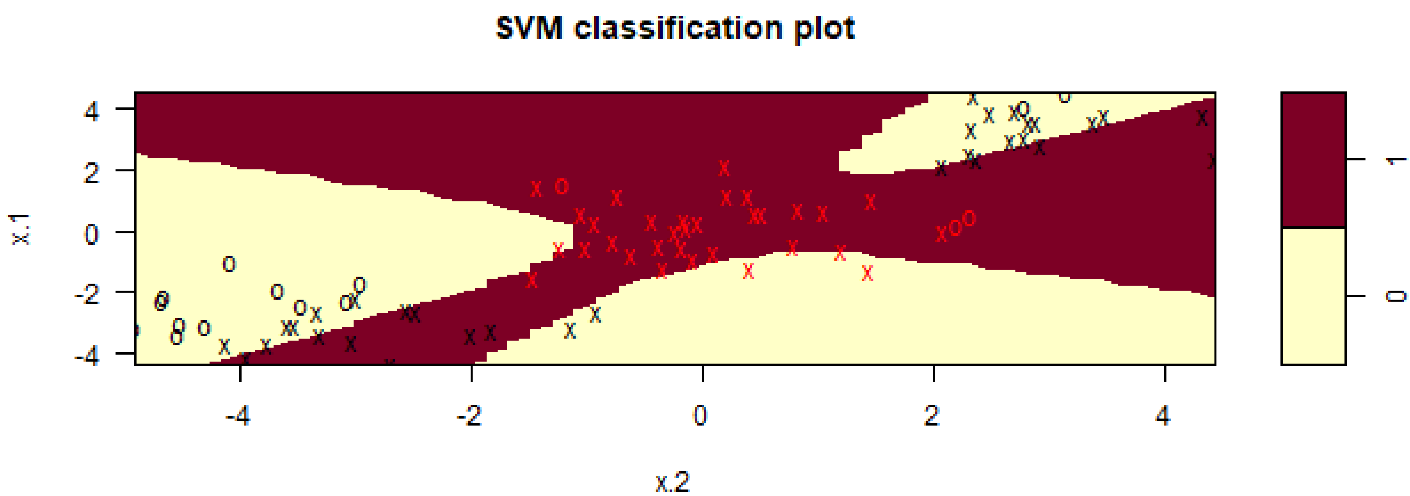
Calculate the training error of the support vector classifier:

```r
table(predict = svm.lin$fitted, truth = dat.train$y)
```

```
       truth
predict  0  1
      0 38 24
      1  7 11
```

Fit with polynomial kernel and calculate the training error rate:

```
svm.poly <- svm(y ~ ., data = dat.train, kernel = 'polynomial', scale = FALSE)
plot(svm.poly, data = dat.train)
```

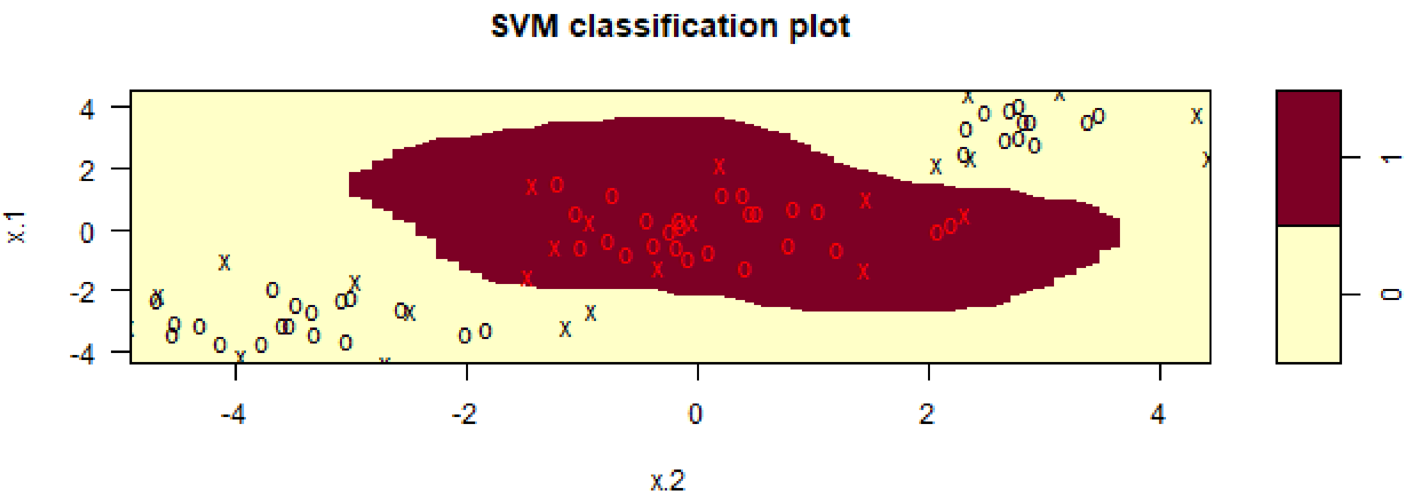



```
table(predict = svm.poly$fitted, truth = dat.train$y)
```

```
       truth
predict  0  1
      0 31  2
      1 14 33
```

Fit with radial kernel and calculate the traing error rate:

```
svm.rad <- svm(y ~ ., data = dat.train, kernel = 'radial', scale = FALSE)
plot(svm.rad, data = dat.train)
```




```r
table(predict = svm.rad$fitted, truth = dat.train$y)
```

```
       truth
predict  0  1
      0 45  0
      1  0 35
```

Compare the test errors of the 3 kernels:

```r
lin.pred <- predict(svm.lin, dat.test)
table(predict = lin.pred, truth = dat.test$y)
```

```
       truth
predict 0 1
      0 7 1
      1 8 4
```

```r
poly.pred <- predict(svm.poly, dat.test)
table(predict = poly.pred, truth = dat.test$y)
```

```
       truth
predict 0 1
      0 6 1
      1 9 4
```

```r
rad.pred <- predict(svm.rad, dat.test)
table(predict = rad.pred, truth = dat.test$y)
```

```
       truth
predict  0  1
      0 13  0
      1  2  5
```

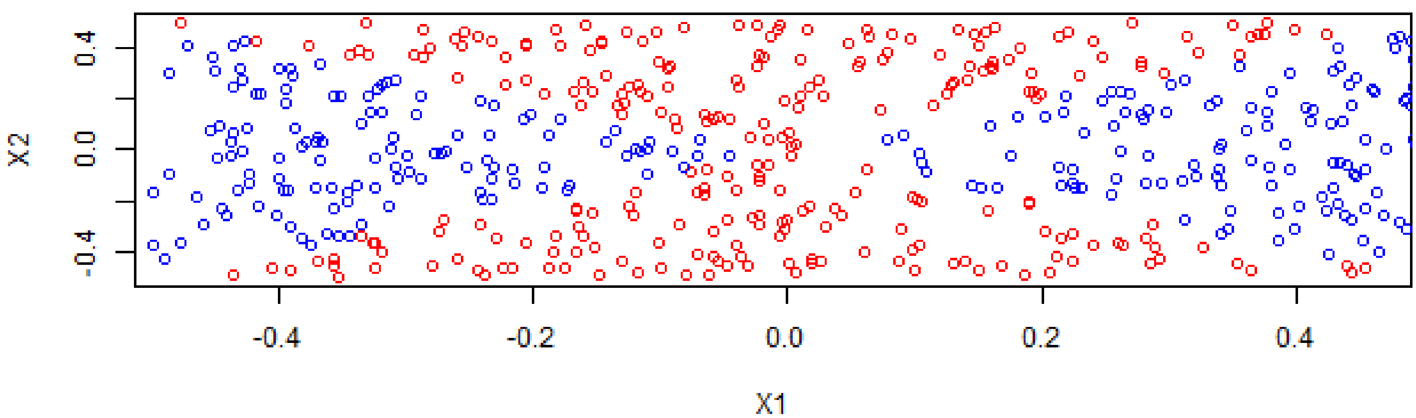
5.) We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

a.) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them:

```
set.seed(1)

x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- as.integer(x1 ^ 2 - x2 ^ 2 > 0)
```

b.) Plot the observations, colored according to their class labels. Your plot should display X 1 on the x-axis, and X 2 on the y-axis:

```
plot(x1[y == 0], x2[y == 0], col = "red", xlab = "X1", ylab = "X2")
points(x1[y == 1], x2[y == 1], col = "blue")
```

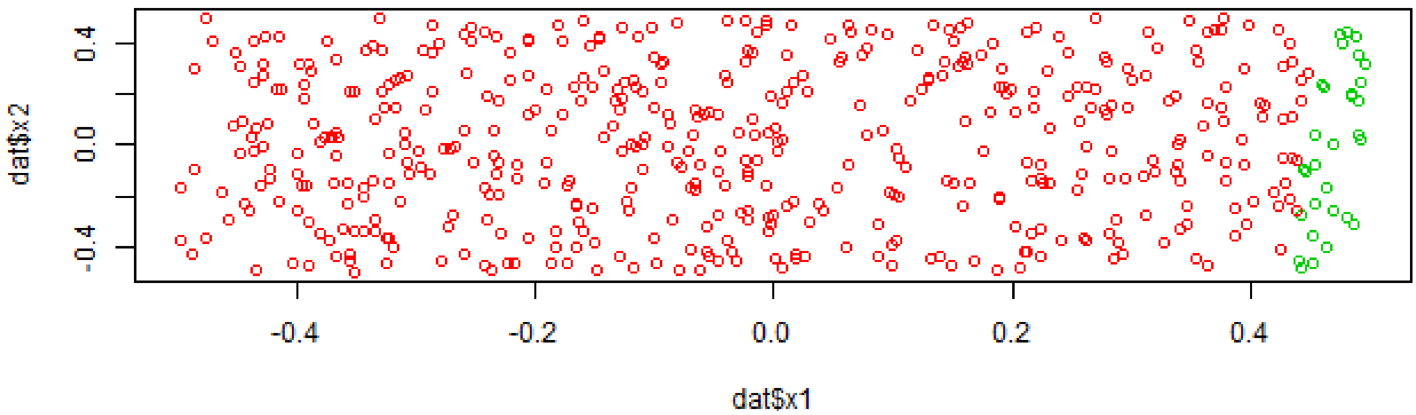


Fit a logistic regression model to the data, using X1 and X2 as predictors.

```
dat <- data.frame(x1 = x1, x2 = x2, y = as.factor(y))
lr.fit <- glm(y ~ ., data = dat, family = 'binomial')
```

d.) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```
lr.prob <- predict(lr.fit, newdata = dat, type = 'response')
lr.pred <- ifelse(lr.prob > 0.5, 1, 0)
plot(dat$x1, dat$x2, col = lr.pred + 2)
```



e.) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X21 , X1×X2, log(X2), and so forth).

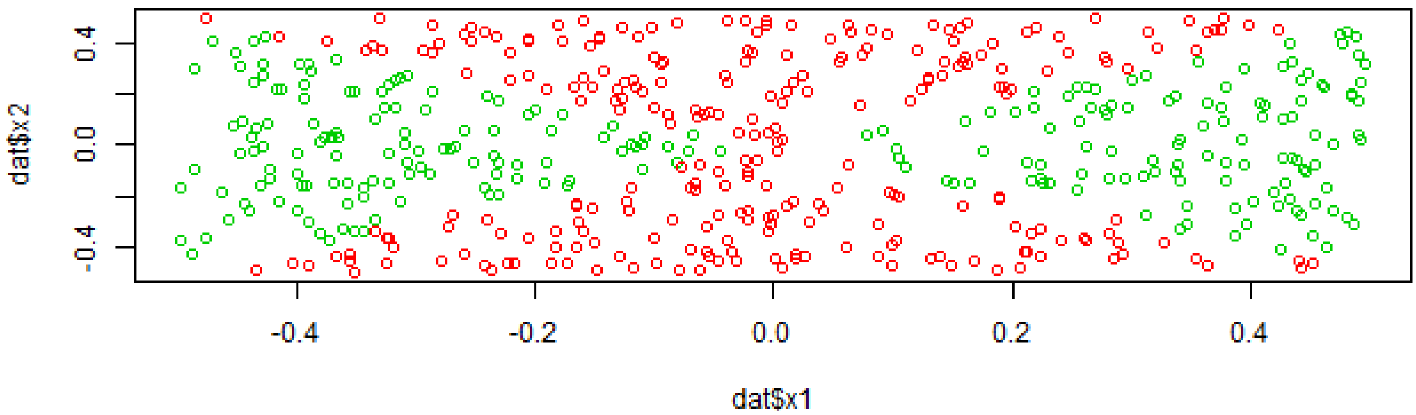
```
lr.nl <- glm(y ~ poly(x1, 2) + poly(x2, 2), data = dat, family = 'binomial')
```

```
Warning: glm.fit: algorithm did not converge
```

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

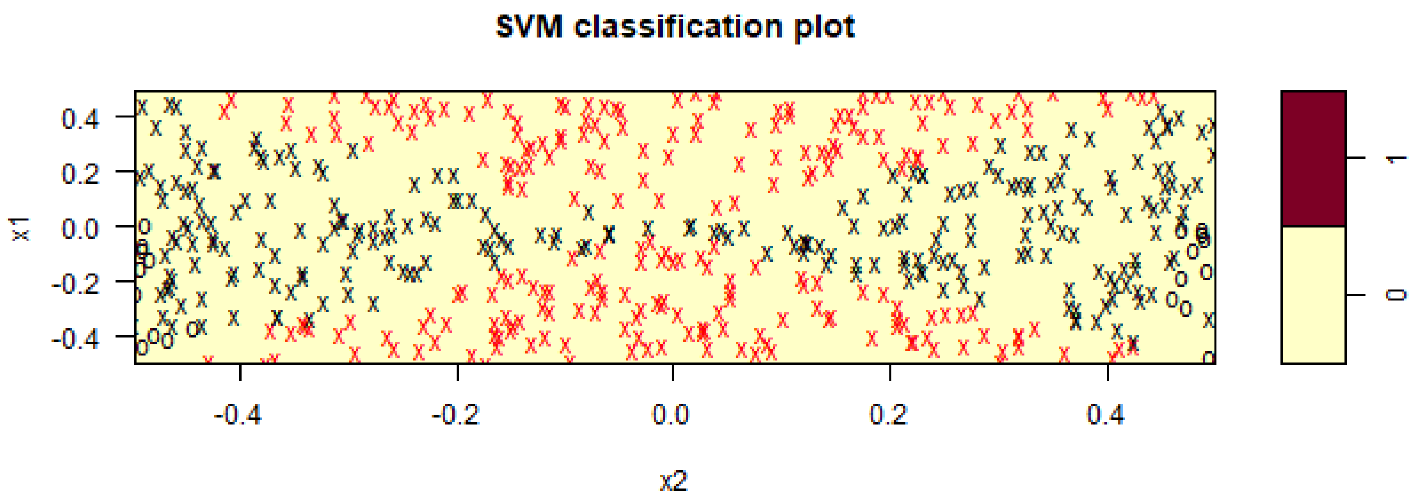
f.) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

```
lr.prob.nl <- predict(lr.nl, newdata = dat, type = 'response')
lr.pred.nl <- ifelse(lr.prob.nl > 0.5, 1, 0)
plot(dat$x1, dat$x2, col = lr.pred.nl + 2)
```



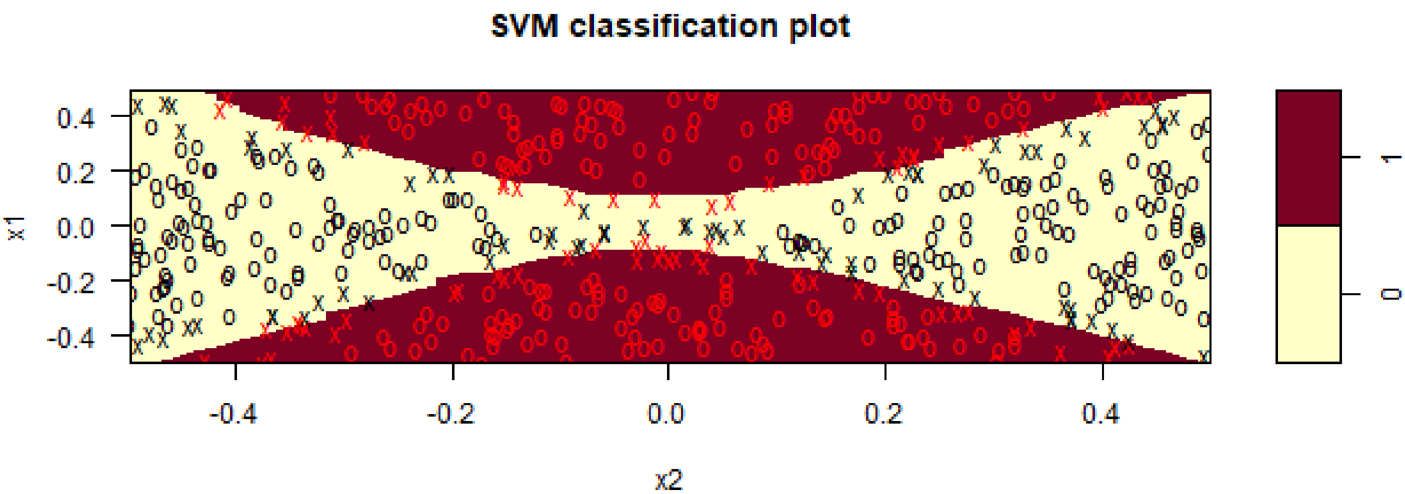
g.) Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
svm.lin <- svm(y ~ ., data = dat, kernel = 'linear', cost = 0.01)
plot(svm.lin, dat)
```



h.) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

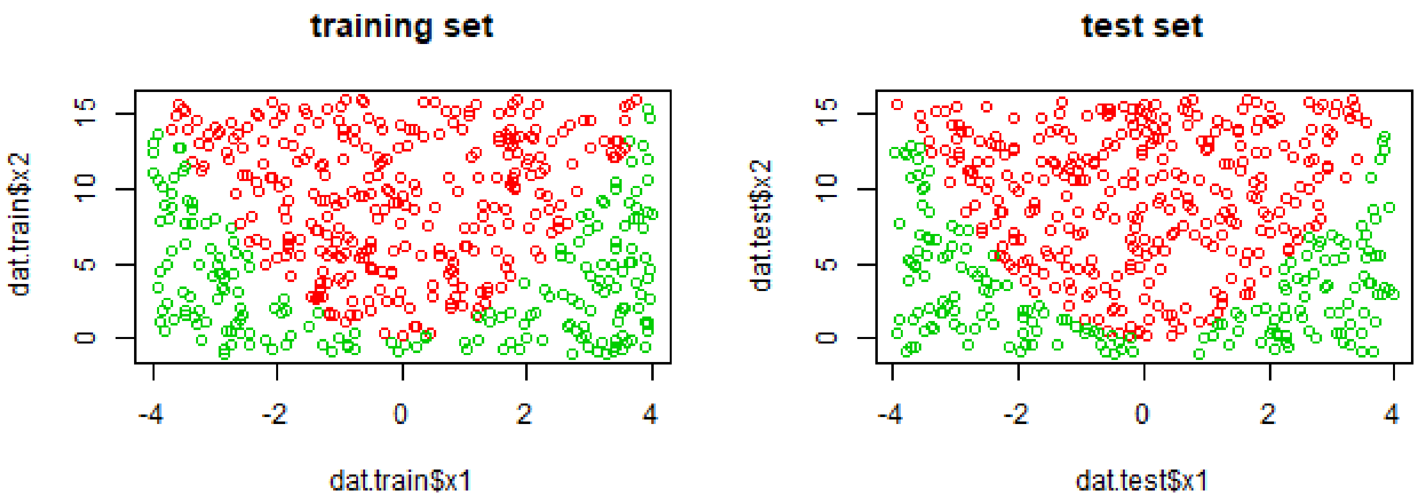
```
svm.nl <- svm(y ~ ., data = dat, kernel = 'radial', gamma = 1)
plot(svm.nl, data = dat)
```



6.) At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations. You will now investigate this claim.

a.) Generate two-class data with p = 2 in such a way that the classes are just barely linearly separable.

```
set.seed(1)
obs = 1000
x1 <- runif(obs, min = -4, max = 4)
x2 <- runif(obs, min = -1, max = 16)
y <- ifelse(x2 > x1 ^ 2, 0, 1)
dat <- data.frame(x1 = x1, x2 = x2, y = as.factor(y))
train <- sample(obs, obs/2)
dat.train <- dat[train, ]
dat.test <- dat[-train, ]
par(mfrow = c(1,2))
plot(dat.train$x1, dat.train$x2, col = as.integer(dat.train$y) + 1, main = 'training set')
plot(dat.test$x1, dat.test$x2, col = as.integer(dat.test$y) + 1, main = 'test set')
```



b.) Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training errors are misclassified for each value of cost considered, and how does this relate to the cross-validation errors obtained?

```
set.seed(1)
cost.grid <- c(0.001, 0.01, 0.1, 1, 5, 10, 100, 10000)
tune.out <- tune(svm, y ~., data = dat.train, kernel = 'linear', ranges = list(cost = cost.grid
```

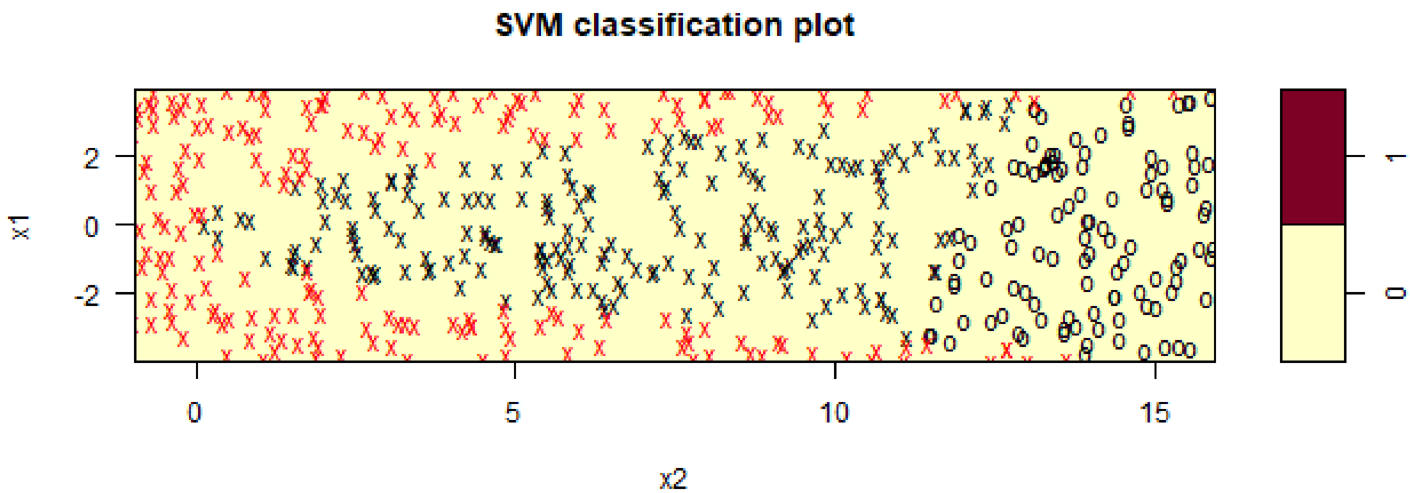
```
summary(tune.out)
```

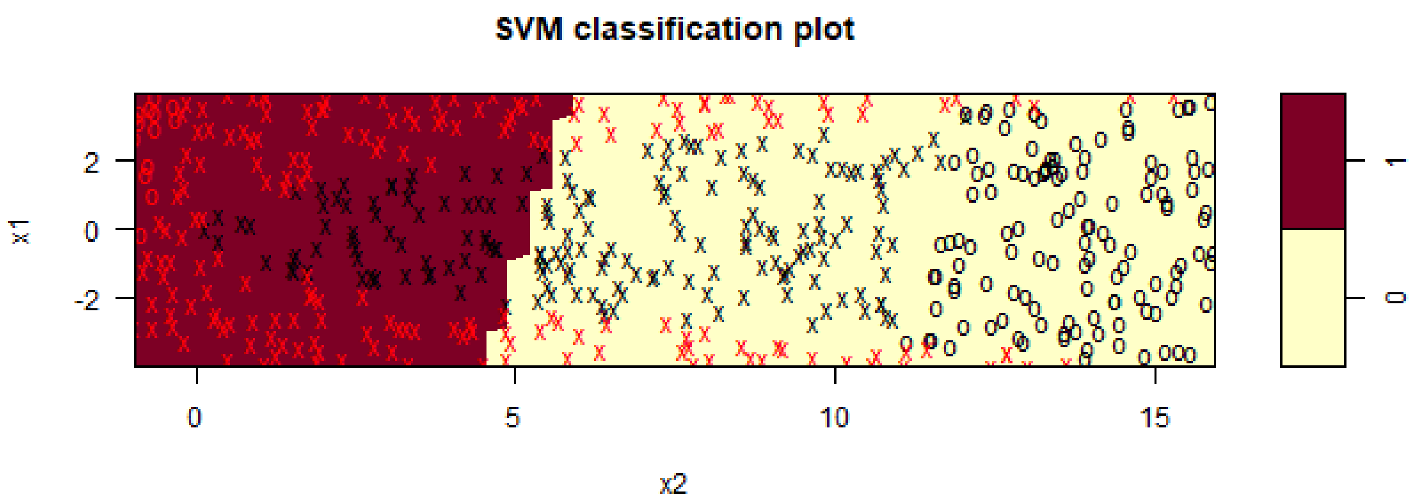
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    5

- best performance: 0.25

- Detailed performance results:
   cost error dispersion
1 1e-03 0.392 0.04825856
2 1e-02 0.256 0.05947922
3 1e-01 0.252 0.05902918
4 1e+00 0.252 0.06051630
5 5e+00 0.250 0.06271629
6 1e+01 0.250 0.06271629
7 1e+02 0.250 0.06271629
```

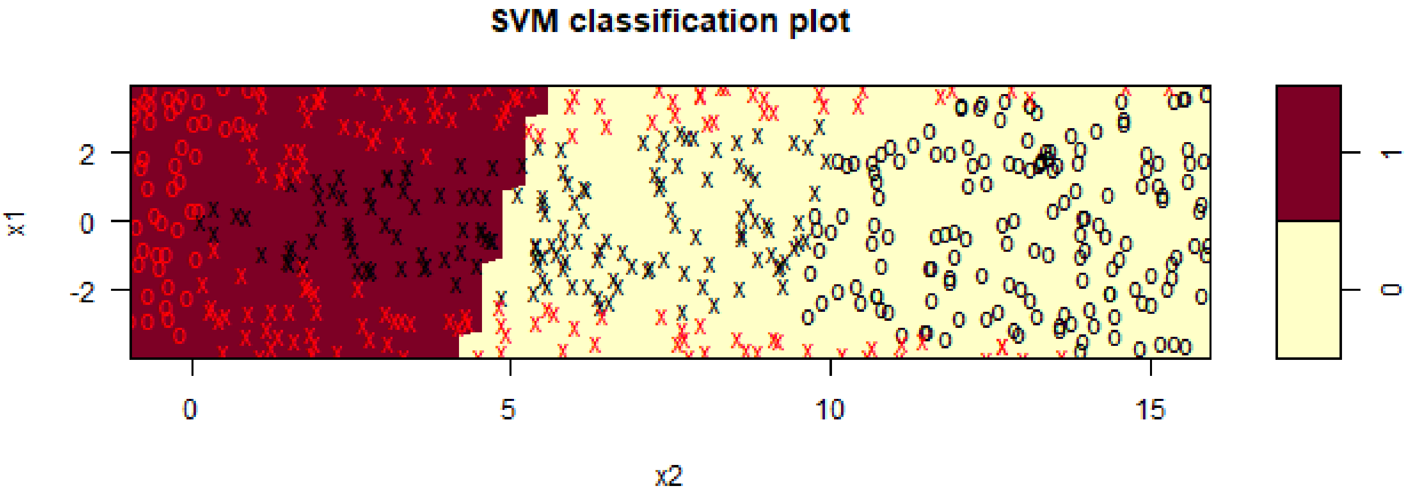
```
8 1e+04 0.390 0.05270463
```

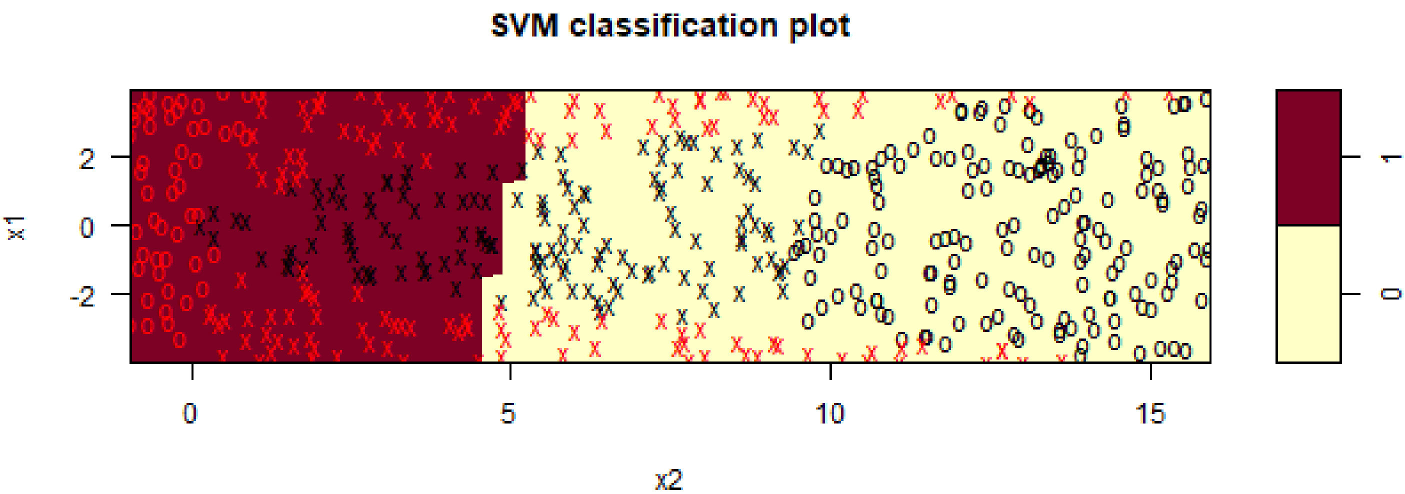
c.) Generate an appropriate trest data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that generate the fewest train errors?

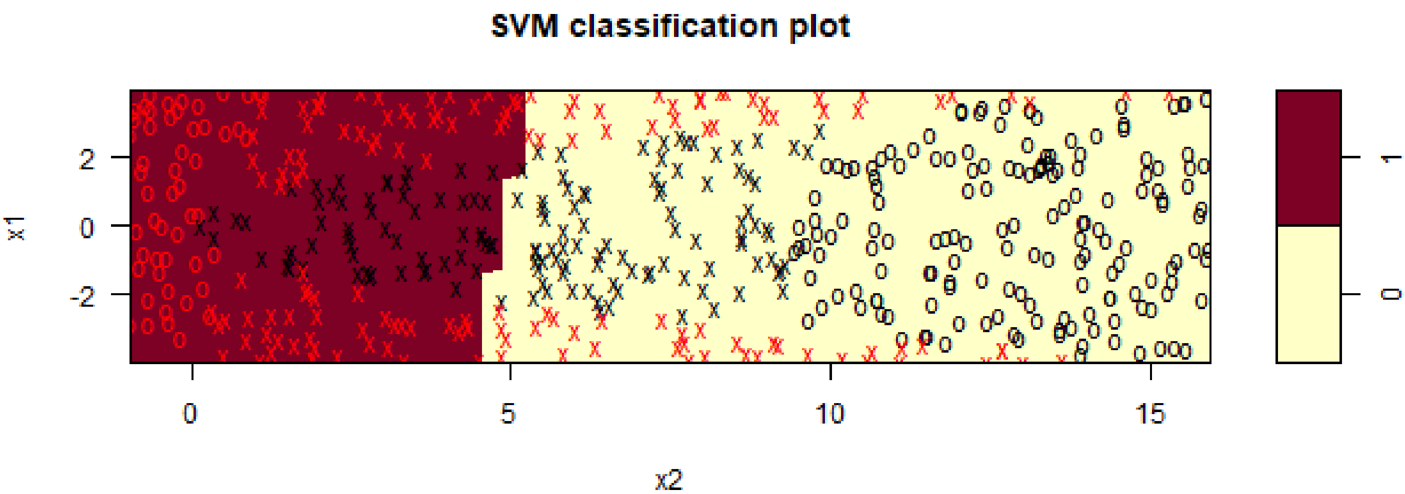
```
err.rate.train <- rep(NA, length(cost.grid))
for (cost in cost.grid) {
  svm.fit <- svm(y ~ ., data = dat.train, kernel = 'linear', cost = cost)
  plot(svm.fit, data = dat.train)
  res <- table(prediction = predict(svm.fit, newdata = dat.train), truth = dat.train$y)
  err.rate.train[match(cost, cost.grid)] <- (res[2,1] + res[1,2]) / sum(res)
}
```

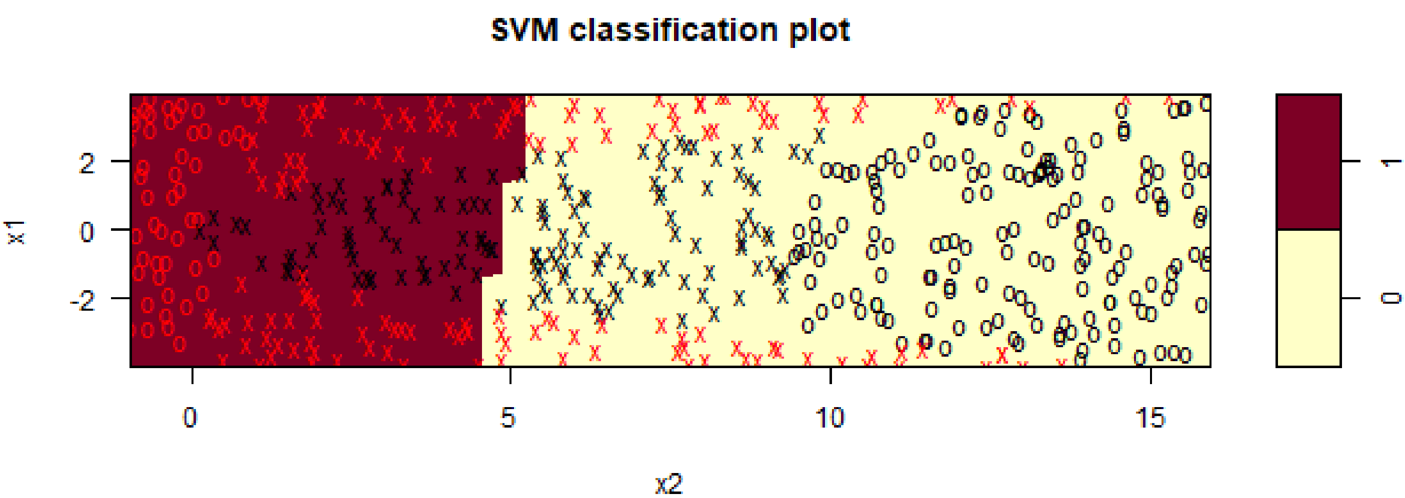


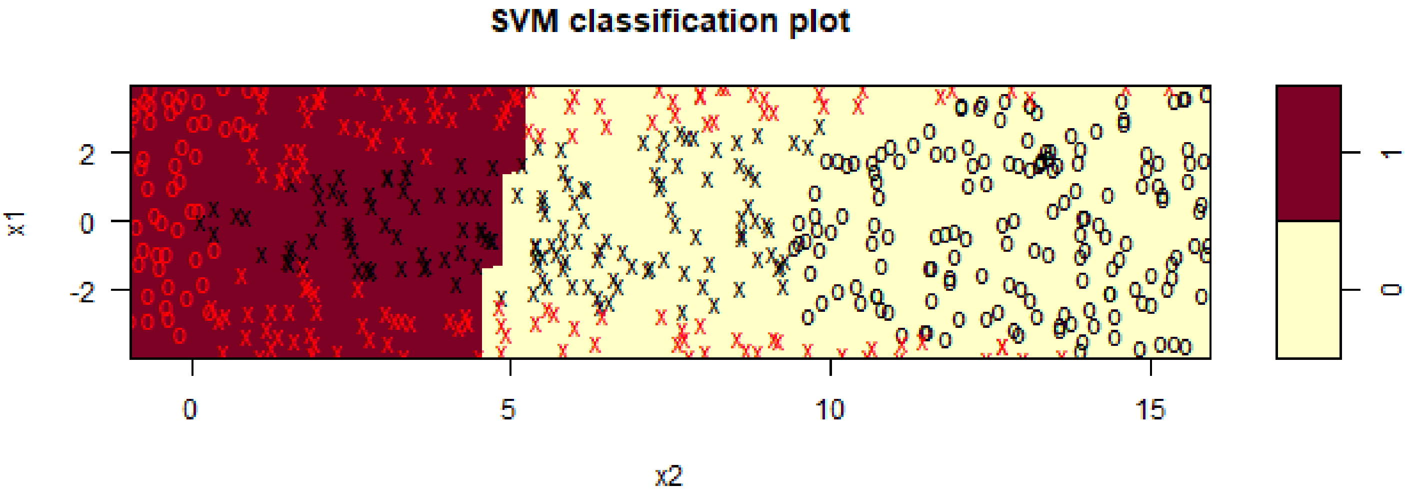


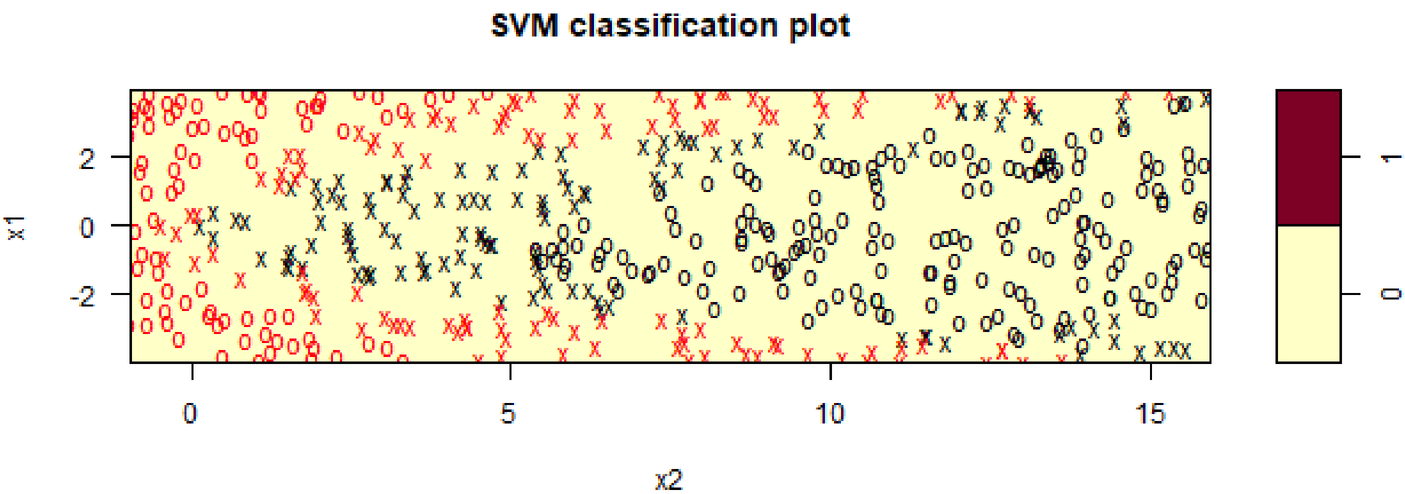

SVM classification plot
x1
2
0
-2
0
5
10
15
x2
1
0


SVM classification plot
x1
2
0
-2
0
5
10
15
x2
1
0


SVM classification plot
x1
x2
2
0
-2
0
5
10
15
1
0


SVM classification plot
x1
x2
2
0
-2
0
5
10
15
1
0






```
paste('The cost', cost.grid[which.min(err.rate.train)], 'has the minimum training error:', min(
```

```
[1] "The cost 0.01 has the minimum training error: 0.246"
```

7.) In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

```
auto <- as.data.table(ISLR::Auto)
```

a.) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage with below the median.

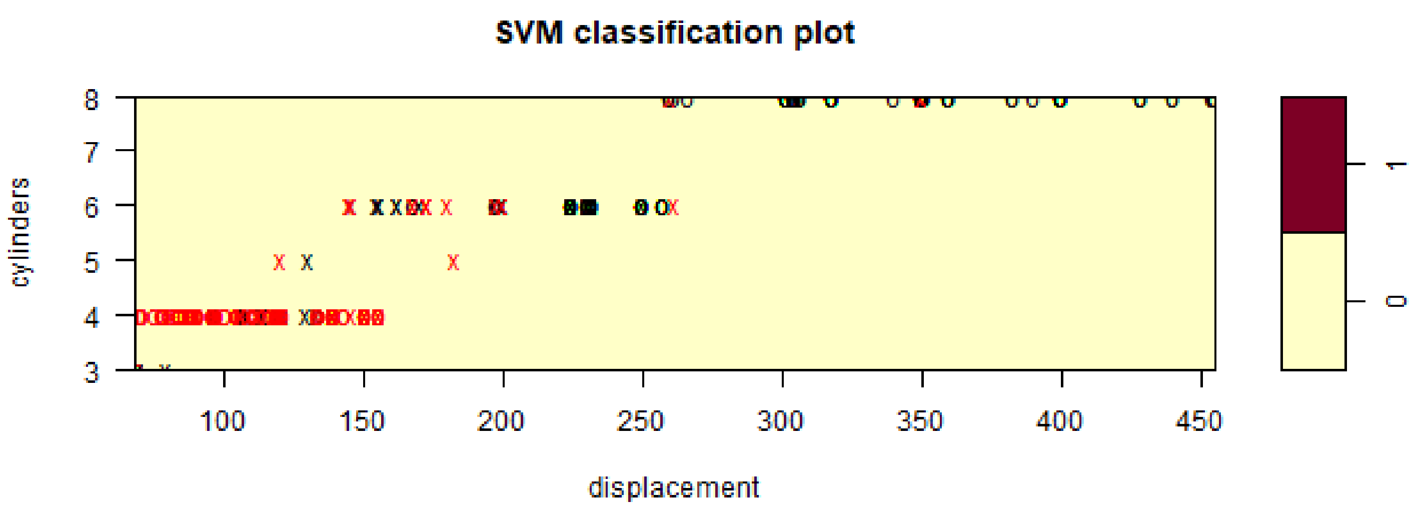
```
auto$eco <- as.factor(ifelse(auto$mpg < median(auto$mpg), 0, 1))
training <- auto[, !"mpg"]
```

b.) Fit a support vector classificer to the data with various values of cost, in order to predict whetehr a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter.

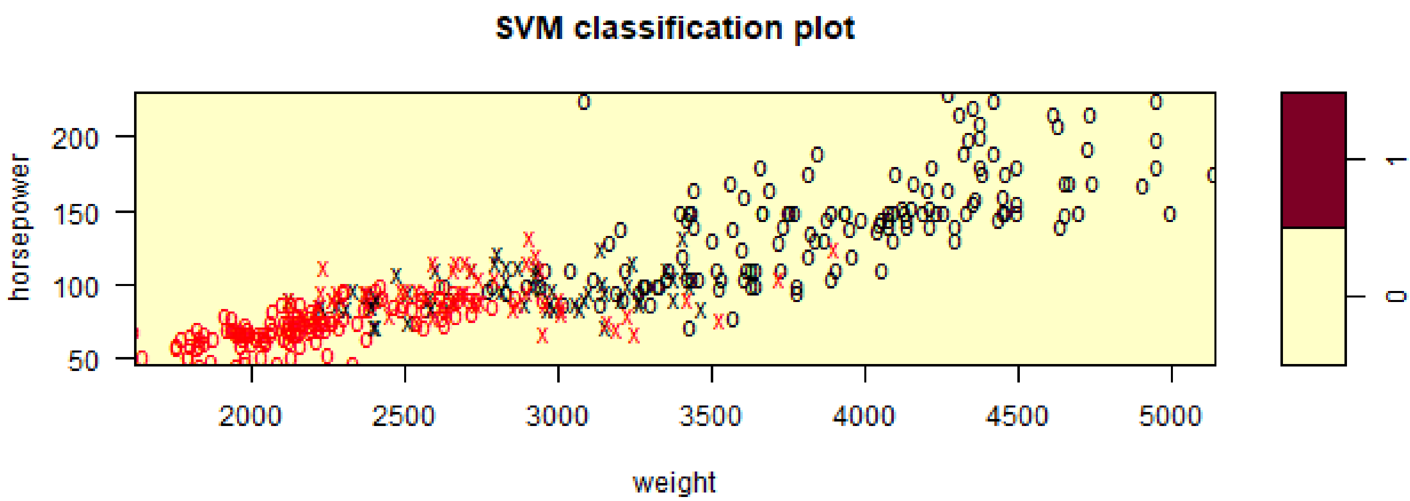
```
cost.grid <- c(0.001, 0.1, 1, 100)
set.seed(1)
tune.res <- tune(svm, eco ~ ., data = training, kernel = 'linear', ranges = list(cost = cost.gr
summary(tune.res)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.08673077

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.13525641 0.05661708
2 1e-01 0.08673077 0.04040897
3 1e+00 0.09961538 0.04923181
4 1e+02 0.11750000 0.06208951
```

```
plot(tune.res$best.model, training, cylinders ~ displacement)
```



```r
plot(tune.res$best.model, training, horsepower ~ weight)
```

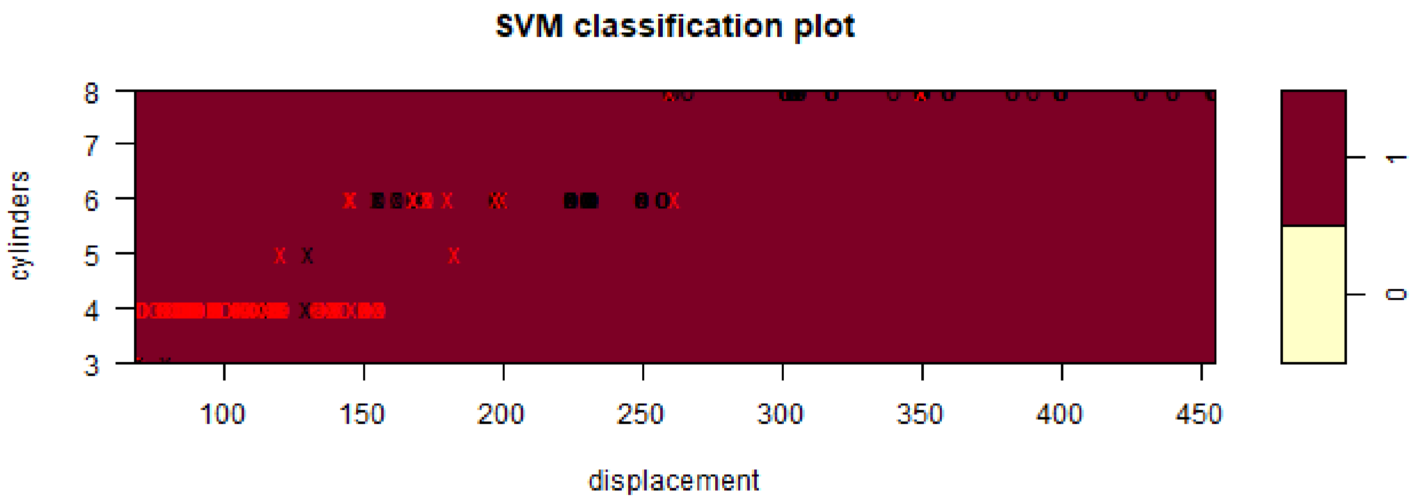


c.) Now repeat this with SVMs with radial and polynomial basis kernels, which different values of gamma and degree.

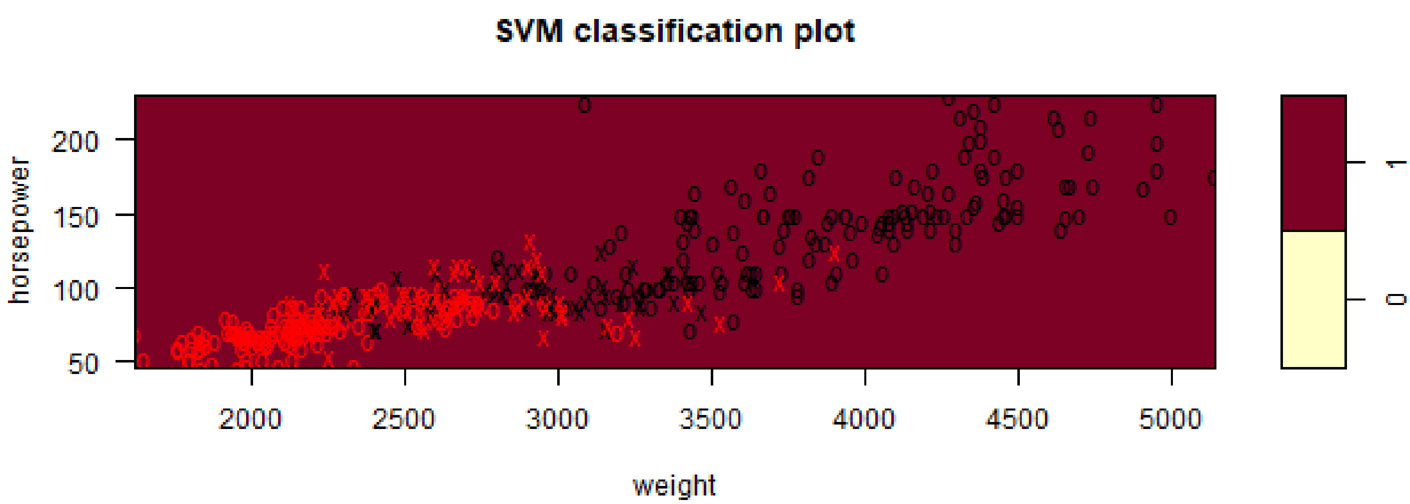
```r
cost.grid <- c(0.001, 0.1, 1, 100)
set.seed(1)
tune.res <- tune(svm, eco ~ ., data = training, kernel = 'radial', ranges = list(cost = cost.gr
summary(tune.res)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  100

- best performance: 0.08692308

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.55115385 0.04366593
2 1e-01 0.16852564 0.07871283
3 1e+00 0.09179487 0.03837336
4 1e+02 0.08692308 0.04886318
```

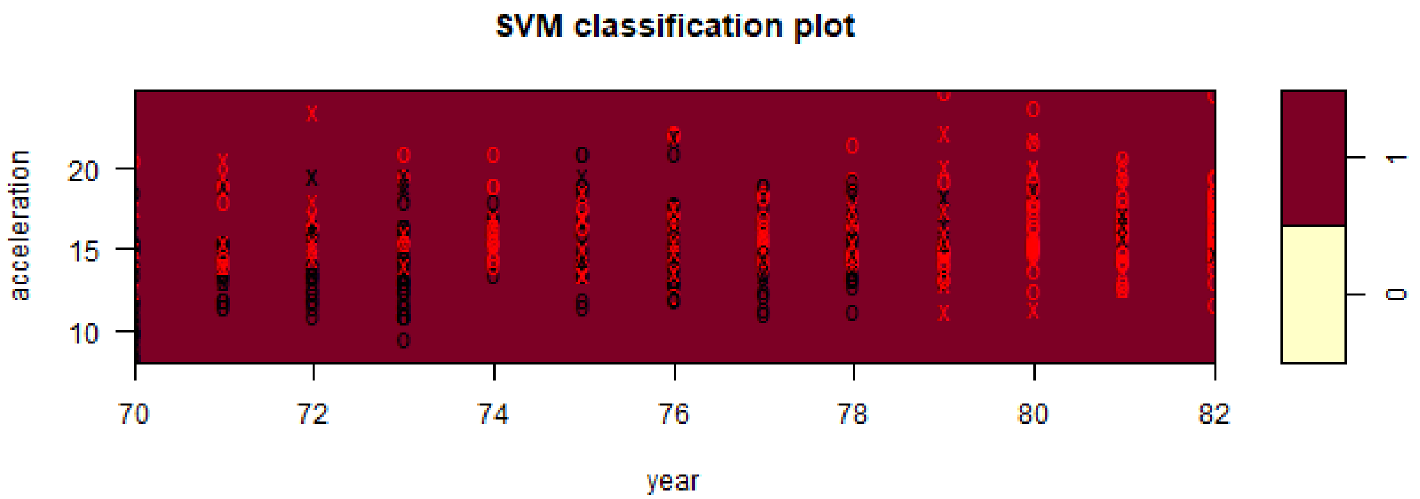
```r
plot(tune.res$best.model, training, cylinders ~ displacement)
```




```r
plot(tune.res$best.model, training, horsepower ~ weight)
```



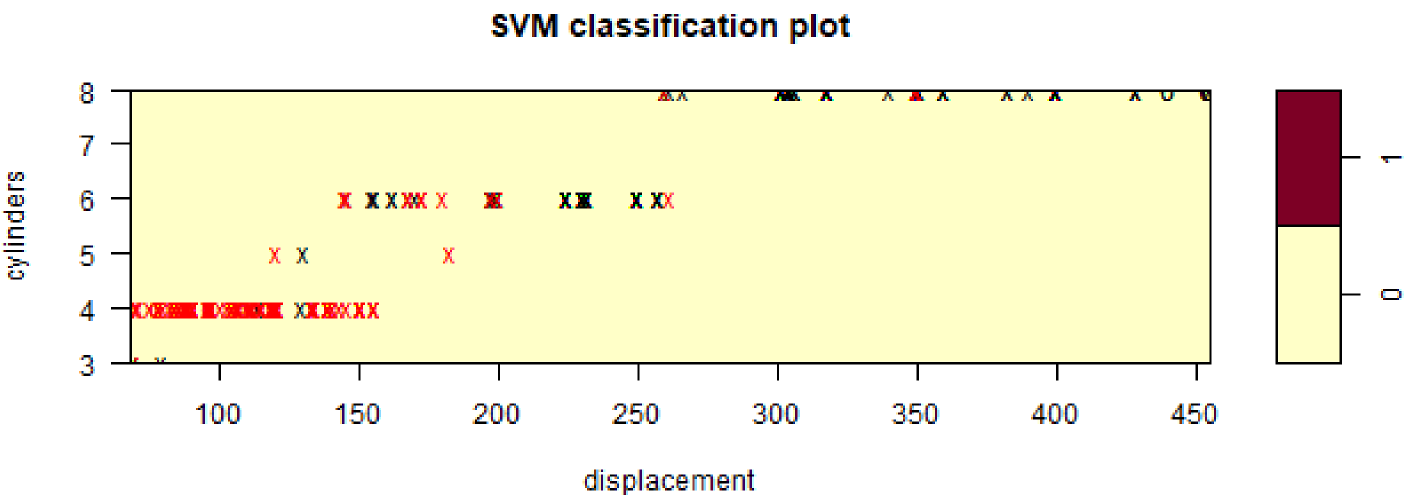

```r
plot(tune.res$best.model, training, acceleration ~ year)
```



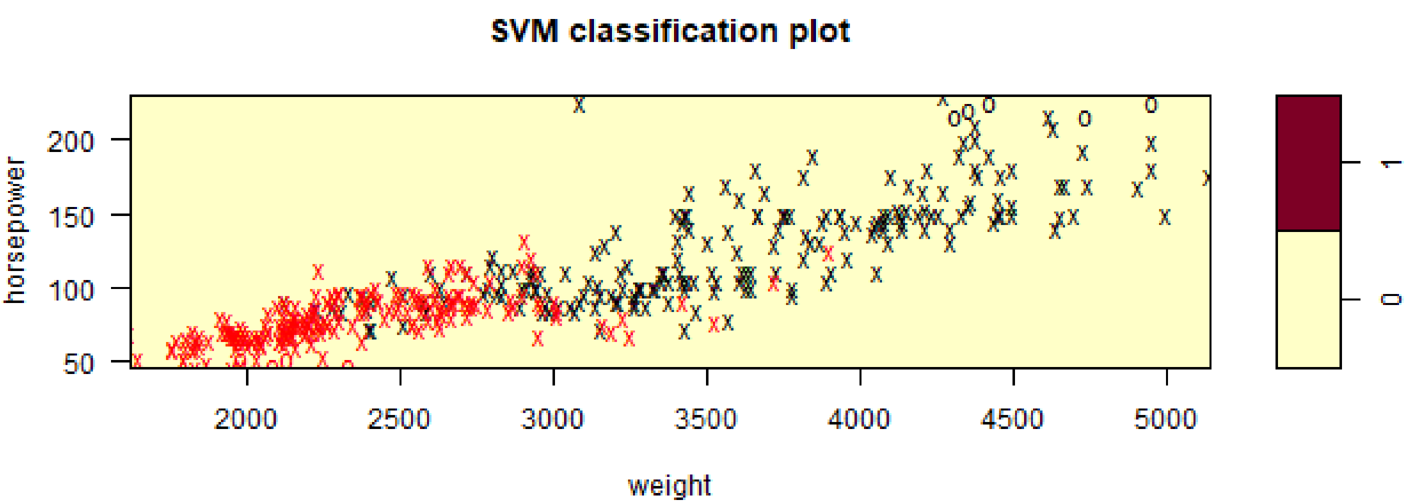

```
cost.grid <- c(0.001, 0.1, 1, 100)
set.seed(1)
tune.res <- tune(svm, eco ~ ., data = training, kernel = 'polynomial', ranges = list(cost = cos
summary(tune.res)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  100

- best performance: 0.4032692

- Detailed performance results:
   cost     error dispersion
1 1e-03 0.5511538 0.04366593
2 1e-01 0.5511538 0.04366593
3 1e+00 0.5511538 0.04366593
4 1e+02 0.4032692 0.10793388
```

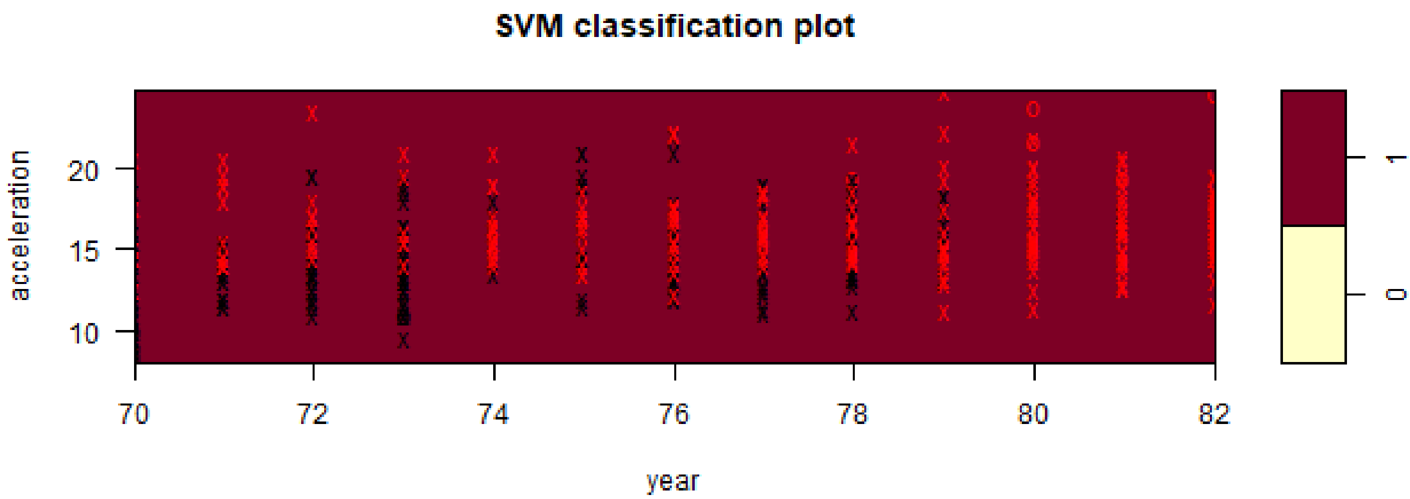
```
plot(tune.res$best.model, training, cylinders ~ displacement)
```




```r
plot(tune.res$best.model, training, horsepower ~ weight)
```




```r
plot(tune.res$best.model, training, acceleration ~ year)
```



8.) This problem involves the OJ data set which is part of the ISLR package.

```
oj <- as.data.table(ISLR::OJ)
```

a.) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
train <- sample(1:nrow(oj), 800)

oj.train <- oj[train,]
oj.test <- oj[-train,]
```

b.) Fit a support vector machine classifier to the training data using cost = 0.01, with purchase as the response and other variables as the predictors.

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

summary(svmfit)
```

```

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


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

Number of Support Vectors:  436
```

```
 ( 219 217 )


Number of Classes:  2

Levels:
 CH MM
```

c.) What are the train and test error rates?

Test error rate:

```r
table(oj.train$Purchase, predict(svmfit))
```

```
      CH  MM
  CH 443  56
  MM  71 230
```

```r
mean(oj.train$Purchase == predict(svmfit))
```

```
[1] 0.84125
```

Test error rate:

```r
table(oj.test$Purchase, predict(svmfit, newdata = oj.test))
```

```
      CH  MM
  CH 126  28
  MM  27  89
```

```r
mean(oj.test$Purchase == predict(svmfit, newdata = oj.test))
```

```
[1] 0.7962963
```

d.) Use the tune() function to select an optimal model.

```r
cost.grid <- c(0.001, 0.1, 1, 100, 250)
out <- tune(svm, Purchase ~ ., data = oj.train, kernel = "linear", ranges = list(cost = cost.gr

summary(out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
```

```
  0.1

- best performance: 0.1575

- Detailed performance results:
     cost   error dispersion
1   0.001 0.35000 0.06373774
2   0.100 0.15750 0.03016160
3   1.000 0.16250 0.03061862
4 100.000 0.17125 0.03175973
5 250.000 0.17125 0.03175973
```

e.) Compute the training and test error rates using this new value for cost:

```r
svmfit <- out$best.model

table(oj.train$Purchase, predict(svmfit))
```

```
      CH  MM
  CH 442  57
  MM  69 232
```

```r
mean(oj.train$Purchase == predict(svmfit))
```

```
[1] 0.8425
```

```r
table(oj.test$Purchase, predict(svmfit, newdata = oj.test))
```

```
      CH  MM
  CH 126  28
  MM  25  91
```

```r
mean(oj.test$Purchase == predict(svmfit, newdata = oj.test))
```

```
[1] 0.8037037
```

f.) Repeat parts b-d using a radial kernel.

```r
cost.grid <- c(0.001, 0.1, 1, 100, 250)
out <- tune(svm, Purchase ~ ., data = oj.train, kernel = "radial", ranges = list(cost = cost.gr

summary(out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
 cost
    1

- best performance: 0.16125

- Detailed performance results:
     cost   error dispersion
1   0.001 0.37625 0.03087272
2   0.100 0.17750 0.05489890
3   1.000 0.16125 0.06050999
4 100.000 0.19750 0.04958158
5 250.000 0.19625 0.05070681
```

```r
svmfit <- out$best.model

table(oj.train$Purchase, predict(svmfit))
```

```

      CH  MM
  CH 456  43
  MM  71 230
```

```r
mean(oj.train$Purchase == predict(svmfit))
```

```
[1] 0.8575
```

```r
table(oj.test$Purchase, predict(svmfit, newdata = oj.test))
```

```

      CH  MM
  CH 136  18
  MM  32  84
```

```r
mean(oj.test$Purchase == predict(svmfit, newdata = oj.test))
```

```
[1] 0.8148148
```

h.) Overall, which approach seems to give the best results?

The linear kernel has the best test error rate.

```r
rm(list = ls())
```