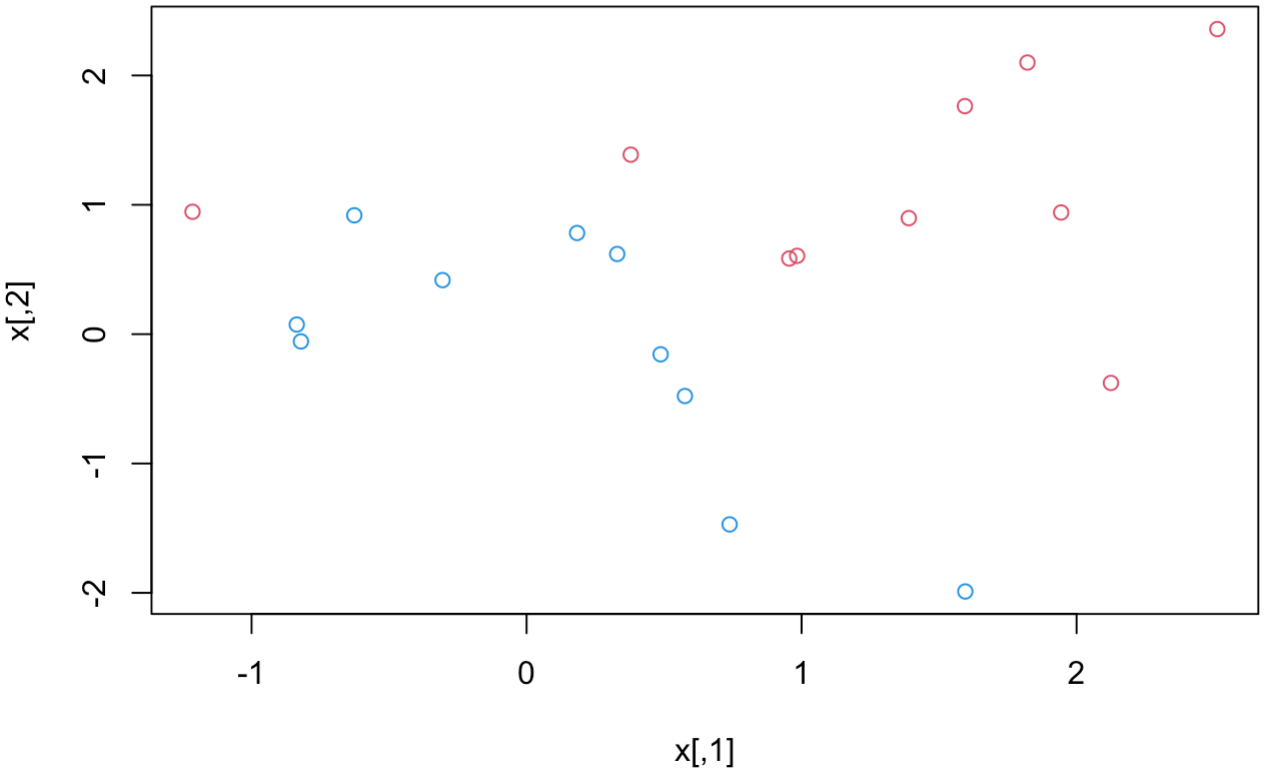
Optimized Kernel Selection for SVM Classification: A Case Study on the OJ Dataset

Preethi Sree Allam

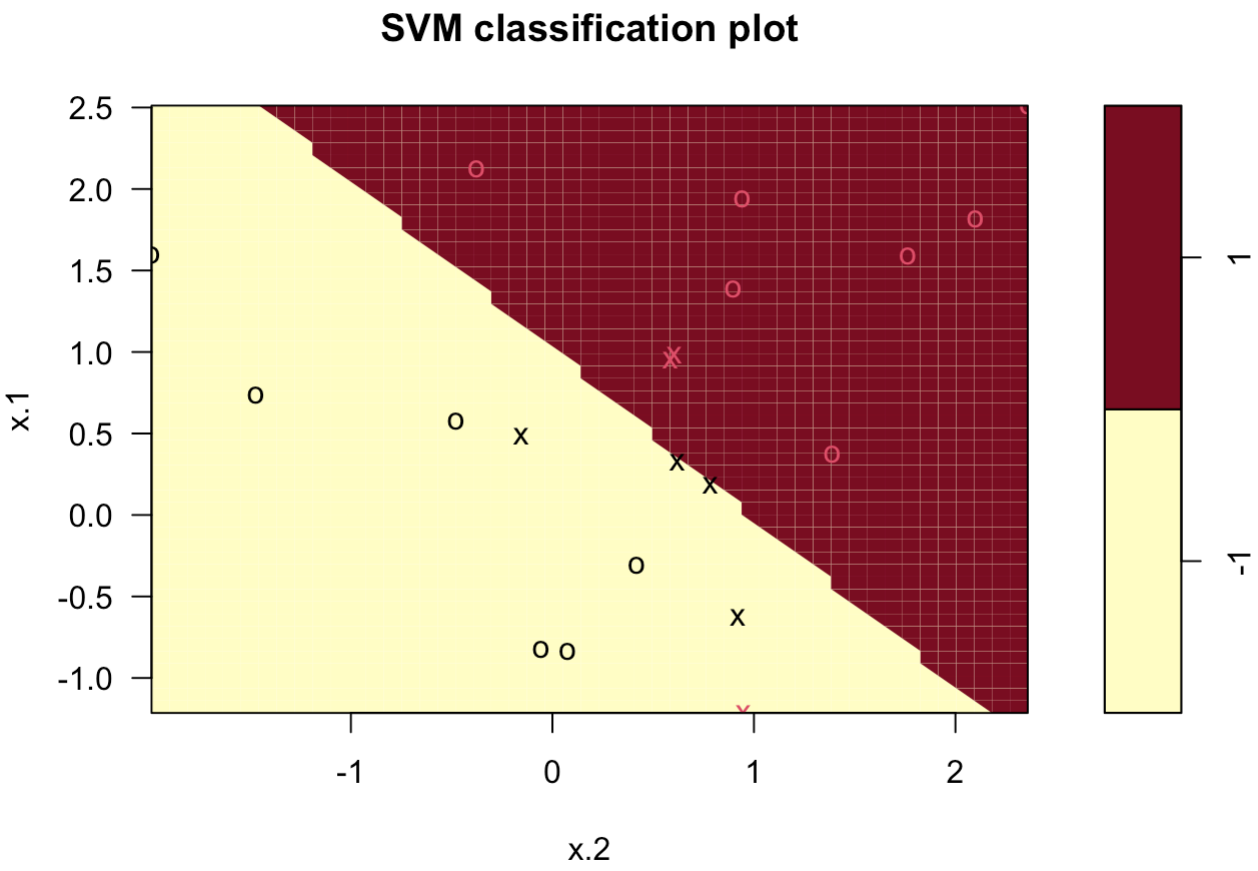
2023-04-28

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.frame(x = x, y = as.factor(y))
library(e1071)
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
              cost = 10, scale = FALSE)
plot(svmfit, dat)
```



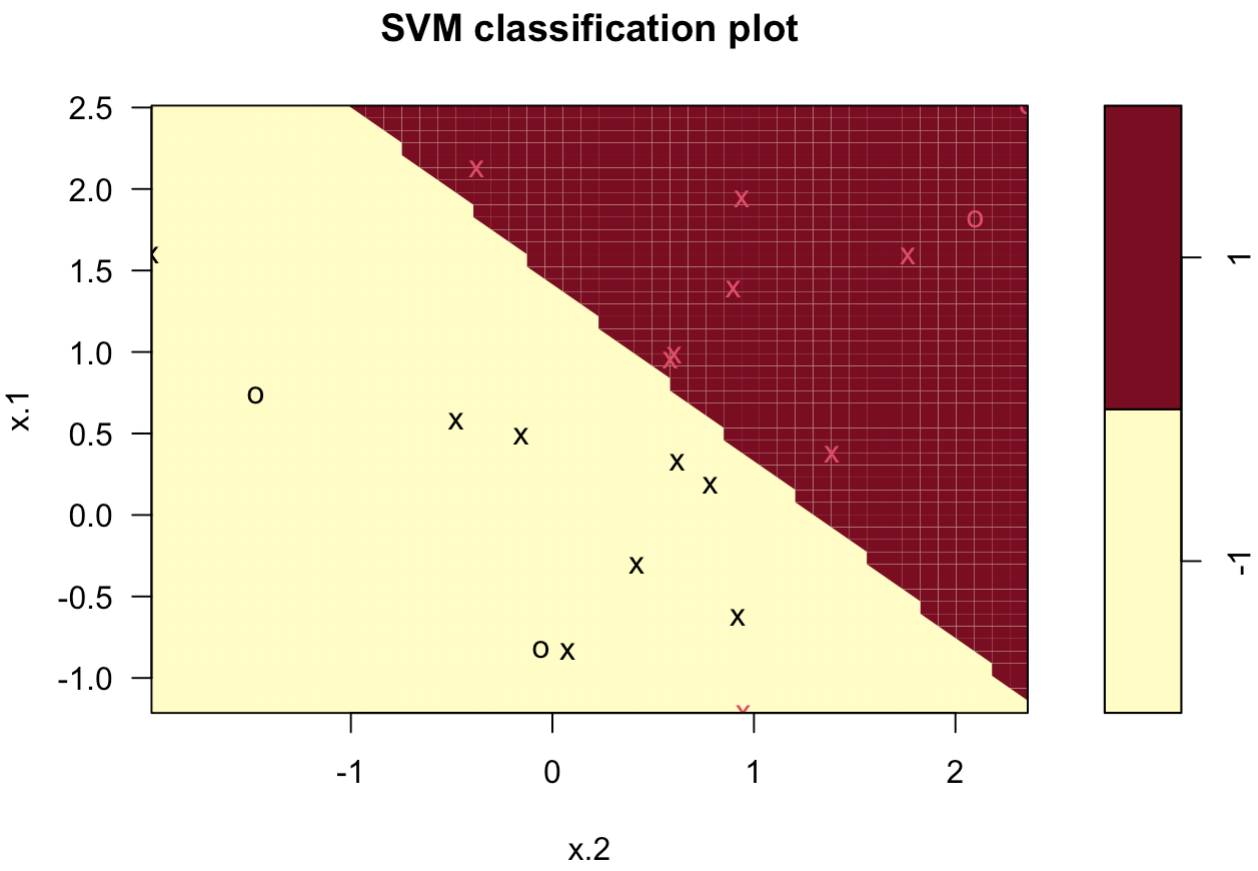

```
svmfit$index
```

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

```
summary(svmfit)
```

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

```
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
```

```
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
```

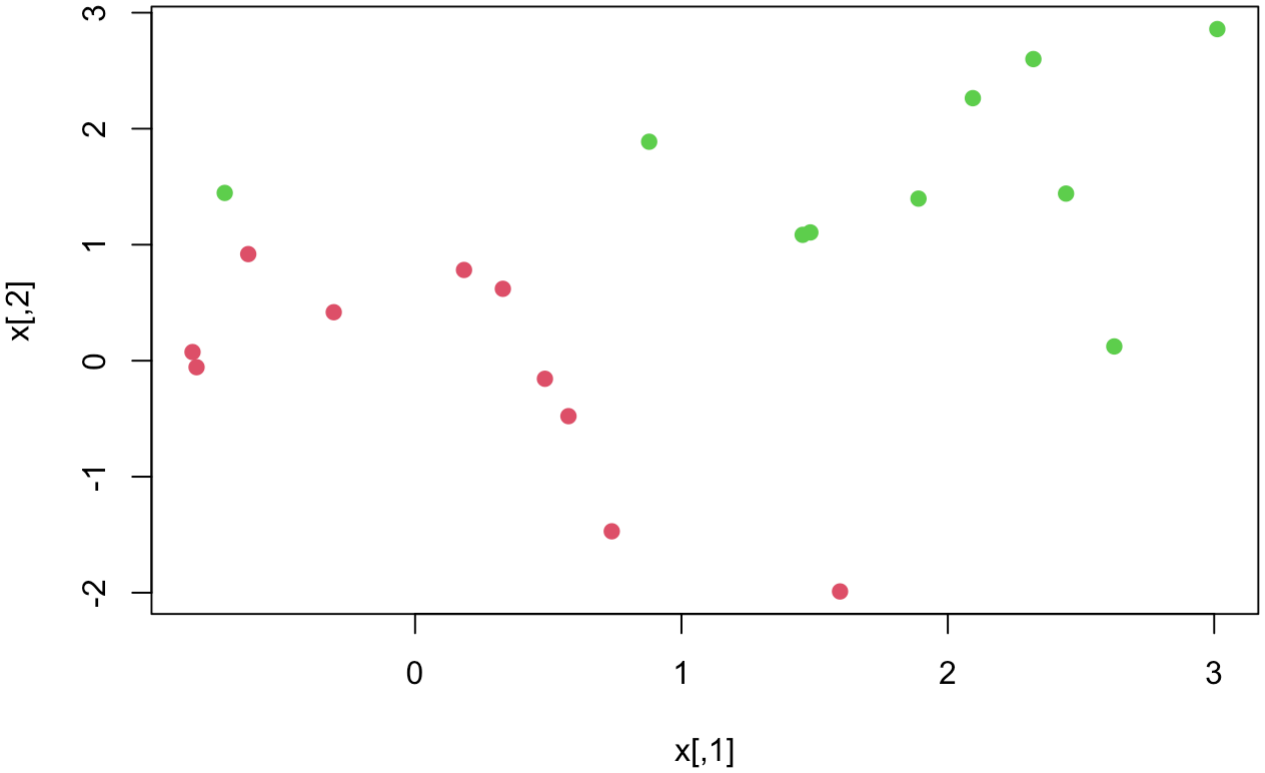
```
xtest <- matrix(rnorm(20 * 2), ncol = 2)
ytest <- sample(c(-1, 1), 20, rep = TRUE)
xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 1
testdat <- data.frame(x = xtest, y = as.factor(ytest))
ypred <- predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
```

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

```
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
```

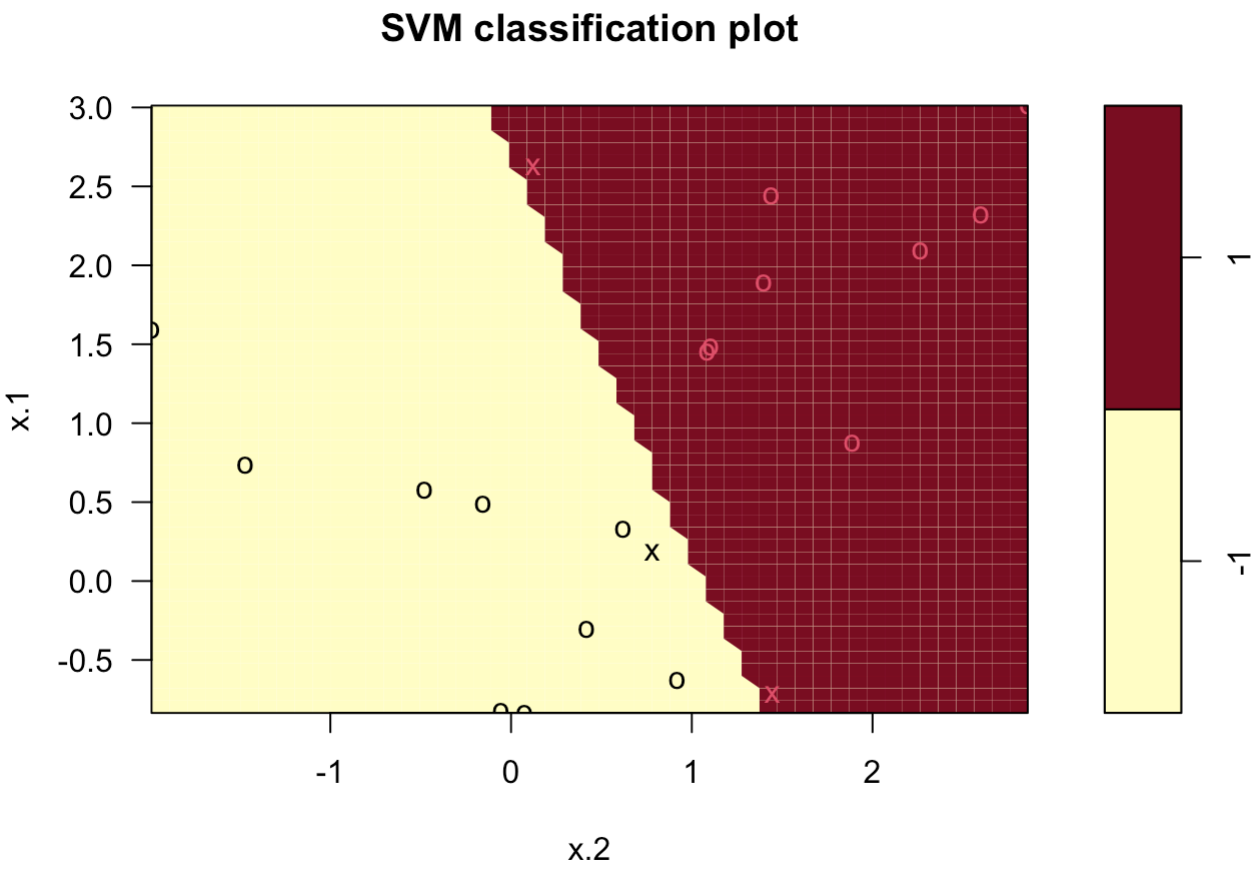
```
x[y == 1, ] <- x[y == 1, ] + 0.5
plot(x, col = (y + 5) / 2, pch = 19)
```




```
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
```

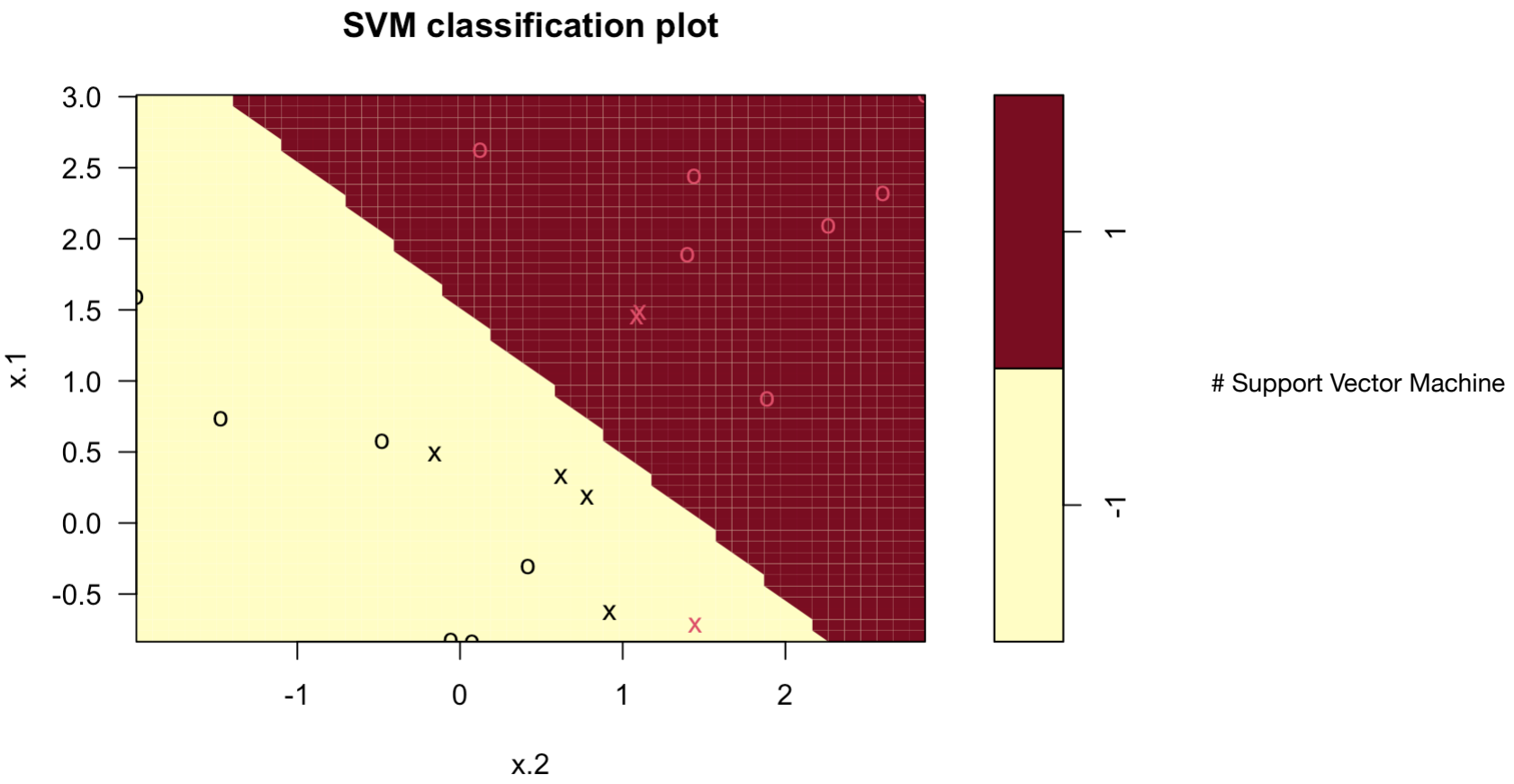
```
plot(svmfit, dat)
```



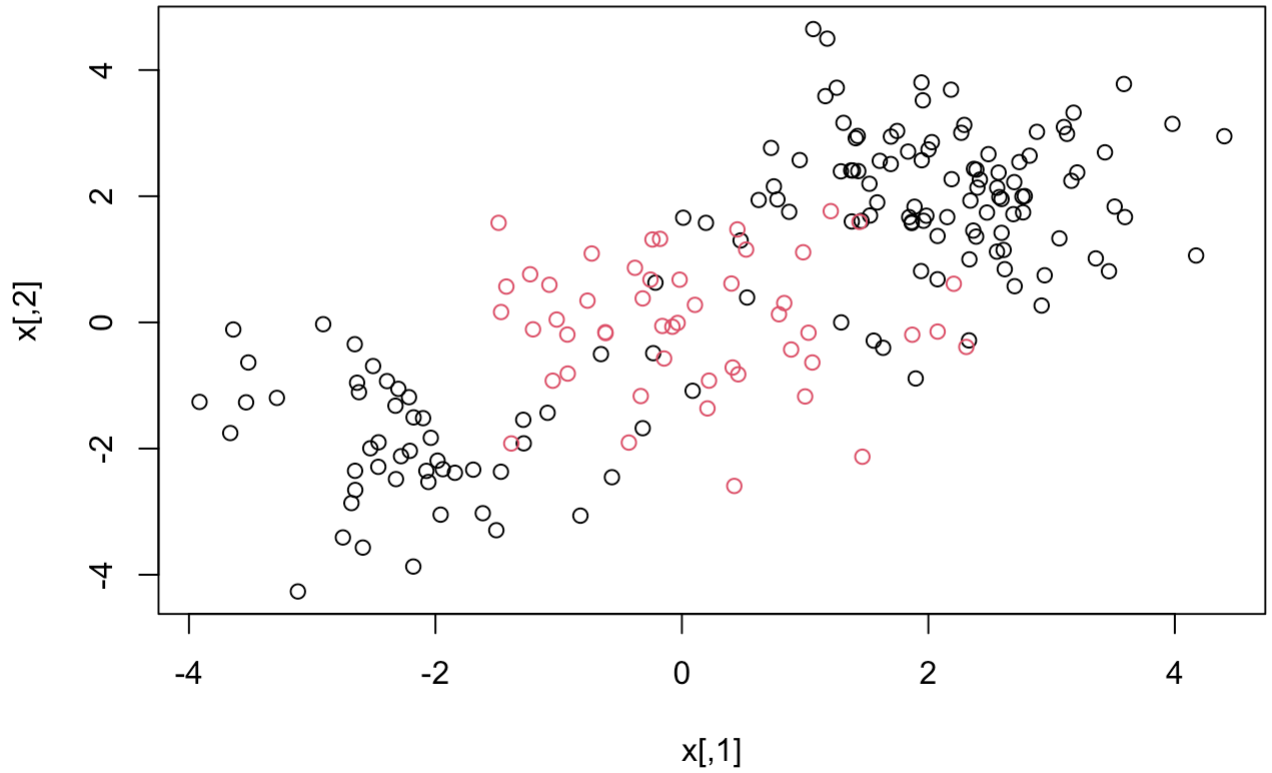

```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 1)
summary(svmfit)
```

```
##
## 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
```

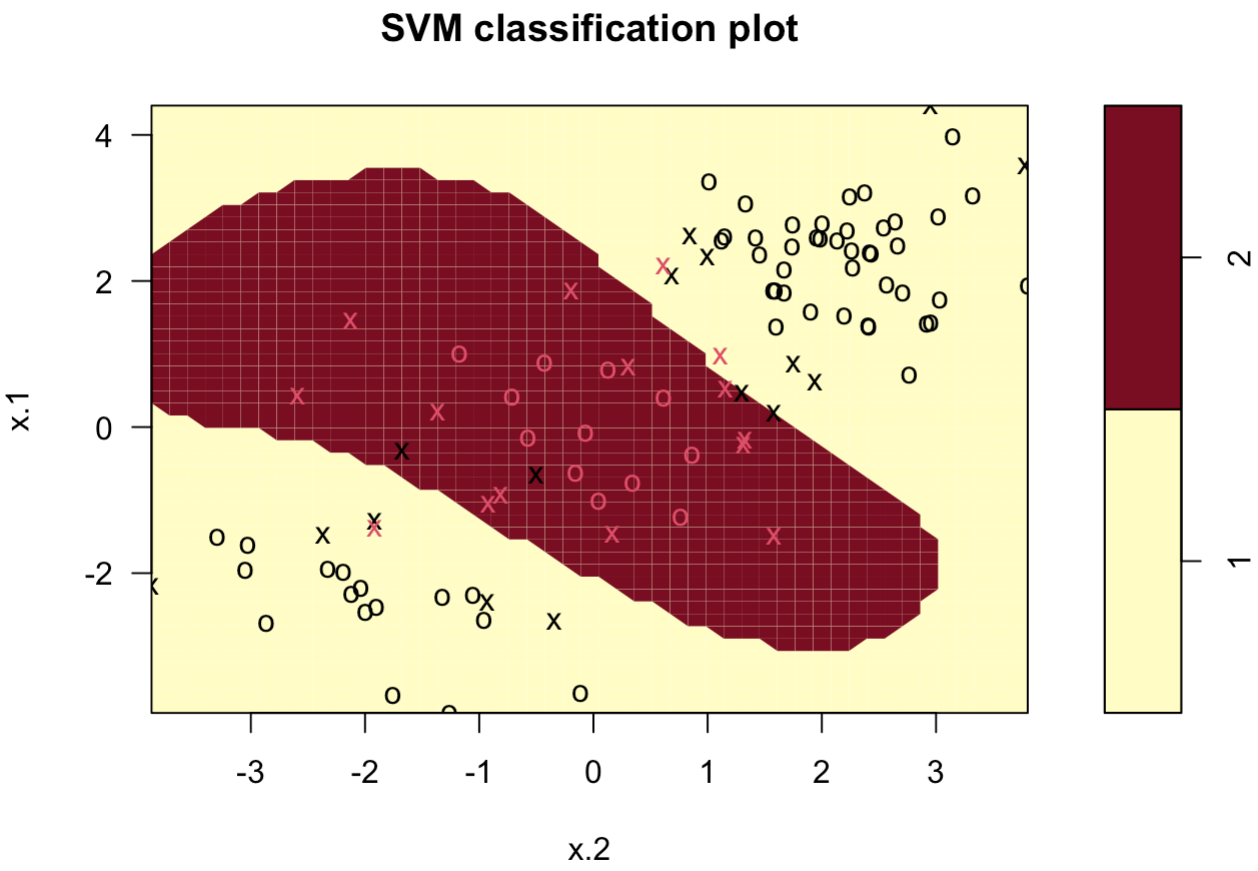
```
plot(svmfit, dat)
```




```
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)
```



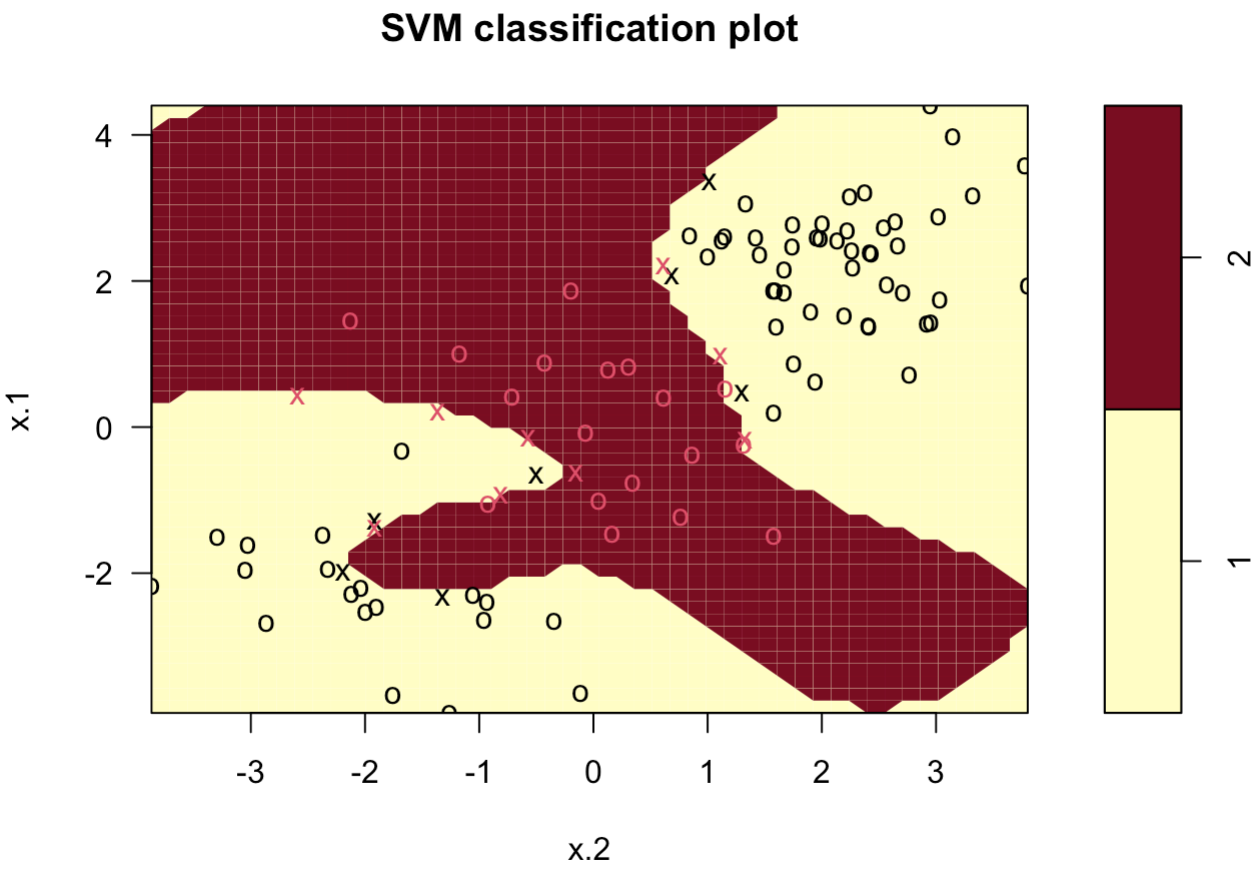

```
train <- sample(200, 100)
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial",
              gamma = 1, cost = 1)
plot(svmfit, dat[train, ])
```




```
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[train, ])
```




```
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
```

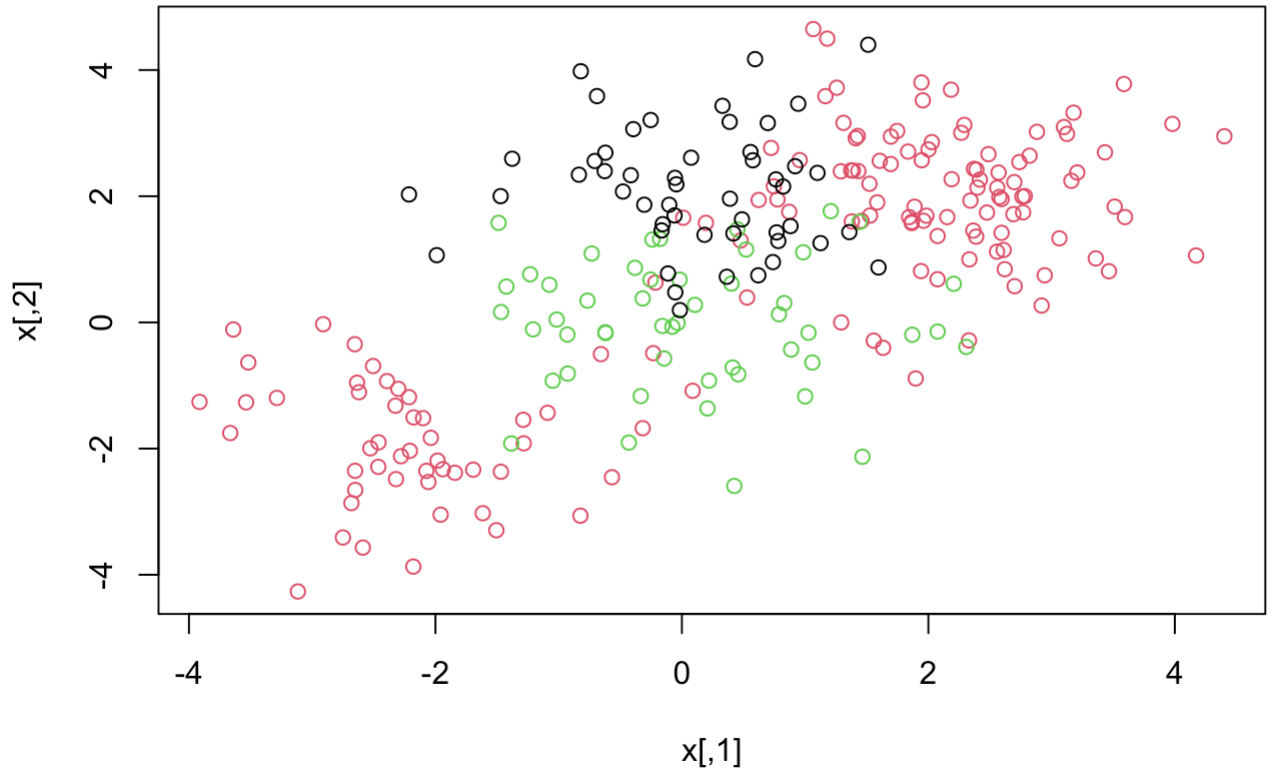
```
## 24 1e+02   4.0  0.13 0.14181365
## 25 1e+03   4.0  0.15 0.13540064
```

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

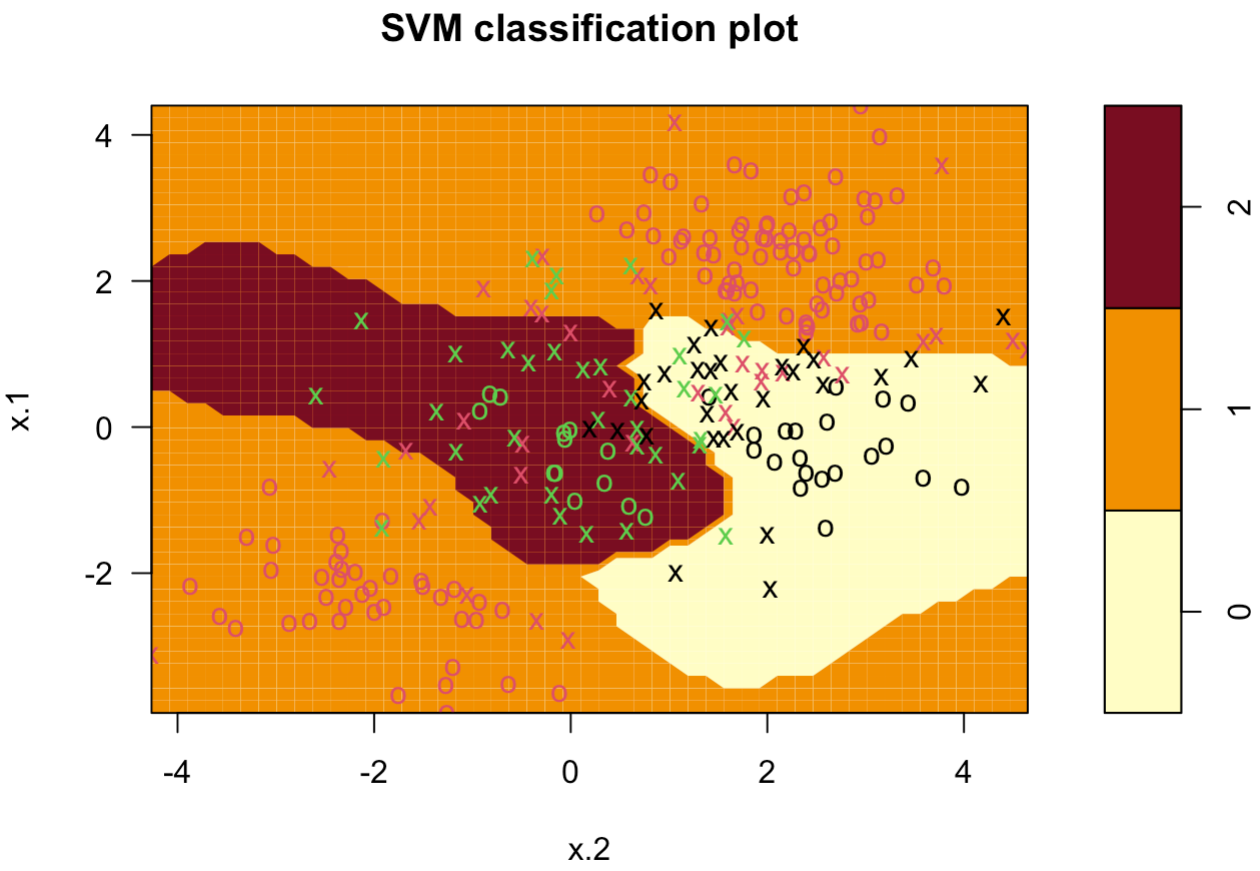
```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

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))
```

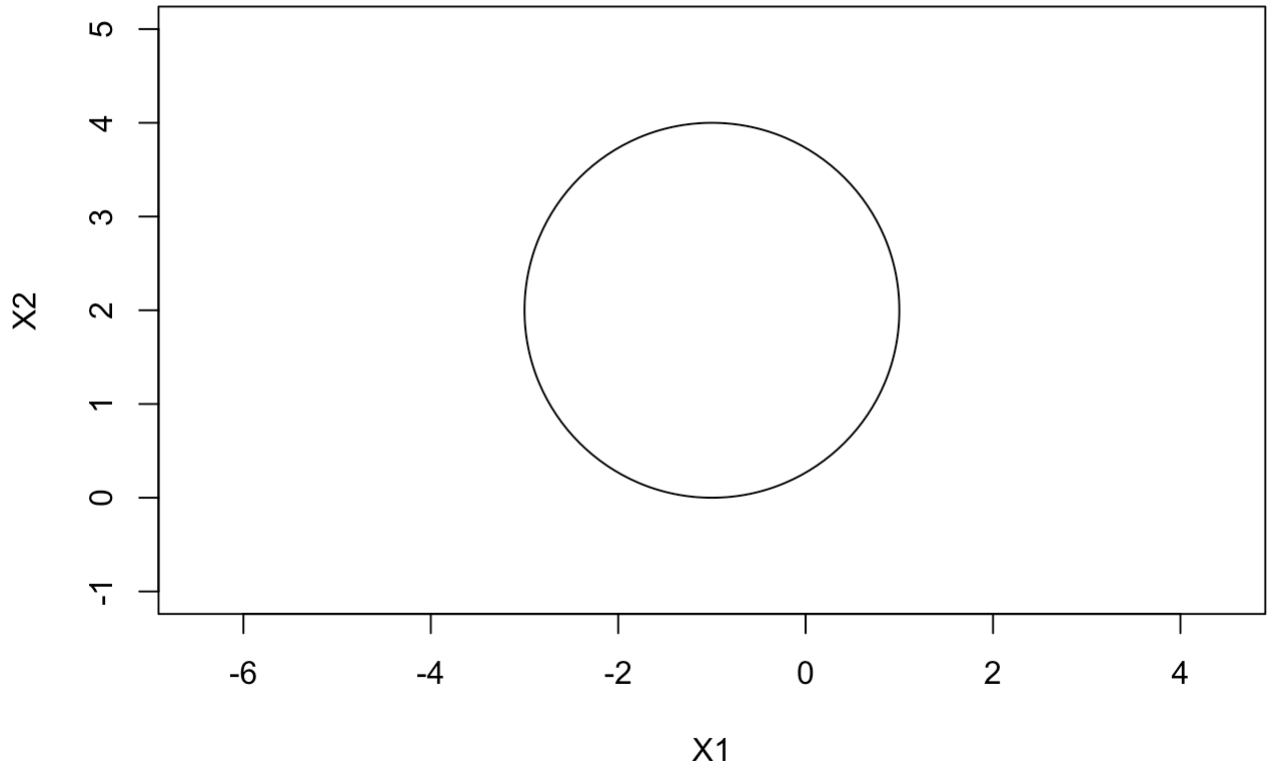



```
svmfit <- svm(y ~ ., data = dat, kernel = "radial",
              cost = 10, gamma = 1)
plot(svmfit, dat)
```



1

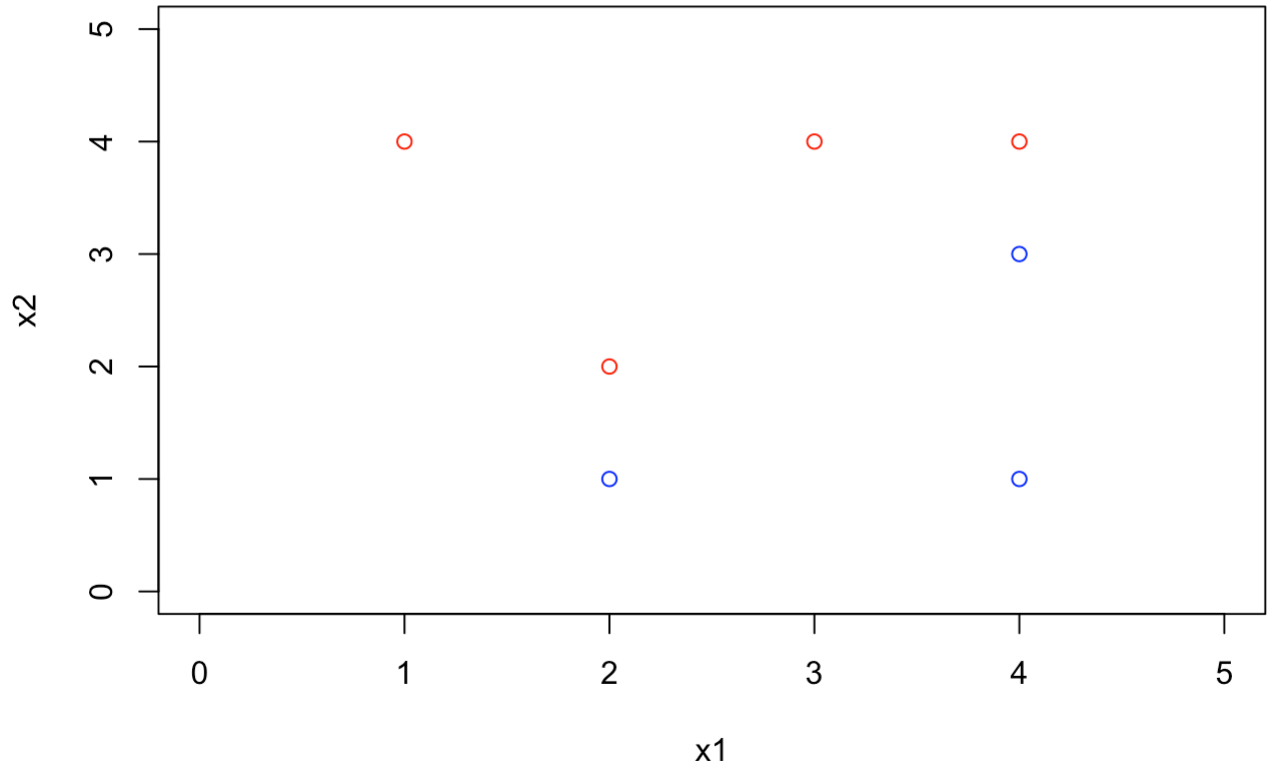
```
plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
```



2

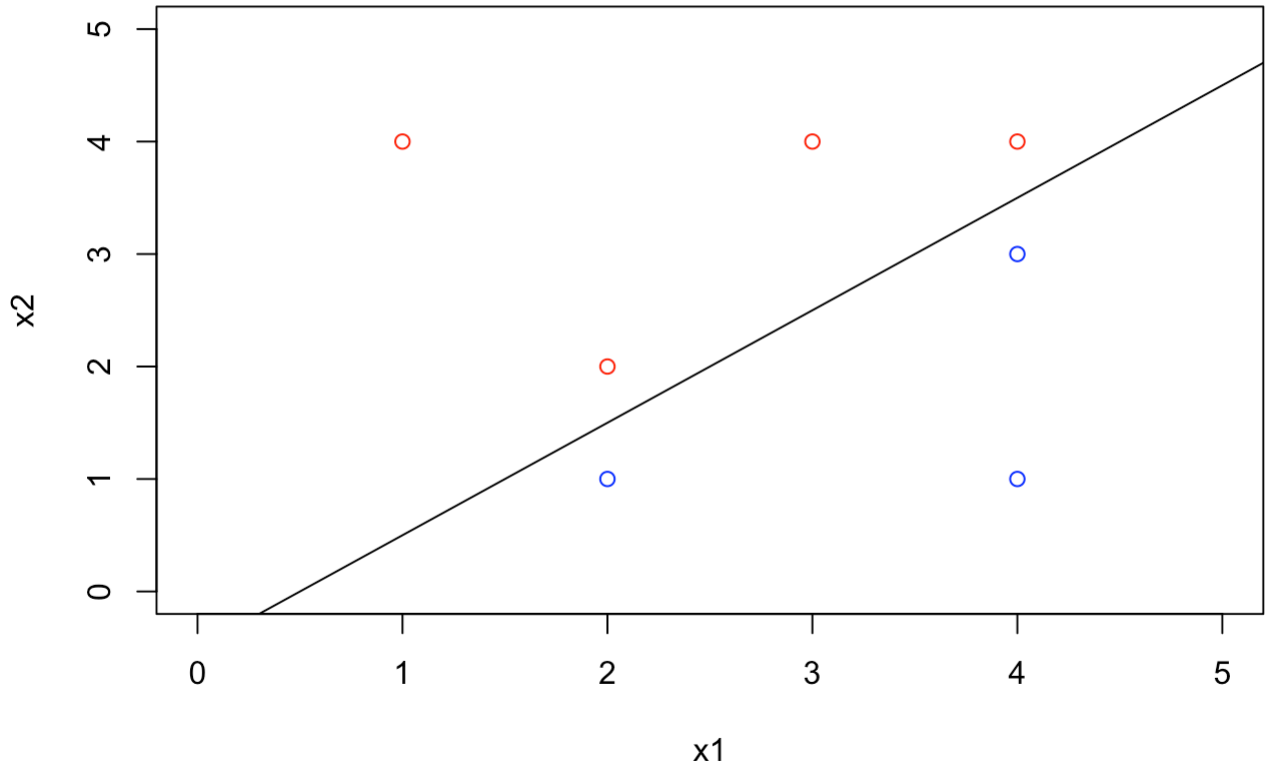
2.1

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
```



2.2

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
```

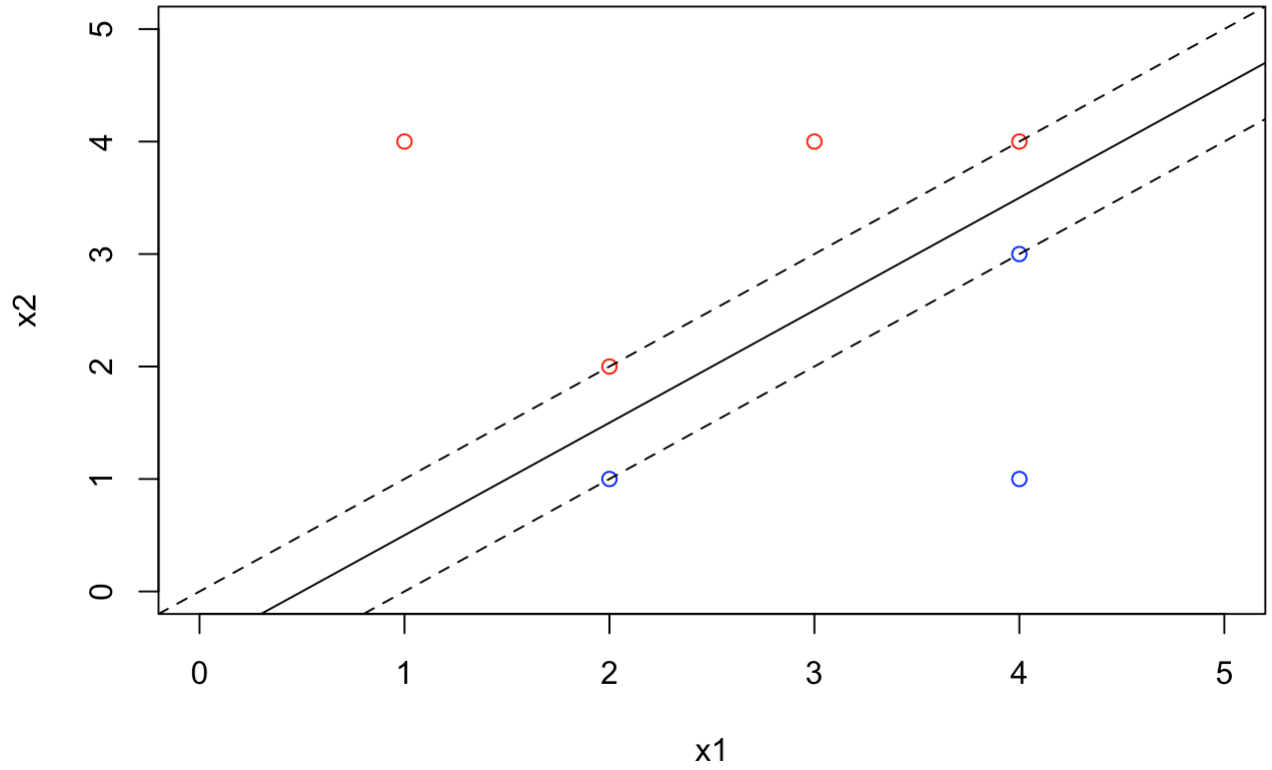

x2
0
1
2
3
4
5
x1
0
1
2
3
4
5

2.3

The classification rule is “Classify to Red if X1−X2−0.5<0, and classify to Blue otherwise.

2.4

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



The margin is here equal to 1/4

2.4

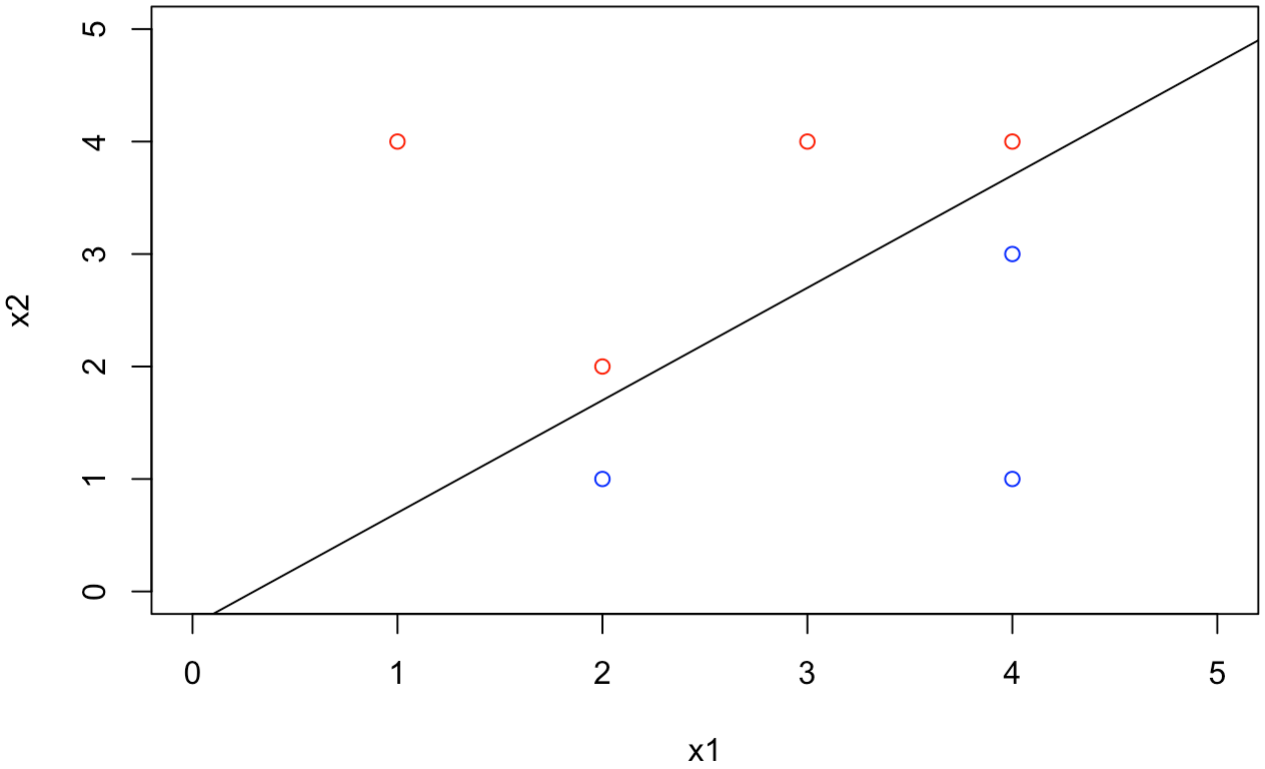
The support vectors are the points (2,1), (2,2), (4,3) and (4,4).

2.5

By examining the plot, it is clear that if we moved the observation (4,1), we would not change the maximal margin hyperplane as it is not a support vector.

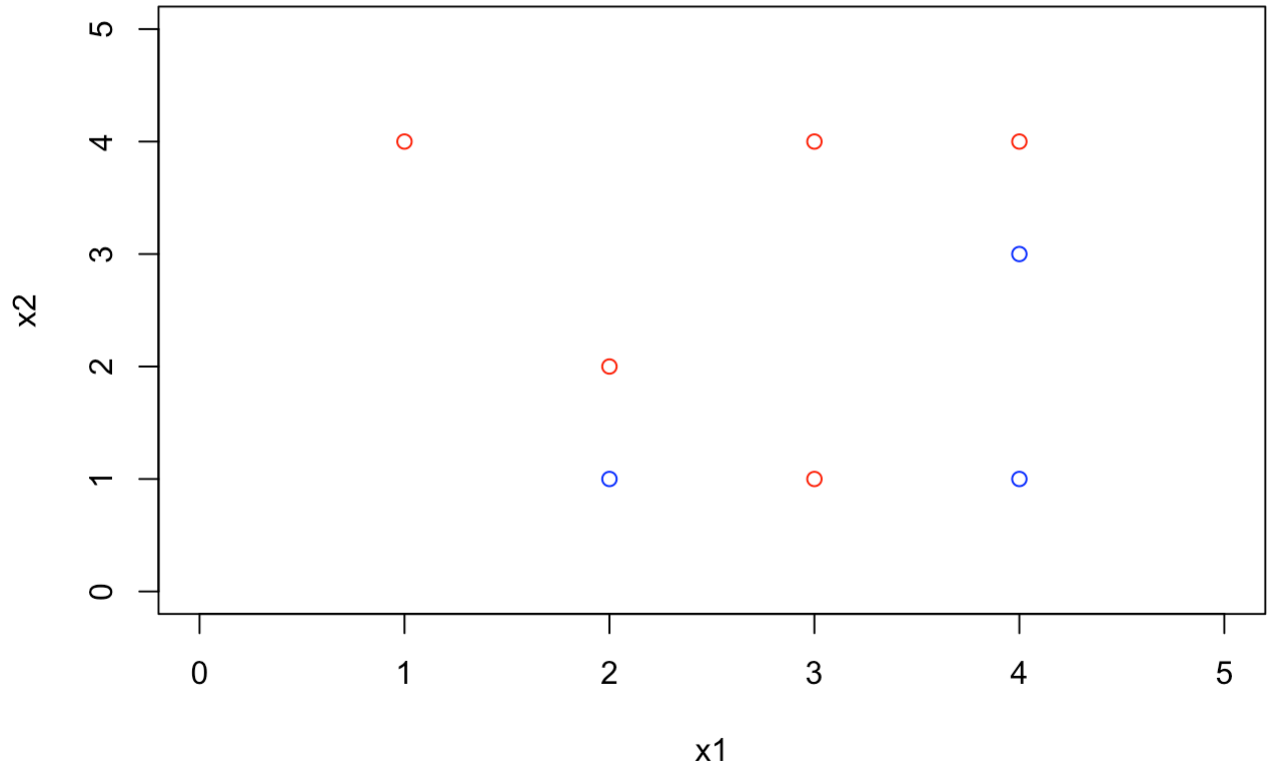
2.6

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.3, 1)
```



2.7

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
points(c(3), c(1), col = c("red"))
```



When the red point (3,1) is added to the plot, the two classes are obviously not separable by a hyperplane anymore.

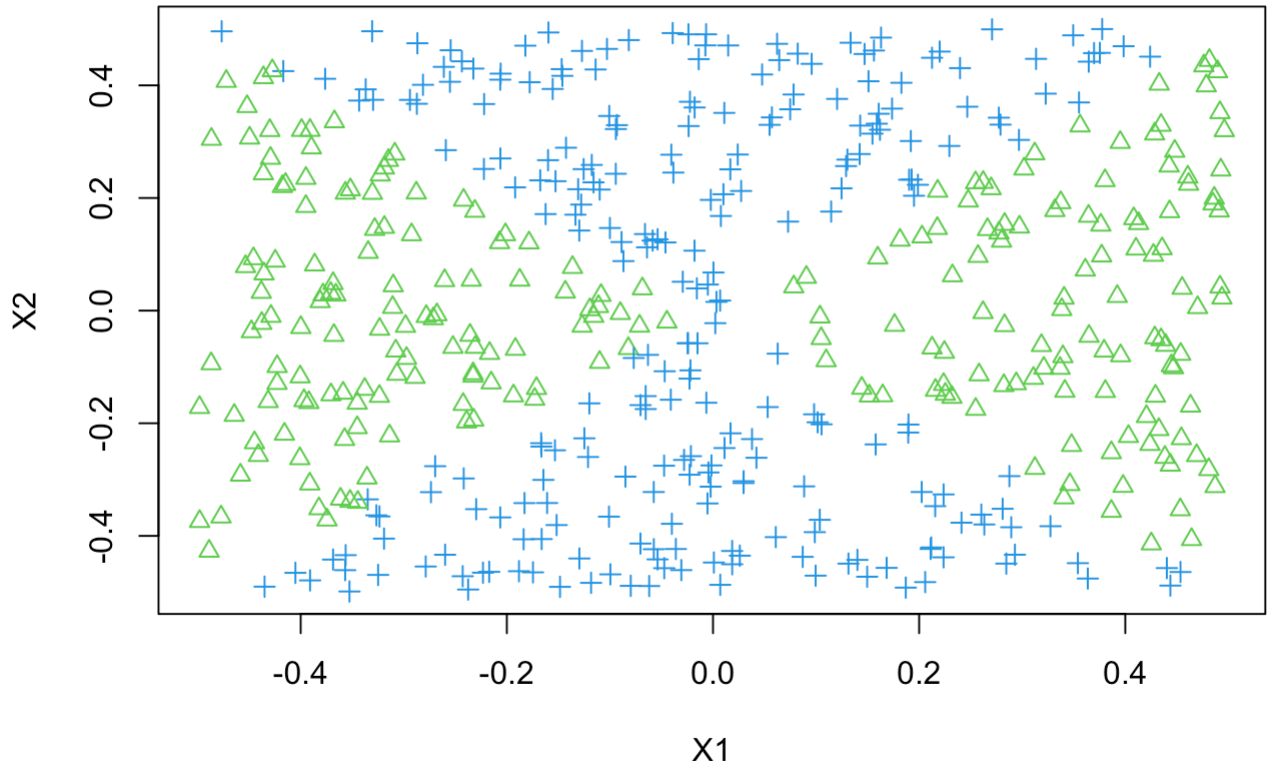
3

3.1

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

3.2

```
plot(x1, x2, xlab = "X1", ylab = "X2", col = (4 - y), pch = (3 - y))
```



3.3

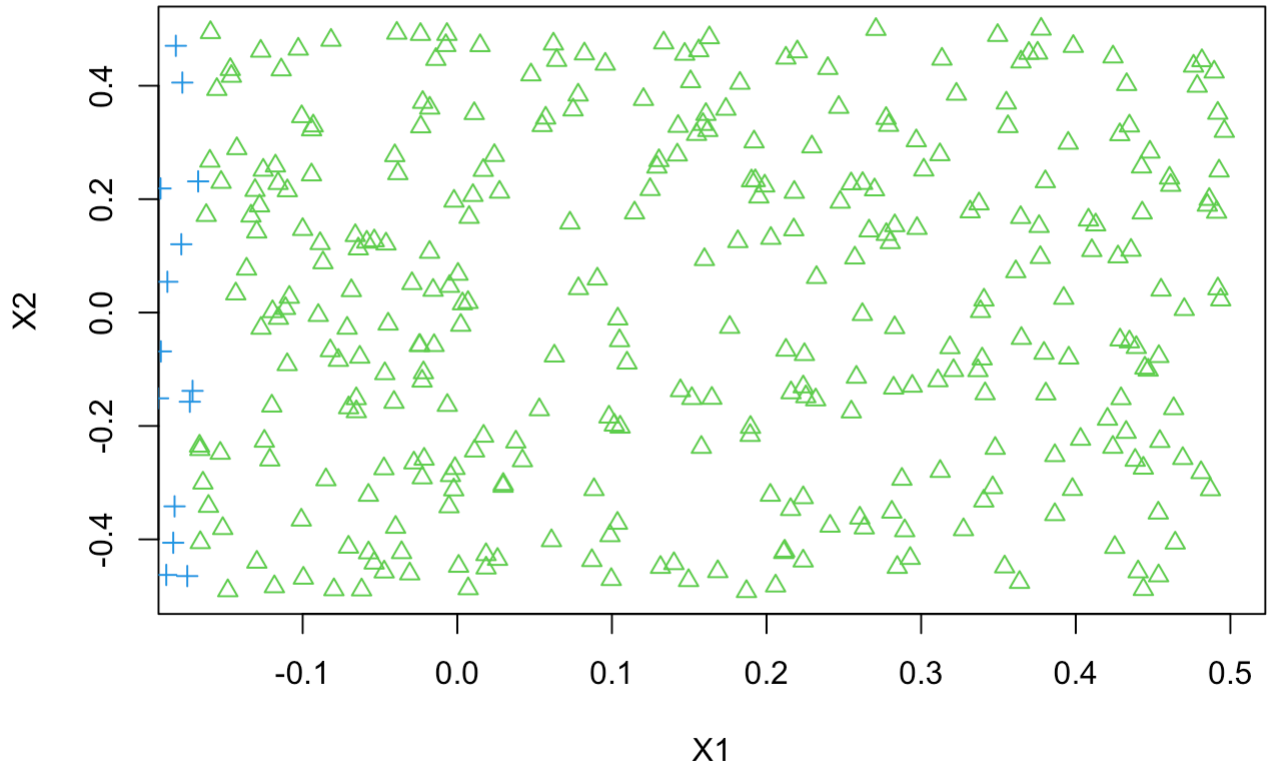
```
logit.fit <- glm(y ~ x1 + x2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## x1           0.196199   0.316864   0.619    0.536
## x2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

None of the variables are statistically significants.

3.4

```
data <- data.frame(x1 = x1, x2 = x2, y = y)
probs <- predict(logit.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```



The decision boundary is obviously linear.

3.5

```
logitnl.fit <- glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
```

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

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

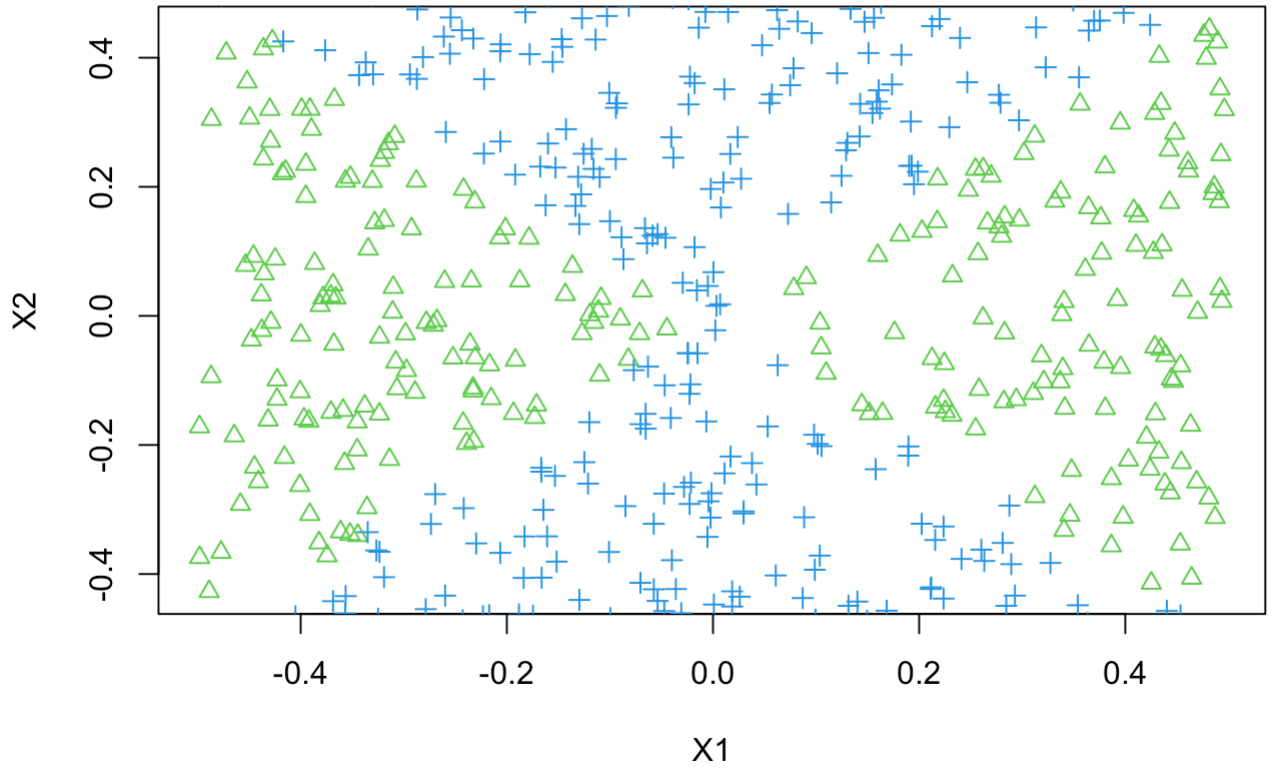
```
summary(logitnl.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -102.2     4302.0  -0.024    0.981
## poly(x1, 2)1   2715.3   141109.5   0.019    0.985
## poly(x1, 2)2  27218.5   842987.2   0.032    0.974
## poly(x2, 2)1   -279.7    97160.4  -0.003    0.998
## poly(x2, 2)2 -28693.0   875451.3  -0.033    0.974
## I(x1 * x2)     -206.4    41802.8  -0.005    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
## Residual deviance: 3.5810e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

Here again, none of the variables are statistically significants.

3.6

```
probs <- predict(logitnl.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```

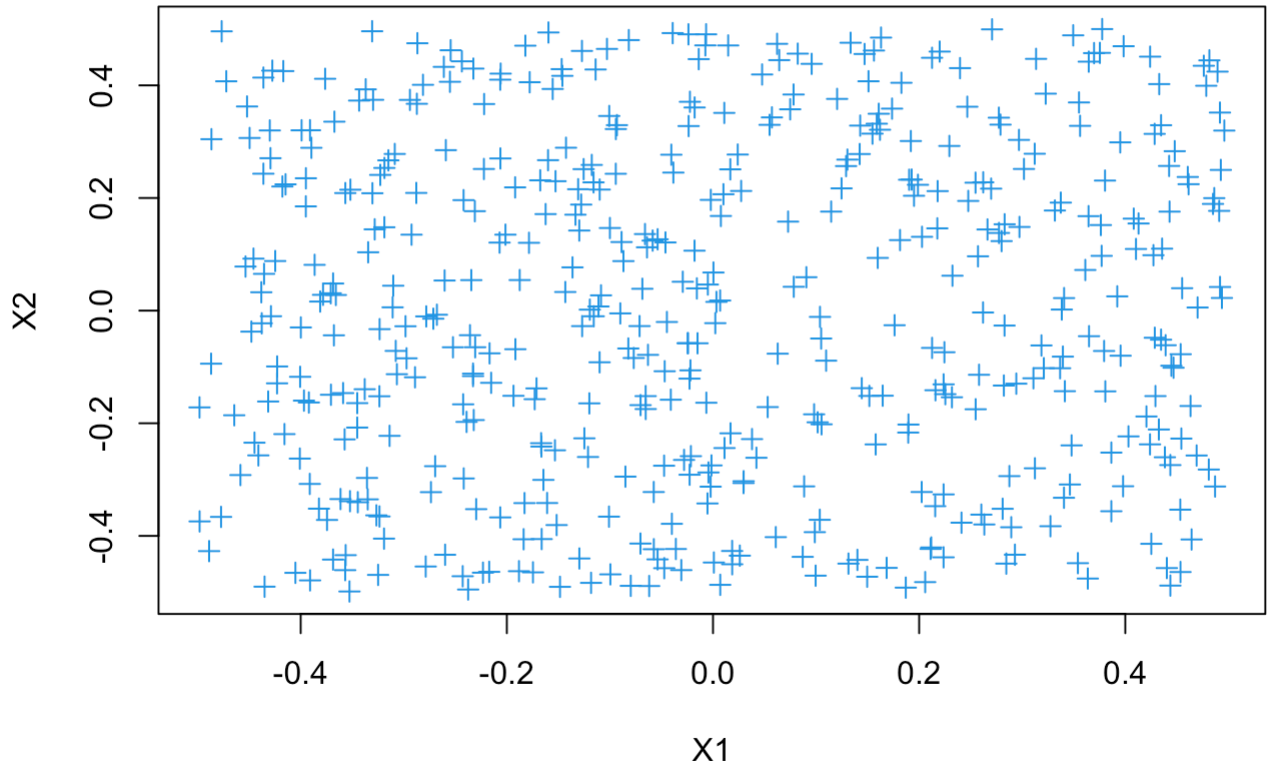


The non-linear decision boundary is surprisingly very similar to the true decision boundary.

3.7

```
data$y <- as.factor(data$y)
svm.fit <- svm(y ~ x1 + x2, data, kernel = "linear", cost = 0.01)
preds <- predict(svm.fit, data)
```

```
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X2")
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```

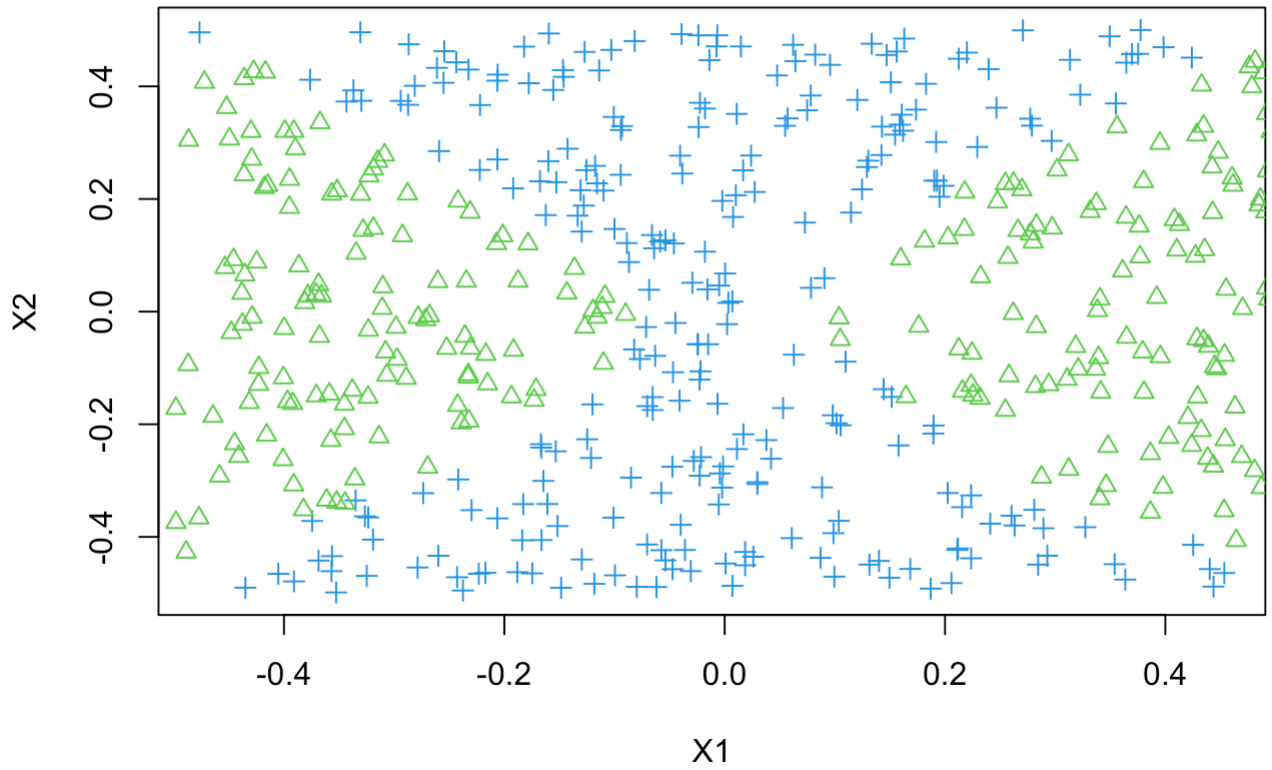


This support vector classifier (even with low cost) classifies all points to a single class.

3.8

```
data$y <- as.factor(data$y)
```

```
svmnl.fit <- svm(y ~ x1 + x2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X2")
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```



Here again, the non-linear decision boundary is surprisingly very similar to the true decision boundary.

3.9

We may conclude that SVM with non-linear kernel and logistic regression with interaction terms are equally very powerful for finding non-linear decision boundaries. Also, SVM with linear kernel and logistic regression without any interaction term are very bad when it comes to finding non-linear decision boundaries. However, one argument in favor of SVM is that it requires some manual tuning to find the right interaction terms when using logistic regression, although when using SVM we only need to tune gamma.

4

4.1

```
library(ISLR)
attach(OJ)
set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

4.2

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

```
##
## 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:  435
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier creates 432 support vectors out of 800 training points. Out of these, 217 belong to level MM and remaining 215 belong to level CH.

4.3

```
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 420  65
##   MM  75 240
```

```
(78 + 55) / (439 + 228 + 78 + 55)
```

```
## [1] 0.16625
```

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```
(31 + 18) / (141 + 80 + 31 + 18)
```

```
## [1] 0.1814815
```

The training error rate is 16.6% and test error rate is about 18.1%

4.4

```
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2, 1, by =
0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  1.778279
##
## - best performance: 0.1675
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17625 0.04059026
## 2   0.01778279 0.17625 0.04348132
## 3   0.03162278 0.17125 0.04604120
## 4   0.05623413 0.17000 0.04005205
## 5   0.10000000 0.17125 0.04168749
## 6   0.17782794 0.17000 0.04090979
## 7   0.31622777 0.17125 0.04411554
## 8   0.56234133 0.17125 0.04084609
## 9   1.00000000 0.17000 0.04090979
## 10  1.77827941 0.16750 0.03782269
## 11  3.16227766 0.16750 0.03782269
## 12  5.62341325 0.16750 0.03545341
## 13 10.00000000 0.17000 0.03736085
```

We may see that the optimal cost is 0.1.

4.5

```
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$best.parameter$cost)
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 423  62
##   MM  69 246
```

```
(71 + 56) / (438 + 235 + 71 + 56)
```

```
## [1] 0.15875
```

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 156  12
##   MM  29  73
```

```
(32 + 19) / (140 + 79 + 32 + 19)
```

```
## [1] 0.1888889
```

We may see that, with the best cost, the training error rate is now 15.8% and the test error rate is 18.8%.

4.6

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```
(28 + 18) / (141 + 83 + 28 + 18)
```

```
## [1] 0.1703704
```

Radial kernel with default gamma creates 379 support vectors, out of which, 188 belong to level CH and remaining 191 belong to level MM. The classifier has a training error of 14.5% and a test error of 17% which is a slight improvement over linear kernel. We now use cross validation to find optimal cost.

```
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1725
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39375 0.03240906
## 2   0.01778279 0.39375 0.03240906
## 3   0.03162278 0.34750 0.05552777
## 4   0.05623413 0.19250 0.03016160
## 5   0.10000000 0.19500 0.03782269
## 6   0.17782794 0.18000 0.04048319
## 7   0.31622777 0.17250 0.03809710
## 8   0.56234133 0.17500 0.04124790
## 9   1.00000000 0.17250 0.03162278
## 10  1.77827941 0.17750 0.03717451
## 11  3.16227766 0.18375 0.03438447
## 12  5.62341325 0.18500 0.03717451
## 13 10.00000000 0.18750 0.03173239
```

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tune.out$best.parameter$cost)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial", cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```
(28 + 18) / (141 + 83 + 28 + 18)
```

```
## [1] 0.1703704
```

Tuning does not reduce train and test error rates as we already used the optimal cost of 1.

4.7

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  447
##
##  ( 225 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 449  36
##   MM 110 205
```

```
(105 + 33) / (461 + 201 + 105 + 33)
```

```
## [1] 0.1725
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 153  15
##   MM  45  57
```

```
(41 + 10) / (149 + 70 + 41 + 10)
```

```
## [1] 0.1888889
```

Polynomial kernel with default gamma creates 454 support vectors, out of which, 224 belong to level CH and remaining 230 belong to level MM. The classifier has a training error of 17.2% and a test error of 18.8% which is no improvement over linear kernel. We now use cross validation to find optimal cost.

```
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, ranges = list(cost = 10^s
eq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.162278
##
## - best performance: 0.18
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39000 0.03670453
## 2   0.01778279 0.37000 0.03395258
## 3   0.03162278 0.36375 0.03197764
## 4   0.05623413 0.34500 0.03291403
## 5   0.10000000 0.32125 0.03866254
## 6   0.17782794 0.24750 0.03322900
## 7   0.31622777 0.20250 0.04073969
## 8   0.56234133 0.20250 0.03670453
## 9   1.00000000 0.19625 0.03910900
## 10  1.77827941 0.19125 0.03586723
## 11  3.16227766 0.18000 0.04005205
## 12  5.62341325 0.18000 0.04133199
## 13 10.00000000 0.18125 0.03830162
```

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cost = tune.out$best.parameter
$cost)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2, cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  3.162278
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  385
##
##  ( 197 188 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 451  34
##   MM  90 225
```

```
(72 + 44) / (450 + 234 + 72 + 44)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 154  14
##   MM  41  61
```

```
(31 + 19) / (140 + 80 + 31 + 19)
```

```
## [1] 0.1851852
```

Tuning reduce train and test error rates.

4.8

Overall, radial basis kernel seems to be producing minimum misclassification error on both train and test data.