DSA 8070 R Session 11: Classification

Whitney

Contents

Linear Discriminant Analysis, Logistic Regression, and Quadratic Discriminant Analysis

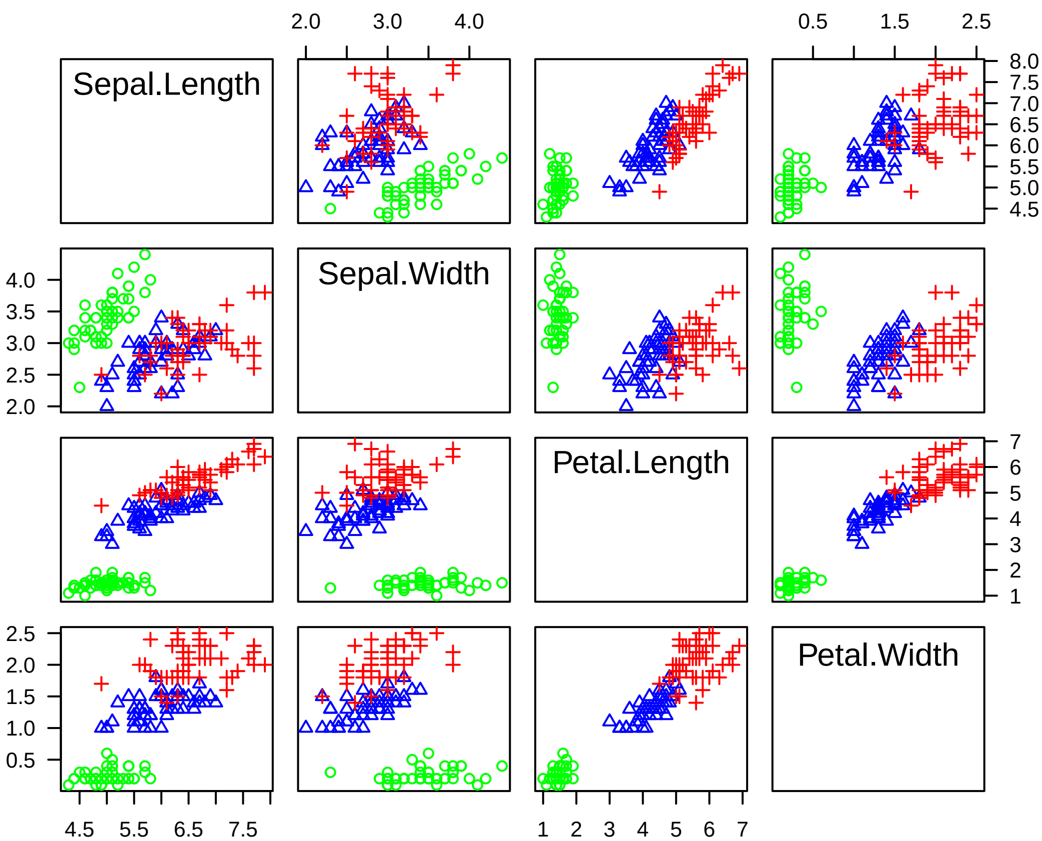
Iris data

```r
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

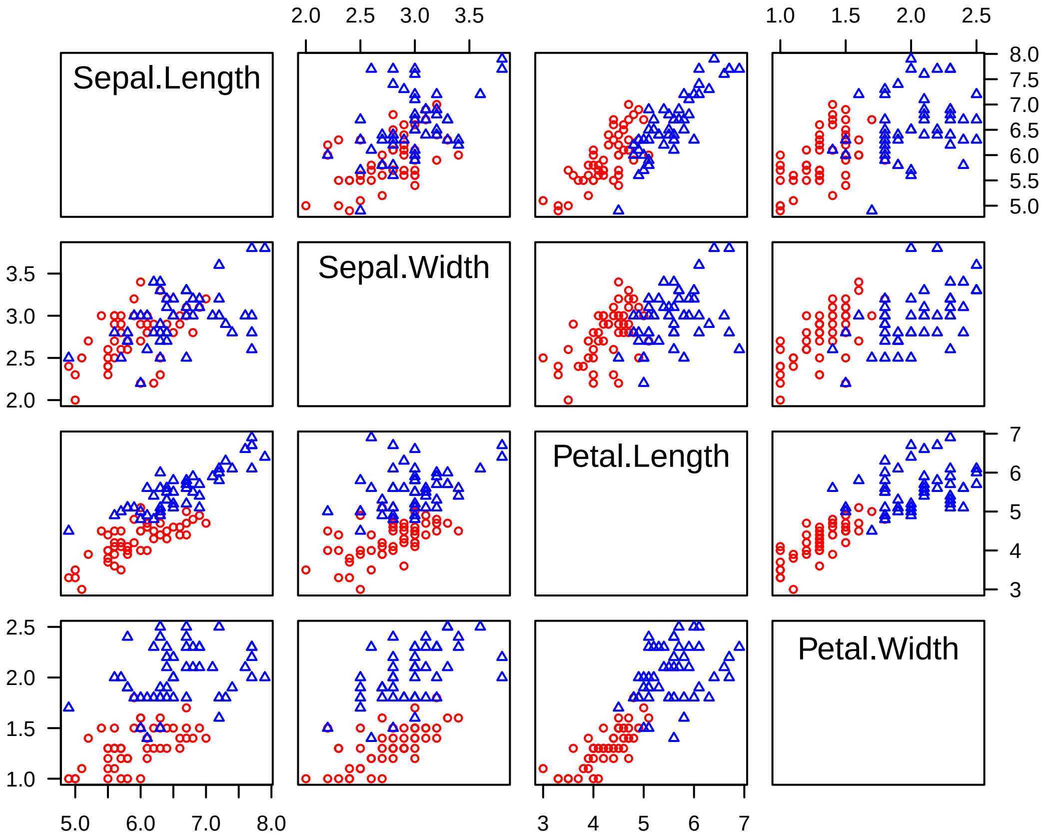
```
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
attach(iris)
library(car)
par(las = 1)
scatterplotMatrix(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width | Species,
                  col = c("green", "blue", "red"), diagonal = F,
                  smooth = F, regLine = F, legend = F)
```



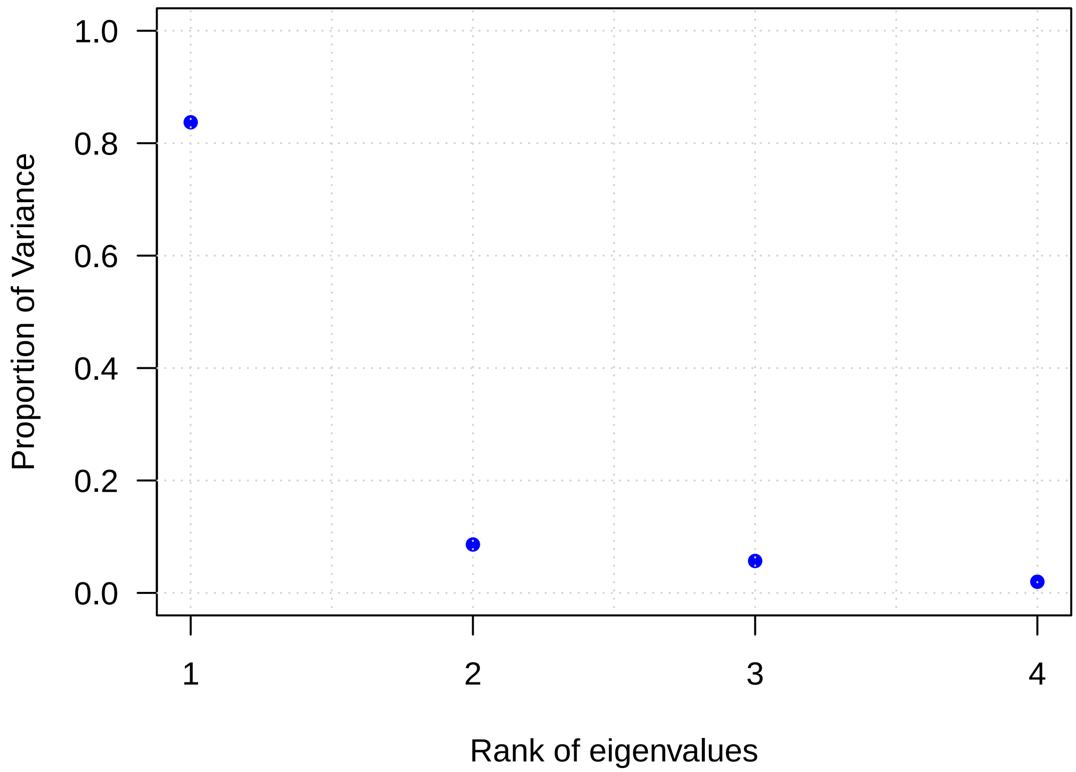
Binary classification

```
irisv = iris[51:150,]
irisv$Species <- factor(irisv$Species)
attach(irisv)
par(las = 1)
scatterplotMatrix(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width | Species,
                  col = c("red", "blue"), diagonal = F,
                  smooth = F, regLine = F, legend = F, cex = 0.75)
```

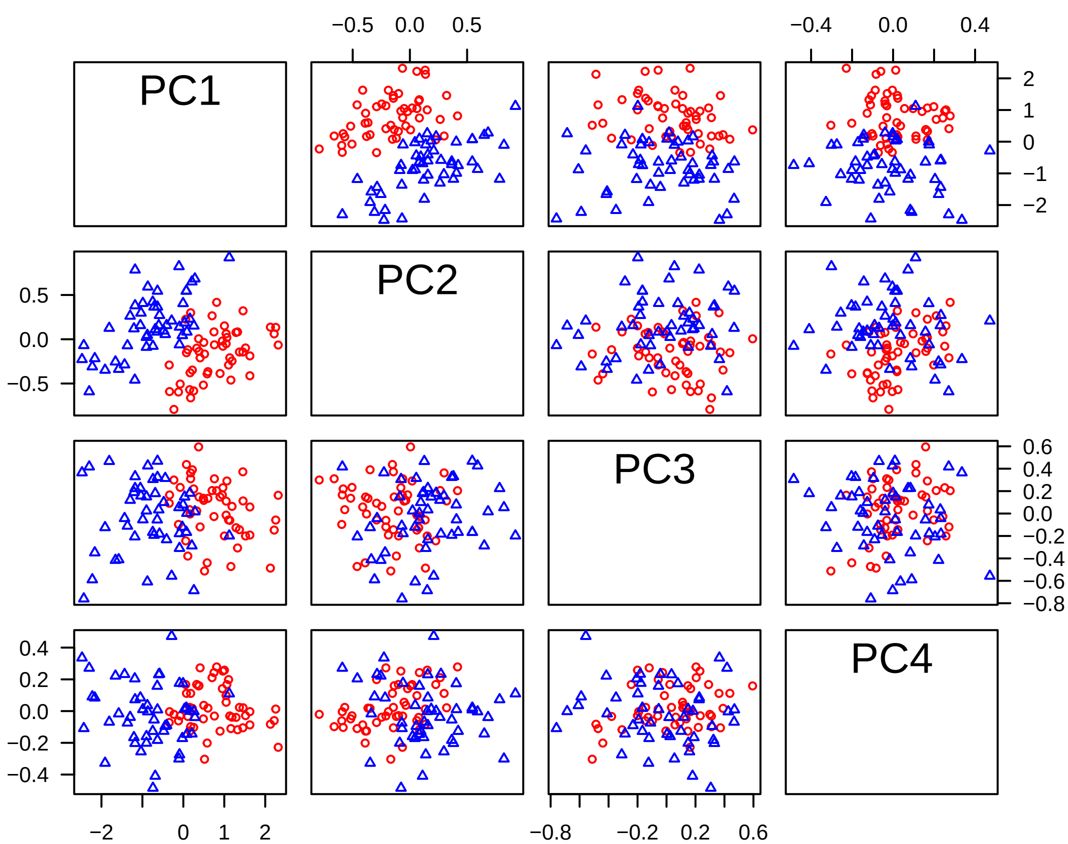


PCA

```r
pca <- prcomp(irisv[, 1:4])
Z <- pca$x; lambda <- pca$sdev^2
par(las = 1)
plot(1:4, lambda / sum(lambda), xaxt = "n", las = 1, xlab = "Rank of eigenvalues",
     ylab = "Proportion of Variance", pch = 16, col = "blue", cex = 1, ylim = c(0, 1))
grid(); axis(1, at = 1:4)
```




```
scatterplotMatrix(~ Z | Species, col = c("red", "blue"), diagonal = F,smooth = F,
                  regLine = F, legend = F, cex = 0.75)
```

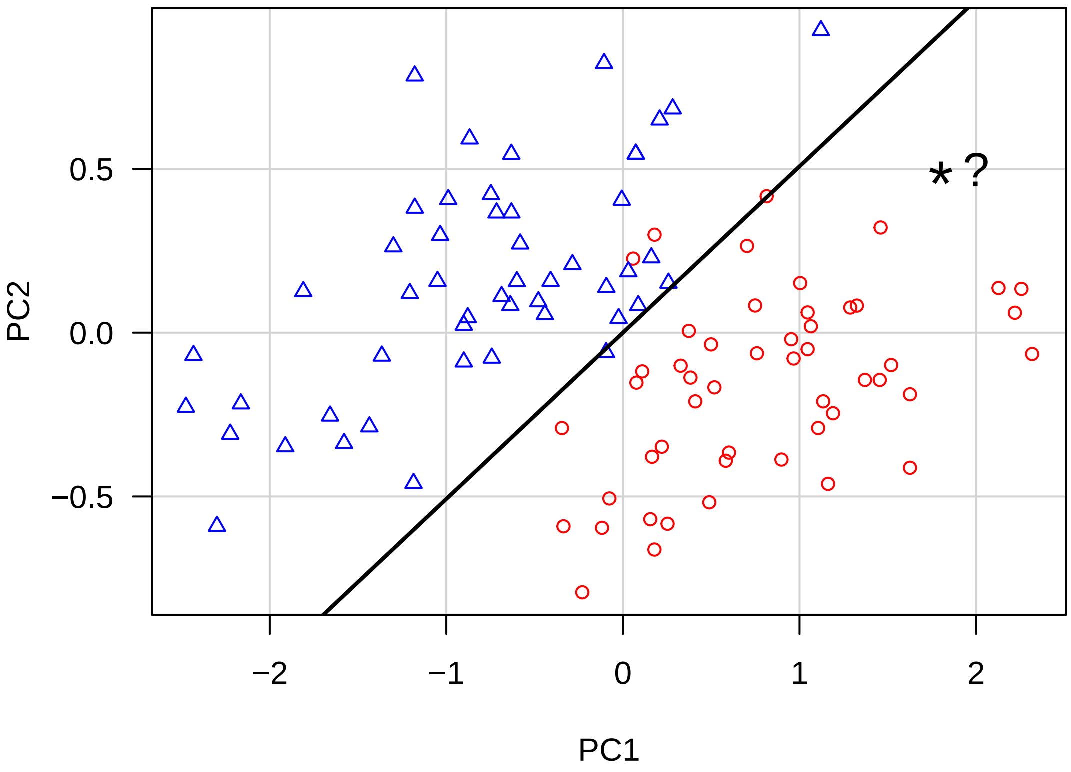


Linear Discriminant Analysis (LDA)

```
library(MASS)
par(las = 1)
scatterplot(PC2 ~ PC1 | Species , Z, smooth = F, regLine = F, legend = F, cex = 0.85,
            col = c("red", "blue"))
fit <- lda(Species ~ Z[, 1:2])
fit # show results
```

```
## Call:
## lda(Species ~ Z[, 1:2])
##
## Prior probabilities of groups:
## versicolor  virginica
##        0.5        0.5
##
## Group means:
##            Z[, 1:2]PC1 Z[, 1:2]PC2
## versicolor   0.7930189  -0.1607571
## virginica   -0.7930189   0.1607571
##
## Coefficients of linear discriminants:
##                    LD1
## Z[, 1:2]PC1 -1.553249
## Z[, 1:2]PC2  3.060560
```

```
abline(0, -fit$scaling[1] / fit$scaling[2], pch = 5, lwd = 2)
points(2, 0.5, pch = "?", cex = 1.5)
points(1.8, 0.5, pch = "*", cex = 2)
```

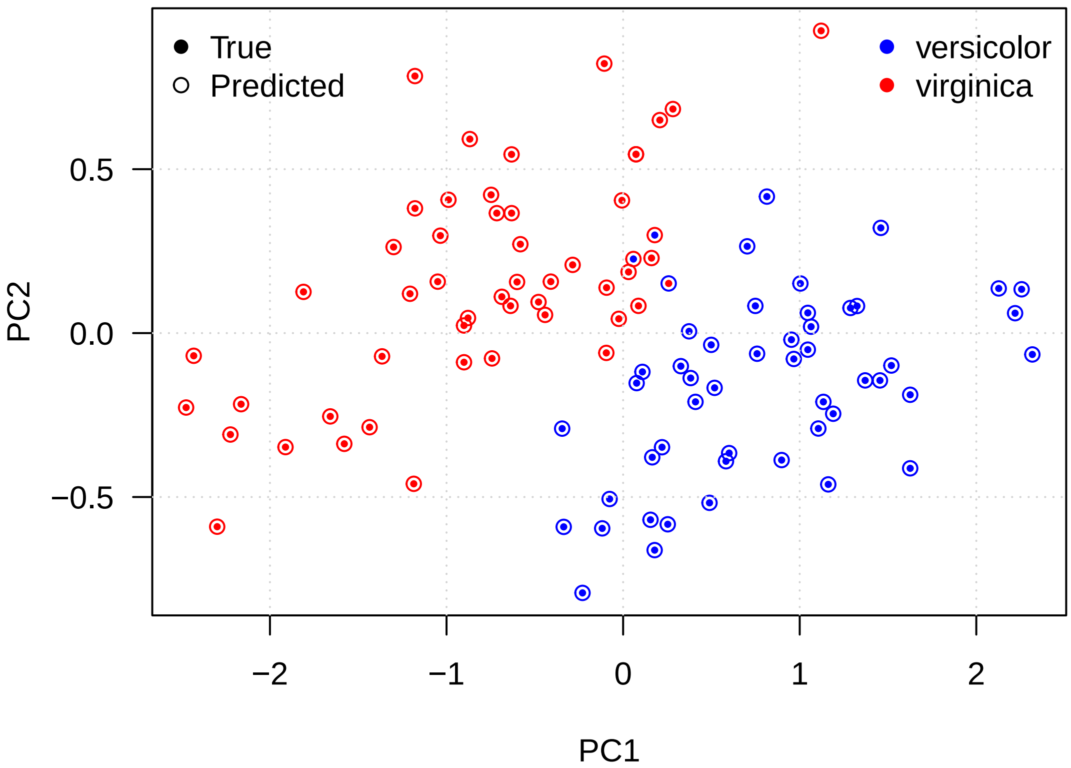


Logistic Regression

```
logfit <- glm(irisv$Species ~ Z[, 1:2], family = binomial)
logpred <- predict(logfit, type = "response")
predCol <- ifelse(logpred <= 0.5, "blue", "red")
Col <- rep(c("blue", "red"), each = 50)
```

Summarize the Result

```
plot(Z[, 1:2], col = predCol, pch = 1, las = 1)
points(Z[, 1:2], col = Col, pch = 16, cex = 0.5)
grid()
legend("topleft", legend = c("True", "Predicted"), pch = c(16, 1), bty = "n")
legend("topright", legend = c("versicolor", "virginica"),
       col = c("blue", "red"), pch = 16, bty = "n")
```




```
logisticPred <- ifelse(logpred <= 0.5, "versicolor", "virginica")
table(irisv$Species, logisticPred)
```

```
##             logisticPred
##              versicolor virginica
##   versicolor         48         2
##   virginica           1        49
```

LDA vs. QDA

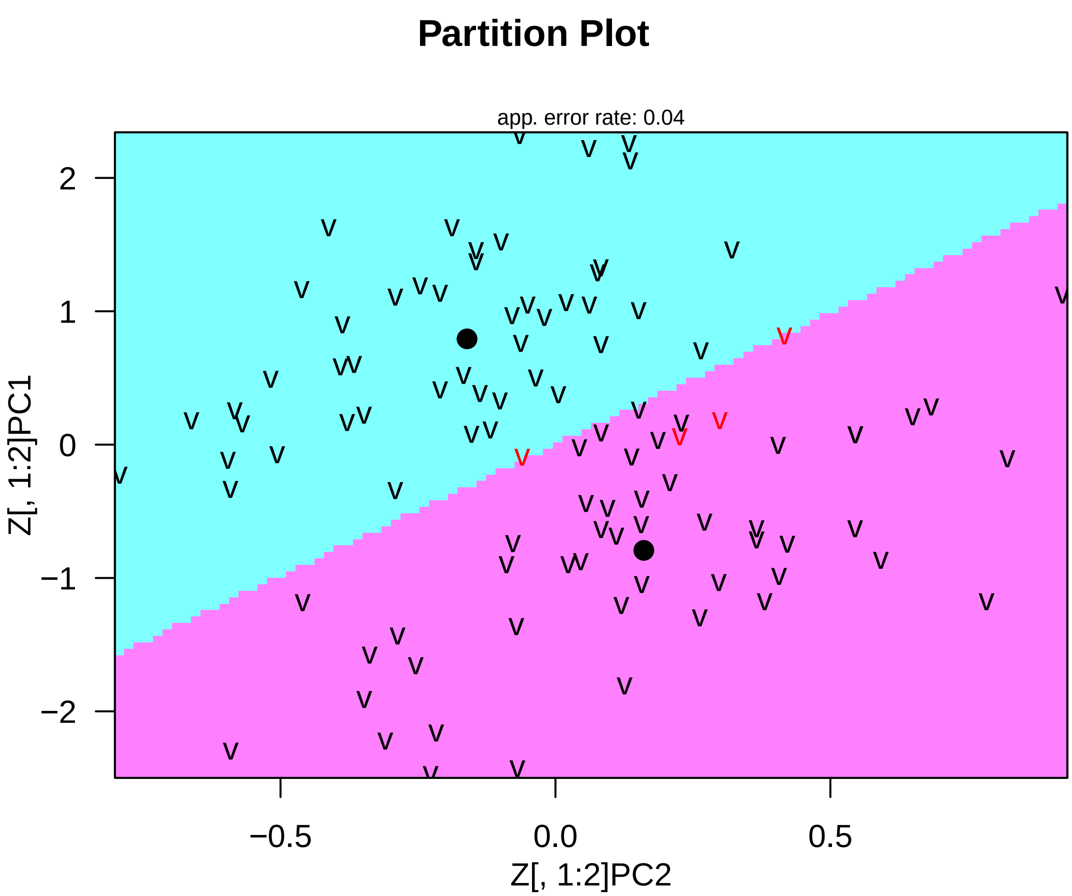
```
#treat data as matrix
z = as.matrix(Z)
# LDA vs. QDA
lda <- lda(irisv$Species ~ Z[, 1:2])
qda <- qda(irisv$Species ~ Z[, 1:2])

fit.LDA = predict(lda)$class
table(irisv$Species, fit.LDA)
```

```
##             fit.LDA
##              versicolor virginica
##   versicolor         47         3
##   virginica           1        49
```

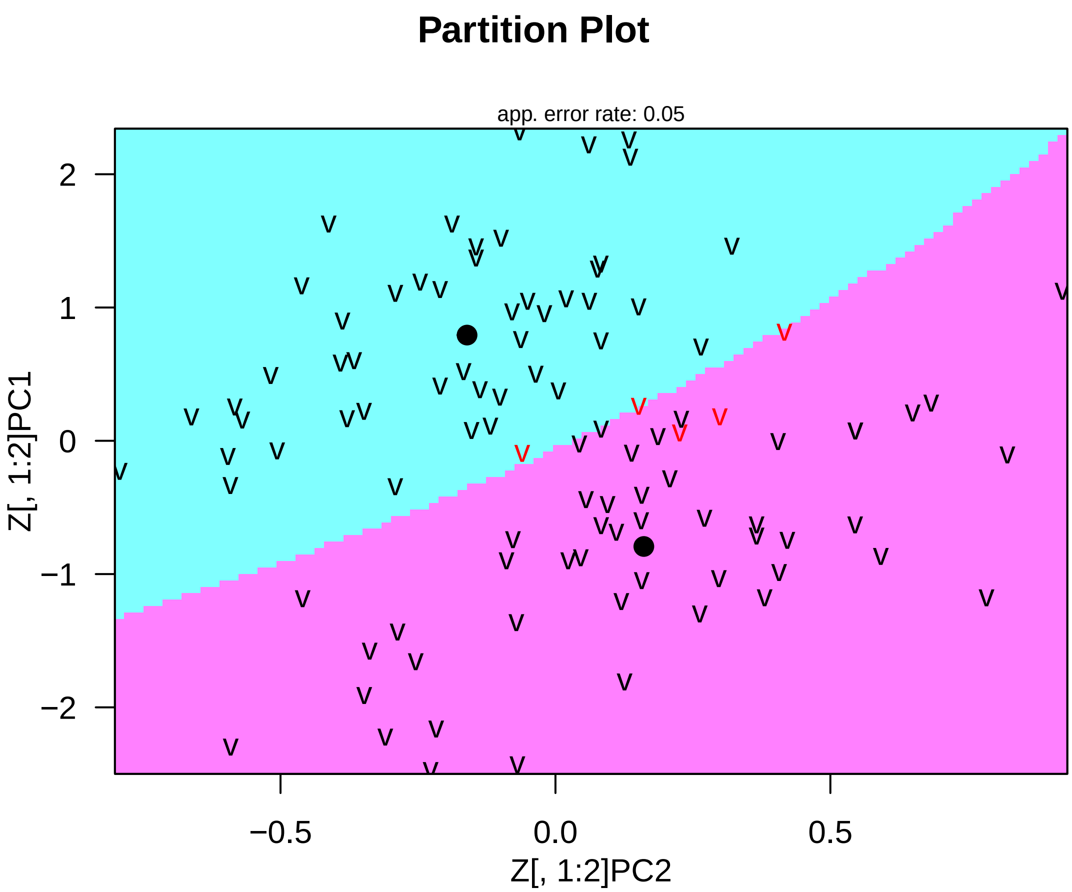
```r
fit.QDA = predict(qda)$class
table(irisv$Species, fit.QDA)
```

```
##             fit.QDA
##              versicolor virginica
##   versicolor         47         3
##   virginica           2        48
```

```r
# show results
library(klaR)
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 2, 1))
partimat(Species ~ Z[, 1:2], method = "lda")
```




```r
partimat(Species ~ Z[, 1:2], method = "qda")
```



Support Vector Machines

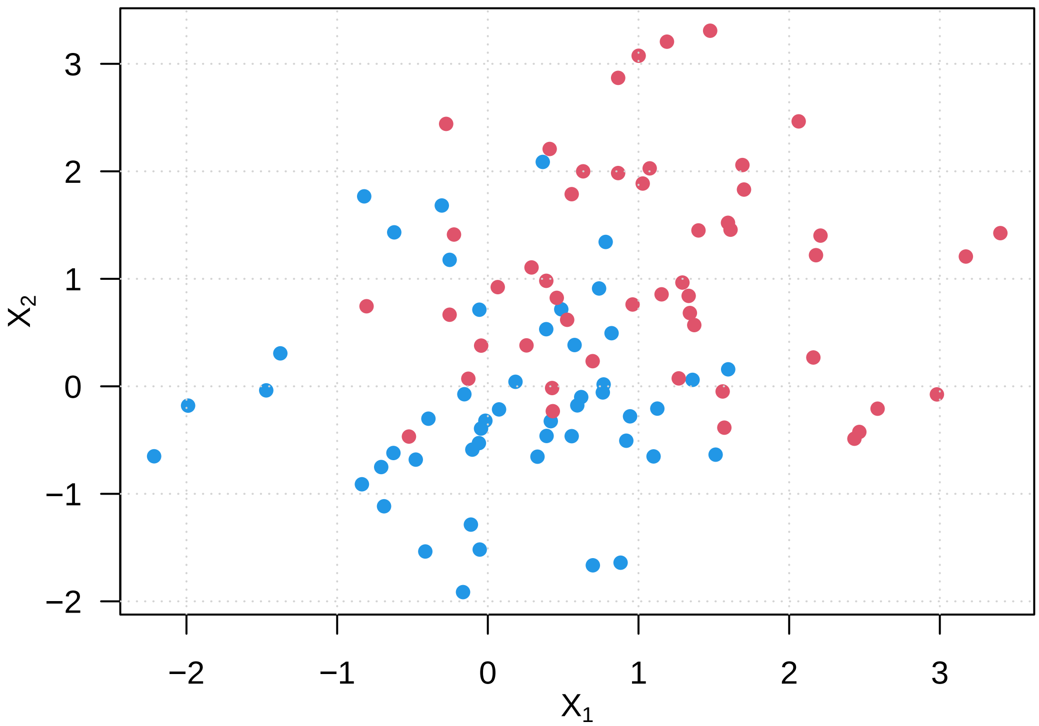
This part of the lab is primarily adapted from ISLR Chapter 9 SVM Lab.

Here we demonstrate the use of the `svm()` function from the library `e1071` on a two-dimensional toy example so that we can visualized the resulting decision boundary. We begin by generating the observations, which belong to two classes, and checking whether the classes are linearly separable.

Simulating Example

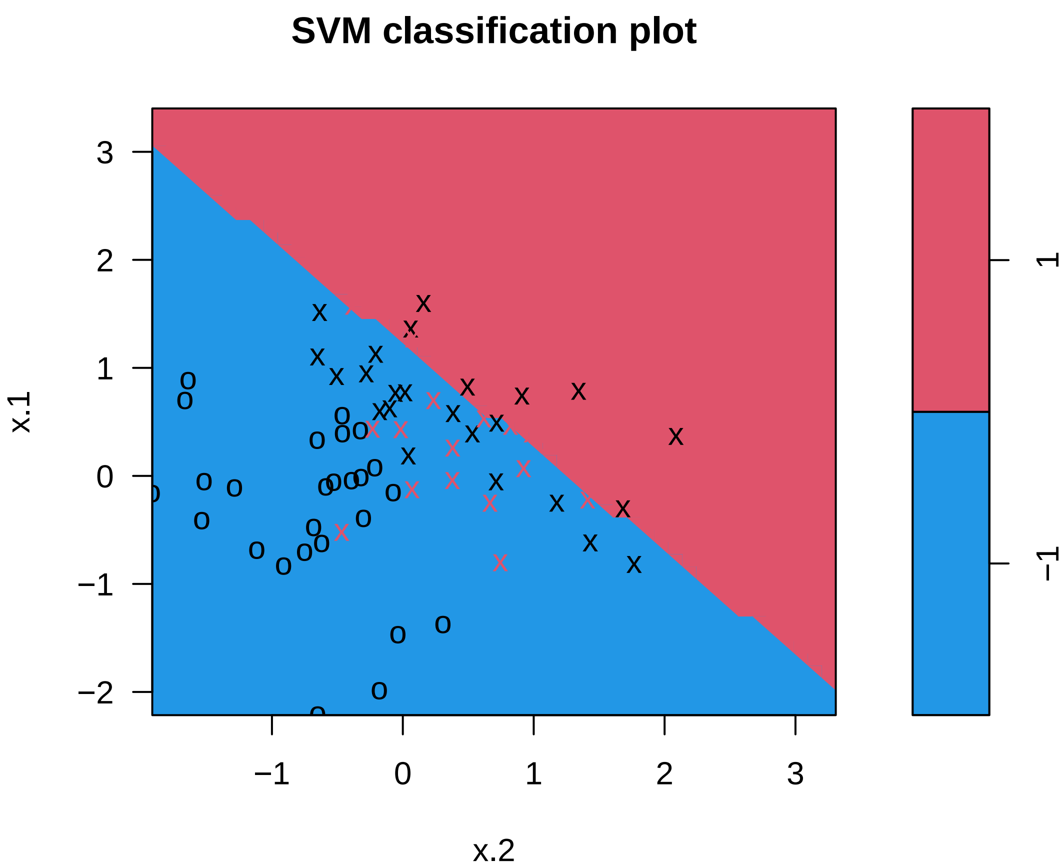
First, simulate two predictors that are independent of each other and follow a standard normal distribution. Set the first half of the response to -1 and the second half of the response to 1. Then, shift the predictor values by 1 when the response is 1.

```
set.seed(1)
x <- matrix(rnorm (100 * 2), ncol = 2)
y <- c(rep(-1, 50), rep(1, 50))
x[y == 1, ] <- x[y == 1, ] + 1
par(las = 1, mgp = c(2, 1, 0))
plot(x, col = ifelse(y == -1, 4, 2),
     xlab = expression(X[1]), ylab = expression(X[2]), pch = 16)
grid()
```



Train a Support Vector Machine

```
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, col = c(4, 2))
```




```r
# support points
svmfit$index
```

```
##  [1]   2   4   6   7   8   9  10  11  13  15  18  19  20  21  22  25  26  31  33
## [20]  39  40  42  47  48  52  53  54  55  58  59  62  65  67  68  69  72  74  75
## [39]  81  84  88  89  90  91  96  97  98  99 100
```

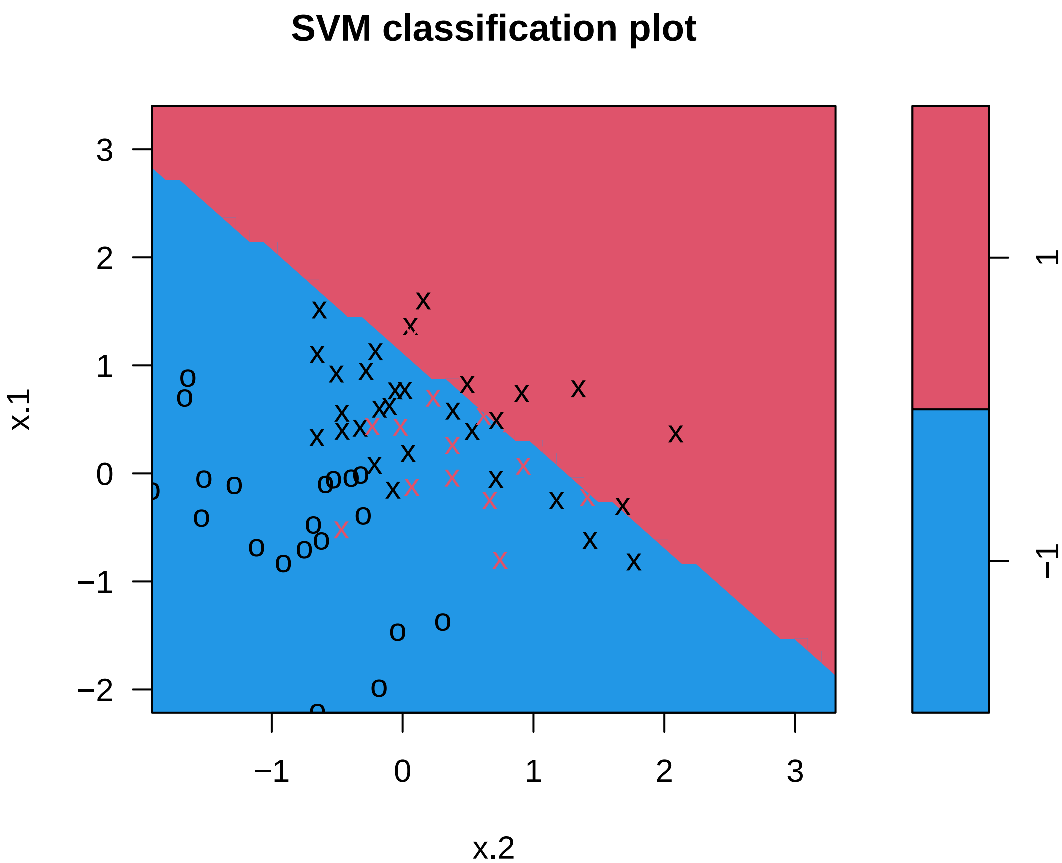
```r
# summary
summary(svmfit)
```

```
##
## 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:  49
##
##  ( 24 25 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Changing cost to allow for a wider margin

```
svmfit <- svm(y ~ ., data = dat , kernel = "linear",
cost = 0.1, scale = FALSE)
plot(svmfit, dat, col = c(4, 2))
```




```
svmfit$index
```

```
##  [1]   2   4   5   6   7   8   9  10  11  12  13  15  18  19  20  21  22  23  25
## [20]  26  27  30  31  33  39  40  42  44  47  48  52  53  54  55  58  59  62  65
## [39]  67  68  69  72  74  75  76  77  80  81  84  86  88  89  90  91  93  95  96
## [58]  97  98  99 100
```

Cross-Validation

The e1071 library includes a built-in function, `tune()`, to perform cross-validation. Here we compare SVMs with a linear kernel, using a range of values of the `cost` parameter.

```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
##     5
##
## - best performance: 0.2
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.45 0.24152295
## 2 1e-02  0.24 0.10749677
## 3 1e-01  0.21 0.09944289
## 4 1e+00  0.21 0.08755950
## 5 5e+00  0.20 0.10540926
## 6 1e+01  0.20 0.10540926
## 7 1e+02  0.20 0.10540926
```

```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:  5
##
## Number of Support Vectors:  49
##
##  ( 24 25 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Predcition

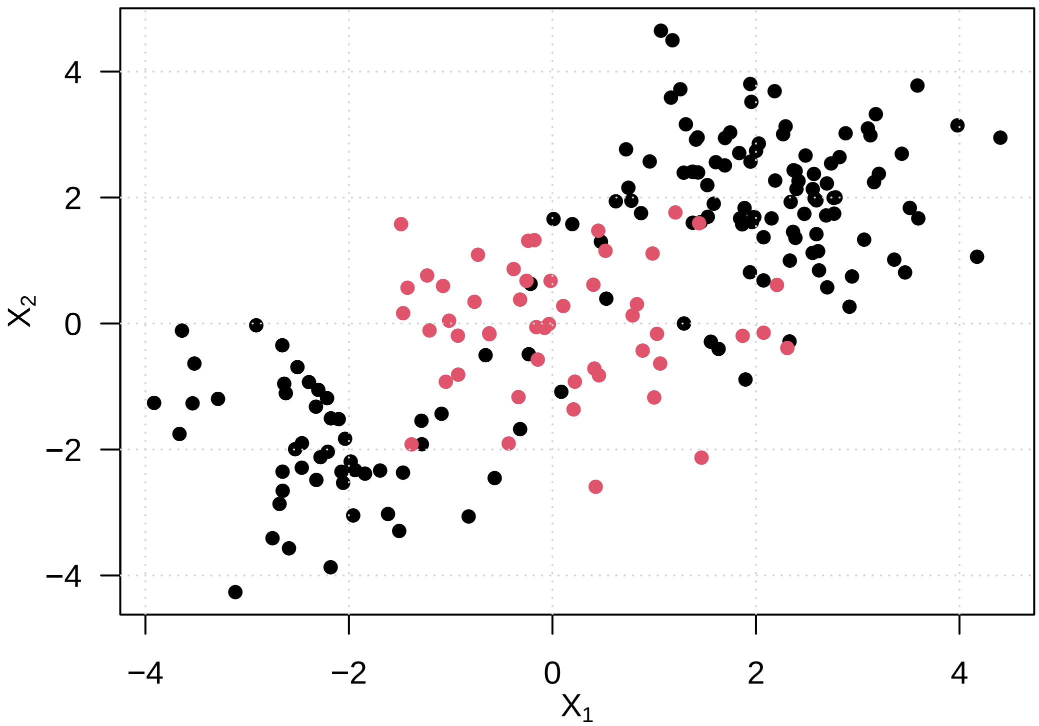
The `predict()` function can be used to predict the class label on a set of test observations, at any given value of the cost parameter.

```
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  8  2
##      1   0 10
```

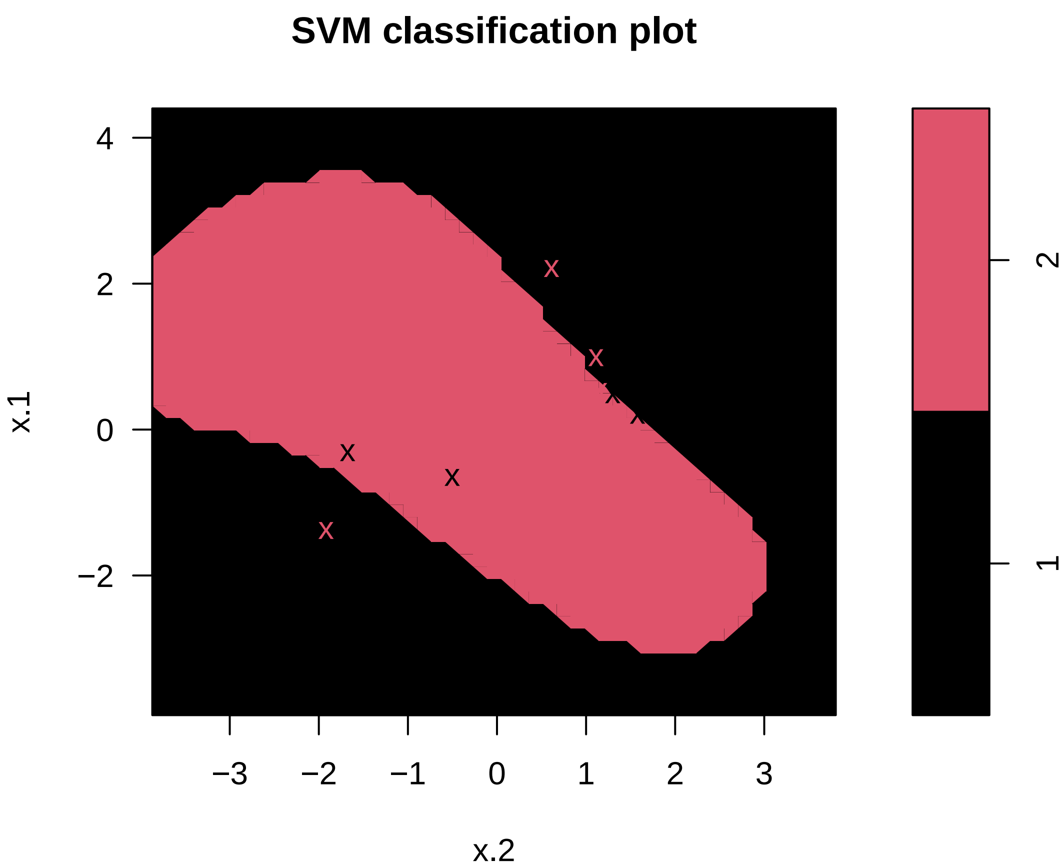
Generate some data with nonlinear class boundary

```
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(las = 1, mgp = c(2, 1, 0))
plot(x, col = ifelse(y == 1, 1, 2),
     xlab = expression(X[1]), ylab = expression(X[2]), pch = 16)
grid()
```



Training an SVM Using a Non-Linear Kernel

```
train <- sample (200, 100)
svmfit <- svm(y ~ ., data = dat[train , ], kernel = "radial",
gamma = 1, cost = 1)
plot(svmfit, dat[train , ], col = 1:2)
```

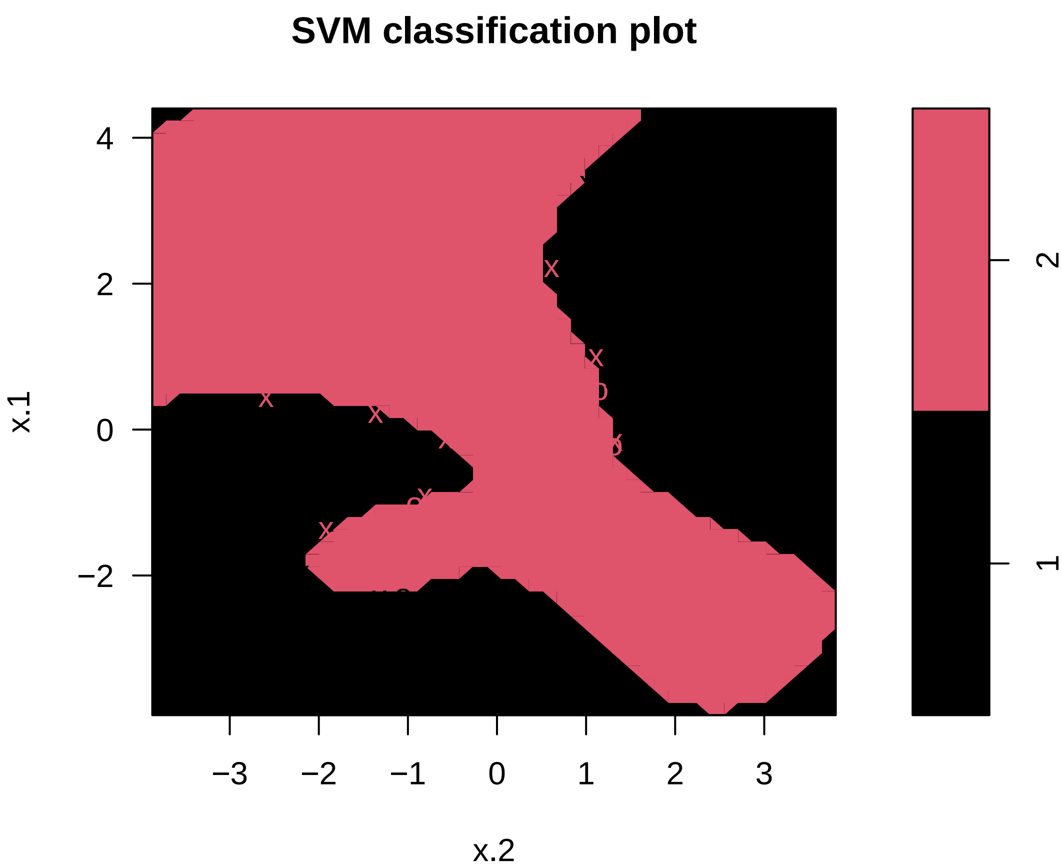



```r
summary(svmfit)
```

```
##
## 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
```

Changing to a Higher Cost Value

```r
svmfit <- svm(y ~ ., data = dat[train , ], kernel = "radial",
gamma = 1, cost = 1e5)
plot(svmfit, dat[train, ], col = 1:2)
```



Cross-Validation

```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
```

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

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