Predictive Modeling

Series 6

Exercise 6.1

Here we explore the maximal margin classifier on a toy data set. First, solve the following tasks by hand, and then use **R** .

a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label according to the following table.

Obs.	X_1	X_2	Y
1	3	4	Red
2	2	2	Red
3	4	4	Red
4	1	4	Red
5	2	1	Blue
6	4	3	Blue
7	4	1	Blue

Sketch the observations.

b) Sketch the optimal separating hyperplane, and provide its equation.

c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

d) On your sketch, indicate the support vectors for the maximal margin classifier.

e) Indicate the margin for the maximal margin hyperplane.

f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

g) Sketch a hyperplane that is separating but *not* the optimal separating hyperplane, and provide the equation for this hyperplane.

h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

Exercise 6.2

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. You may modify the following code:

```
set.seed(42)

x1 = runif(500) - 0.5
x2 = runif(500) - 0.5

y = ifelse(-0.2 * x1 + x2 > 0, "red", "blue")
plot(x1, x2, col = y)
```

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?

c) Generate an appropriate test 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 yield the fewest training errors and the fewest cross-validation errors?

d) Discuss your results.

Result Checker

Predictive Modeling

Solutions to Series 6

Solution 6.1

a)
```
x <- matrix(c(3, 2, 4, 1, 2, 4, 4, 4, 2, 4, 4, 1, 3, 1),
    ncol = 2)
y <- c(rep(1, 4), rep(-1, 3))
plot(x, col = 3 - y)
```

b) The function `coef()` returns the coefficients of the SVM model, i.e., β_0, β_1, and β_2 as named numerics with the names `(Intercept)`, `x.1`, and `x.2`, respectively. In order to plot the line with the function `abline()`, we need the X_2-intercept a and slope b,i.e., an equation of the form $X_2 = a + bX_1$. We have

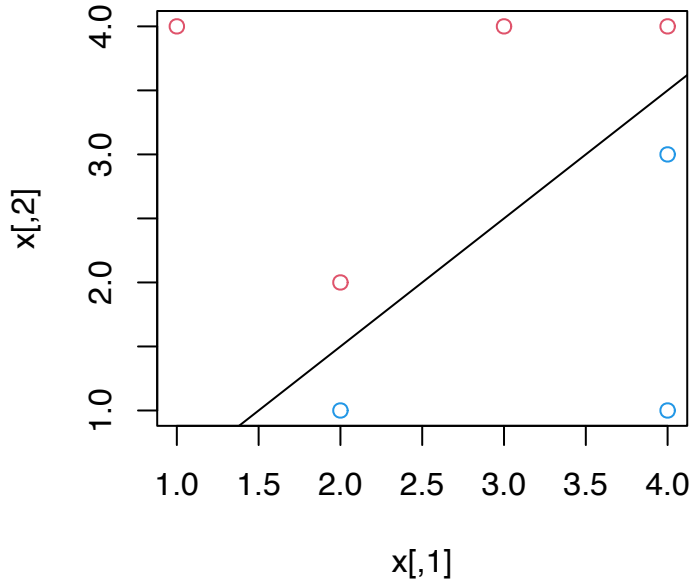
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \quad \Leftrightarrow \quad X_2 = -\frac{\beta_1}{\beta_2} X_1 - \frac{\beta_0}{\beta_2}$$

and hence

$$a = -\frac{\beta_0}{\beta_2} \quad \text{and} \quad b = -\frac{\beta_1}{\beta_2}$$

```
library(e1071)
dataset <- data.frame(x = x, y = as.factor(y))
svm.fit <- svm(y ~ ., data = dataset, kernel = "linear",
    cost = 10, scale = FALSE)
coefficients <- coef(svm.fit)
beta0 <- coefficients[["(Intercept)"]]
beta1 <- coefficients[["x.1"]]
beta2 <- coefficients[["x.2"]]

a <- -beta0/beta2
b <- -beta1/beta2
abline(a, b)
```



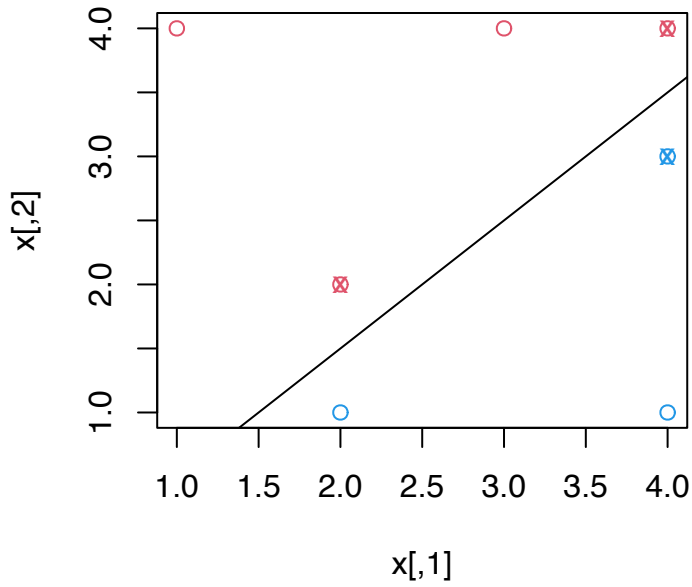
Here, we have $\beta_0 = 1$, $\beta_1 = -2$, and $\beta_2 = 2$, so the separating hyperplane has the equation

$$1 - 2X_1 + 2X_2 = 0$$

c) The rule is: "Classify to red if $1 - 2X_1 + 2X_2 > 0$, and classify to blue otherwise." Note that one has to check the colors.

d) The indices of the support vectors can be obtained from the field **index**.

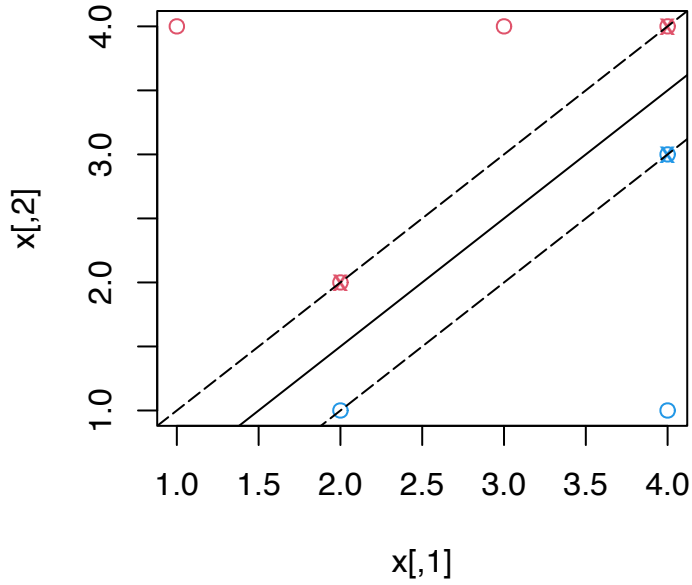
```
sv_ind <- svm.fit[["index"]]  # indices of support vectors
x1 <- x[sv_ind, 1]  # X1-coordinates of support vectors
x2 <- x[sv_ind, 2]  # X2-coordinates of support vectors
points(x1, x2, pch = "x", col = 3 - y[sv_ind])
```



Note that the point $(2,1)$ is indeed a support vector, but not recognized by **R**. This is due to small numerical errors.

e) The bounding hyperplanes are above resp. below the separating hyperplane and have the same slope. The vertical shift can be obtained from one of the support vectors. If the support vector is (x_1, x_2), then the shift is x_2 minus the value of the separating hyperplane at x_1, so $x_2 - (a + bx_1)$.

```
shift <- x[sv_ind[1], 2] - (a + b * x[sv_ind[1], 1])
a.o <- a + shift
a.u <- a - shift
abline(a.o, b, lty = "longdash")
abline(a.u, b, lty = "longdash")
```



f) Observation 7 is the lower right blue point. Since the support vectors are closer to the separating hyperplane than this observation, they are still closer after a slight movement of observation 7.

g) We can use the parallel hyperplane $1.1 - 2X_1 + 2X_2 = 0$.

h) E.g., if one adds a blue observation above the upper bounding hyperplane of the margin, then the points are not separable by a hyperplane any more.

Solution 6.2

b) A straightforward way to produce the training error is to simply train several models.

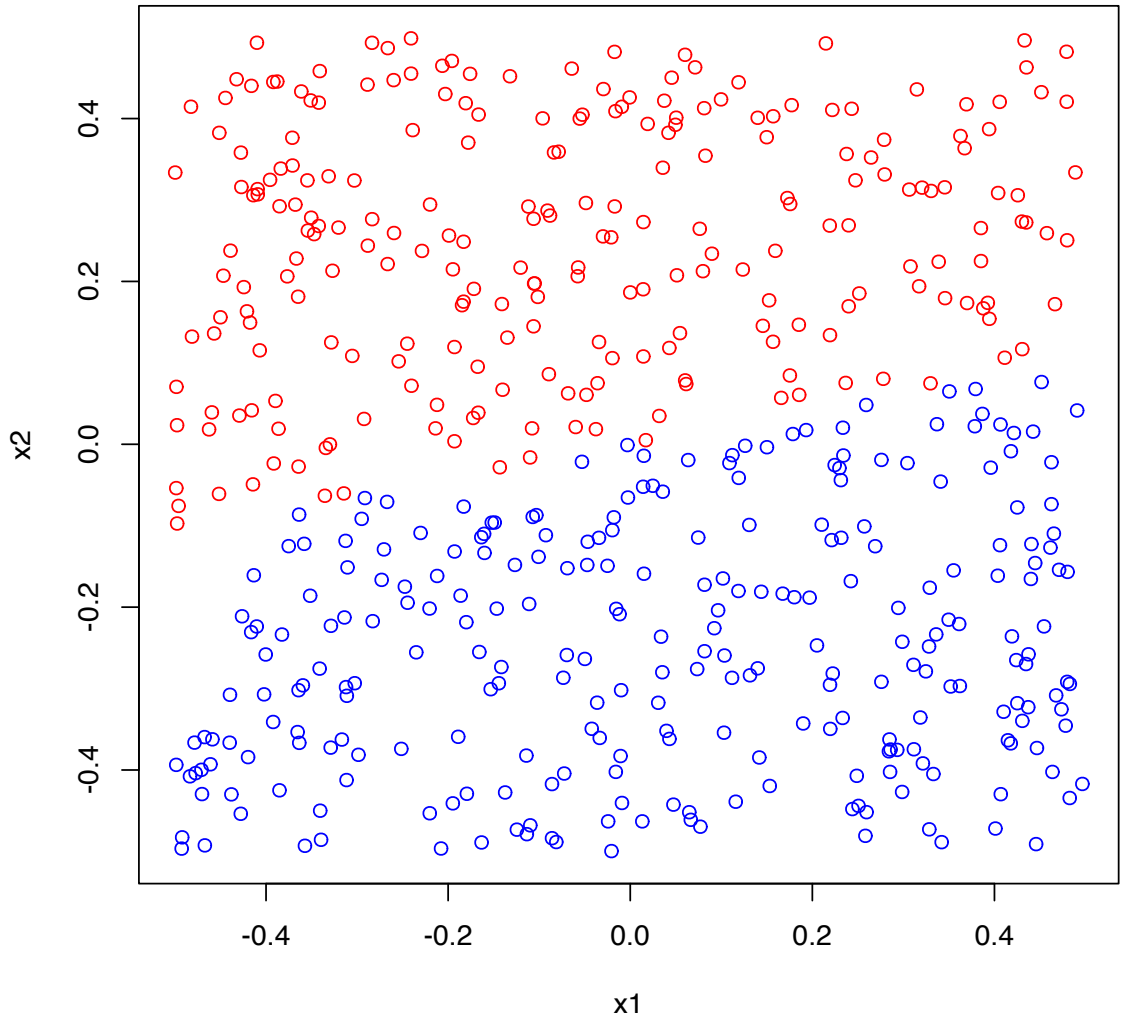
```
library(e1071)
costs=seq(1, 50, length.out = 20)
y = as.factor(y)

perf=c()
for(c in costs){
  svm.fit = svm(y~.,
                data=data.frame(x1,x2,y),
                type='C',
                kernel='linear',
```

a)
```r
set.seed(42)

x1 = runif(500) - 0.5
x2 = runif(500) - 0.5

y = ifelse(-0.2 * x1 + x2 > 0, "red", "blue")
plot(x1, x2, col = y)
```




```r
                    cost=c)
svm.pred = predict(svm.fit,
                       data.frame(x1,x2))
```

```
  error = mean(svm.pred!=y)
  perf = rbind(perf, c(c,error))
}
print(sprintf('Linear SVM training error rate= %10.6f',
          min(perf)))

## [1] "Linear SVM training error rate=   0.002000"
```

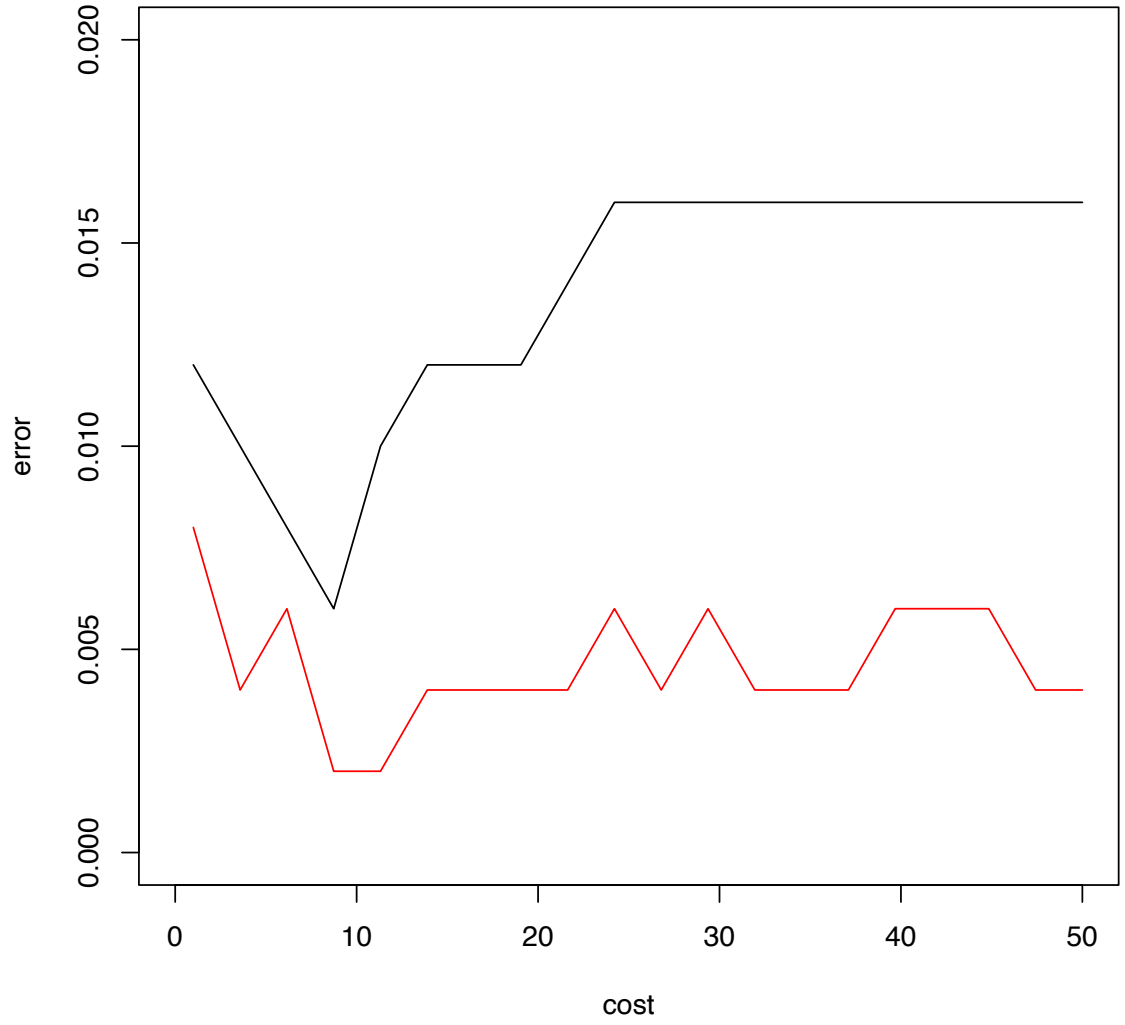
The **tune** function can be used in order to obtain cross validation error. If we choose to do 10-fold cross validation, we simply need to set the parameter **tunecontrol=tune.control(cross=20)**. Note that different executions of the snippet below will produce different graphs due to the sampling that occurs when cross validation is done.

```
set.seed(42)
costs=data.frame(cost=costs)
svm.tune = tune(svm, y~.,
                data=data.frame(x1, x2, y),
                ranges=costs,
                tunecontrol=tune.control(cross=10))
print(sprintf('Linear SVM CV error rate= %10.6f',
              min(svm.tune$performances[,2])))

## [1] "Linear SVM CV error rate=   0.006000"
```

From the plot one can note that the cross-validation error (shown in black) and the training error follow the same trend, with the cross-validation error achieving a higher value.

```
plot(svm.tune$performances[,c(1,2)], type='l',
     ylim=c(0,0.02), xlim=c(0,50))
lines(perf, col='red')
```



c) We generate test data:

```
set.seed(87)

x1 = runif(500) - 0.5
x2 = runif(500) - 0.5

y = ifelse(-0.2 * x1 + x2 > 0, "red", "blue")
```

```
dat_test = data.frame(x1 = x1, x2 = x2, y = y)
y_hat = predict(svm.tune$best.model, newdata = dat_test)
print(sum(y_hat == y))

## [1] 497

print(sprintf("Linear SVM test error rate= %10.6f", 1 -
    sum(y_hat == y)/length(y)))

## [1] "Linear SVM test error rate=   0.006000"
```

d) The test error rate corresponds to the CV error rate. The training error rate was overoptimistic.