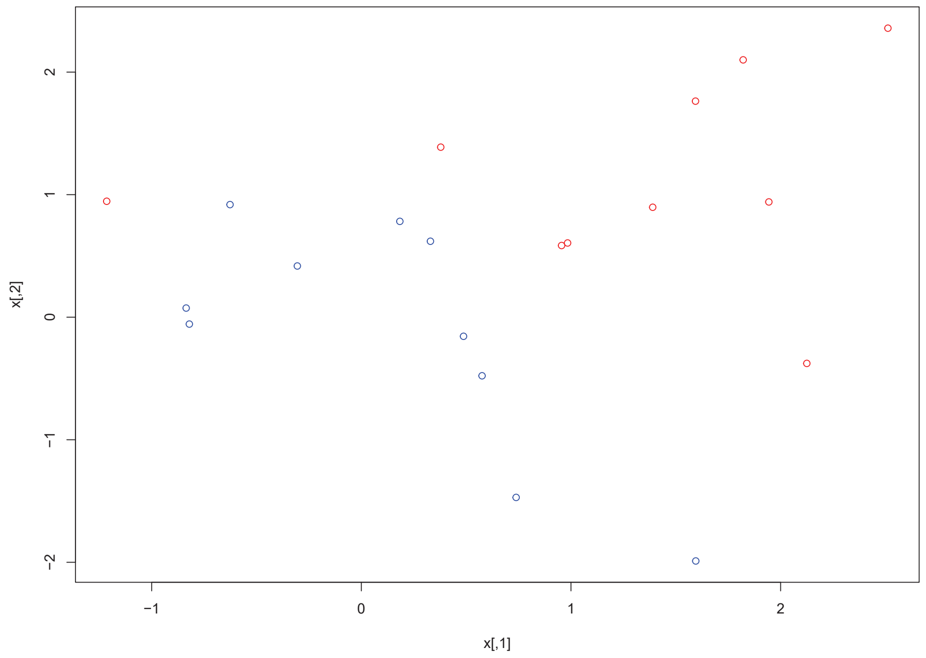
demonstrate the use of this function on a two-dimensional example so that we can plot the resulting decision boundary. We begin by generating the observations, which belong to two classes, and checking whether the classes are linearly separable.

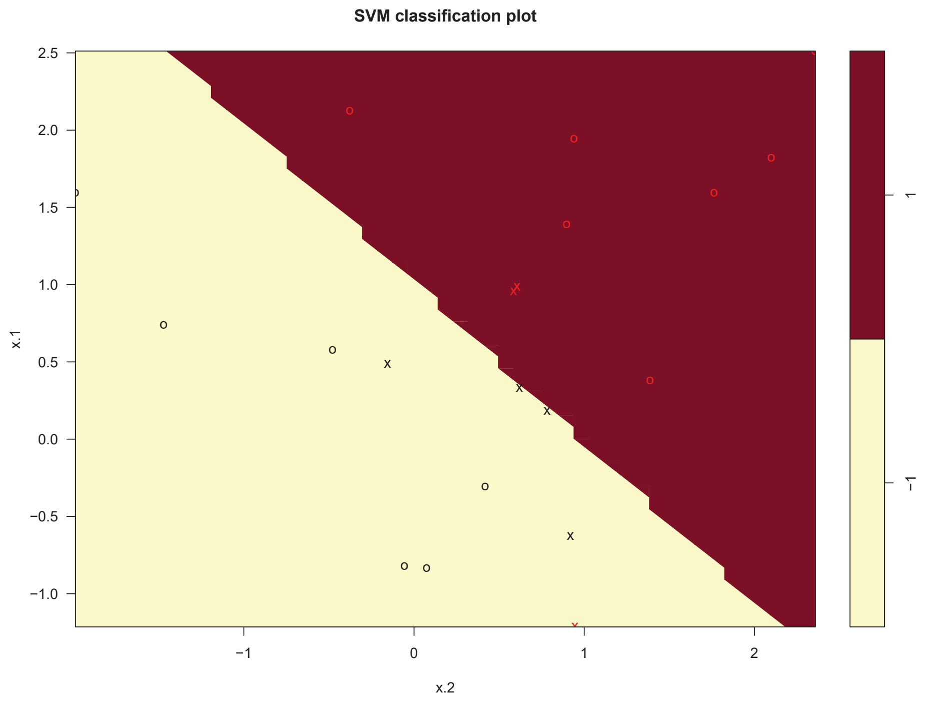
```
> set.seed(1)
> x <- matrix(rnorm(20 * 2), ncol = 2)
> y <- c(rep(-1, 10), rep(1, 10))
> x[y == 1, ] <- x[y == 1, ] + 1
> plot(x, col = (3 - y))
```



The classes are not linealy separable.

Next, we fit the support vector classifier and plot it.

```
> 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)
```



The region of feature space that will be assigned to the −1 class is shown in light yellow, and the region that will be assigned to the +1 class is shown in red. The decision boundary between the two classes is linear. The support vectors are plotted as crosses and the remaining observations are plotted as circles; We can determine the identities of support vectors as follows:

```
> svmfit$index
[1]  1  2  5  7 14 16 17
```

We can obtain some basic information about the support vector classifier fit using the summary() command:

```
> 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

> table(predict = svmfit$fitted, truth = dat$y)
       truth
predict -1 1
     -1  8 1
     1   2 9
```

the training err. rate $= \frac{3}{20} = 0.15$

What if we instead used a smaller value of the cost parameter?

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

> svmfit$index
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
> table(predict = svmfit$fitted, truth = dat$y)
       truth
predict -1  1
     -1 10  1
     1   0  9
```

training err. rate $= \frac{1}{20} = 0.05$

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.

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

```
> 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: 0.1

- Detailed performance results:
   cost error dispersion
1 1e-03  0.70  0.4216370
2 1e-02  0.70  0.4216370
3 1e-01  0.10  0.2108185
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
```

We see that cost = 0.1 results in the lowest cross-validation error rate. The tune() function stores the best model obtained, which can be accessed as follows:

```
> 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
```

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. We begin by generating a test data set.

```
> 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))
```

Now we predict the class labels of these test observations. Here we use the best model obtained through cross-validation in order to make predictions.

```
> ypred <- predict(bestmod, testdat)
> table(predict = ypred, truth = testdat$y)
       truth
predict -1  1
     -1 10  3
     1   1  6
```

test err. rate $= \frac{4}{20} = 0.2$

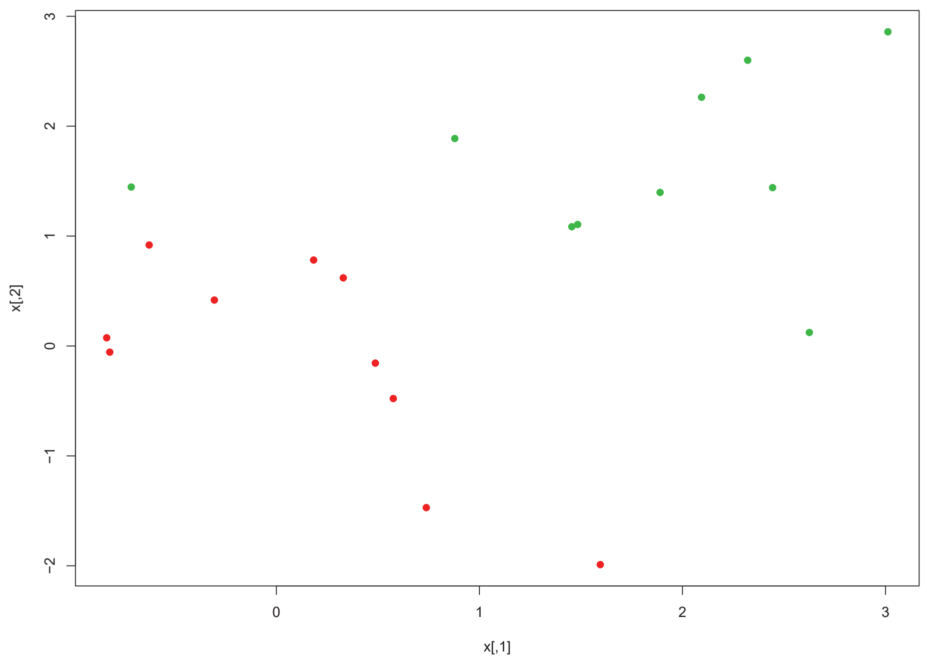
What if we had instead used cost = 0.01?

```
> 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 10  4
     1   1  5
```

test err. rate $= \frac{5}{20} = 0.25$

Now consider a situation in which the two classes are linearly separable. Then we can find a separating hyperplane using the svm() function. We first 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)
```

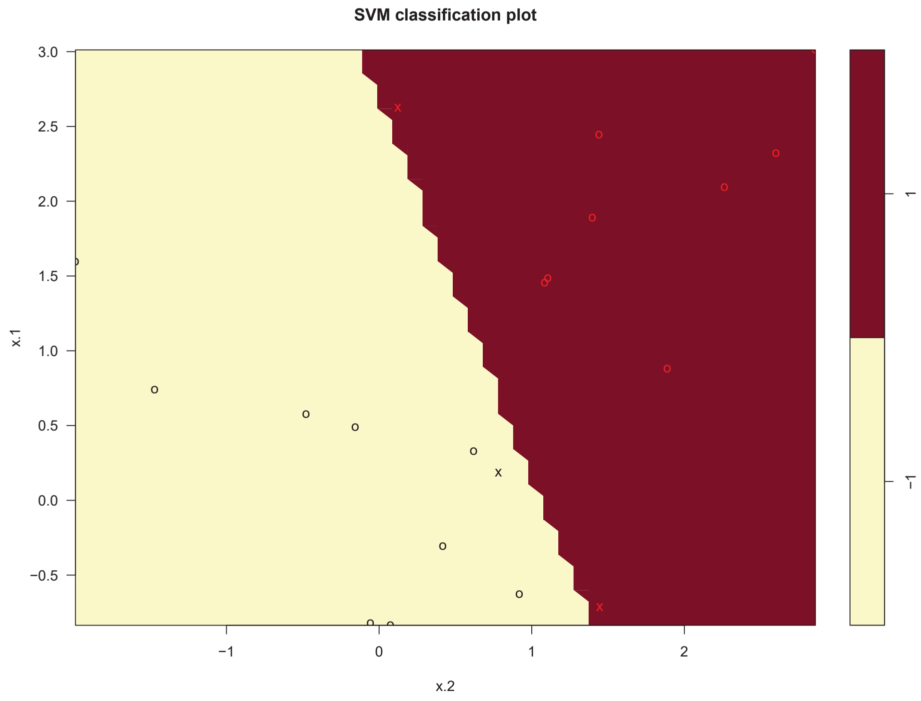


Now the observations are just barely linearly separable. We fit the support vector classifier and plot the resulting hyperplane, using a very large value of cost so that no observations are misclassified.

```
> 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


> table(predict = svmfit$fitted, truth = dat$y)
       truth
predict -1  1
     -1 10  0
     1   0 10

> plot(svmfit, dat)
```

training err. rate = 0



No training errors were made and only three support vectors were used. However, we can see from the figure that the margin is very narrow (because the observations that are not support vectors, indicated as circles, are very close to the decision boundary). We can expect that this model will perform poorly on test data:

```
> xtest <- matrix(rnorm(20 * 2), ncol = 2)
> ytest <- sample(c(-1, 1), 20,  rep = TRUE)
> xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 1.5
> testdat <- data.frame(x = xtest, y = as.factor(ytest))
> ypred <- predict(svmfit, testdat)
> table(predict = ypred, truth = testdat$y)
       truth
predict -1 1
     -1  6 5
     1   3 6
```

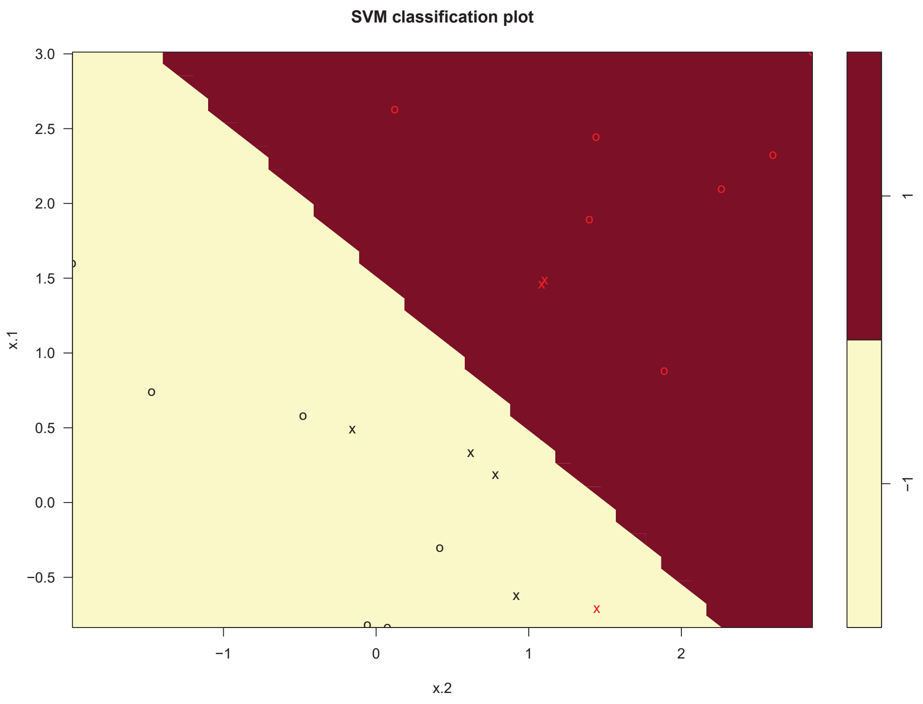
test err. rate $= \frac{8}{20} = 0.4$

We now try a smaller value of cost:

```
> 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

> table(predict = svmfit$fitted, truth = dat$y)
       truth
predict -1  1
     -1 10  1
     1   0  9
```

training err. rate $= \frac{1}{20} = 0.05$

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



Using cost = 1, we misclassify a training observation, but we also obtain a much wider margin and make use of seven support vectors. It seems likely that this model will perform better on test data than the model with cost = 10^5.