Support vector classifier

Trevor Hastie and Robert Tibshirani

Here, I am adapting part of the lab associated with Chapter 9 of the textbook.

We use the `e1071` library in `R` to demonstrate the support vector classifier and the SVM. Another option is the `LiblineaR` library, which is useful for very large linear problems.

Support Vector Classifier

The `e1071` library contains implementations for a number of statistical learning methods. In particular, the `svm()` function can be used to fit a support vector classifier when the argument `kernel = "linear"` is used. This function uses a slightly different formulation from the lecture for the support vector classifier.

The `type` argument specifies the algorithm to be invoked by the function. The algorithm is capable of doing both classification and regression. We only do classification in this course. Note that there are two types of classification algorithms, `nu` and `C` classification. The documentation says that they essentially differ in the way that they penalise margin and boundary violations, but that it can be shown to lead to equivalent results. Since we did everything in terms of `C`, we will continue to use `C`. The `C` refers to the cost which we discuss next.

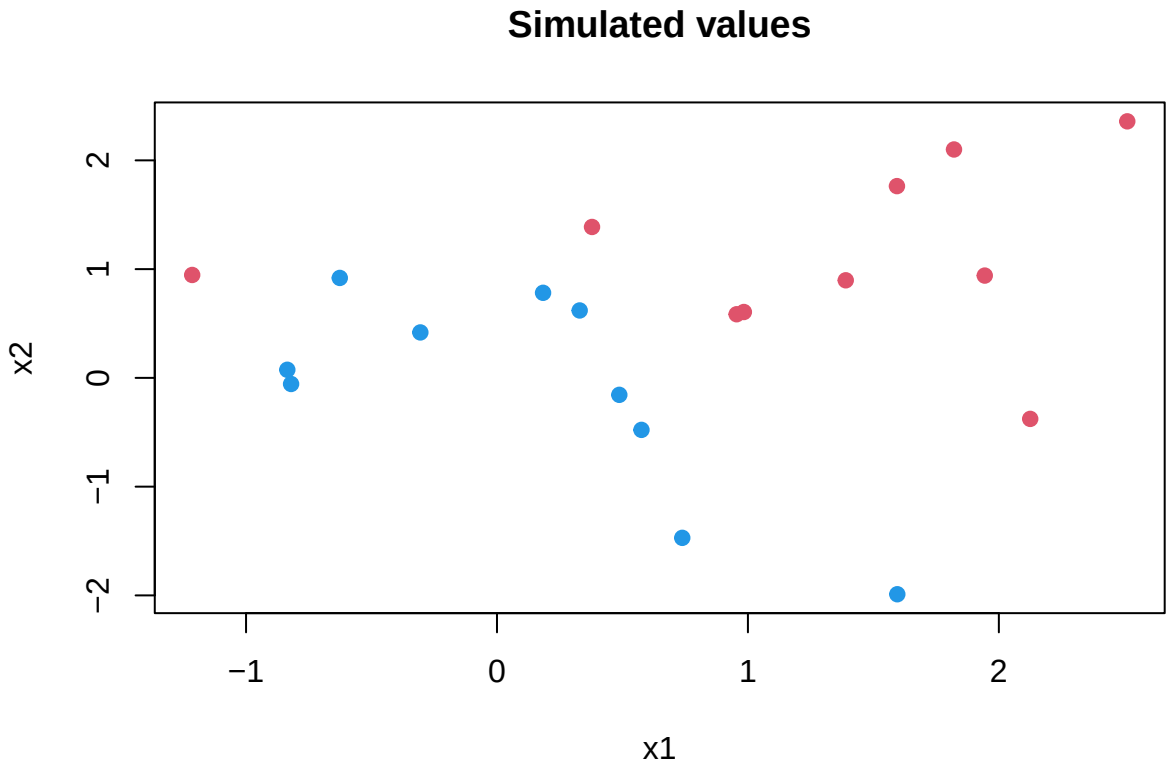
The `cost` argument specifies the penalty to be applied for boundary violations, i.e, it allows us to specify the cost of a violation to the margin. This argument can vary from 0 to "infinity" (in practice a large number compared to 0, say 10^6 or 10^8). When the `cost` argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the `cost` argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin. The default value is 1.

The `kernel` argument specifies the kind of function to be used to construct the decision boundary. The options are `linear`, `polynomial`, and `radial`. We'll first focus on linear kernels where the decision boundary is a straight line.

The `scale` parameter is a Boolean that tells the algorithm whether or not the data points should be scaled to have zero mean and unit variance (i.e. shifted by the mean and scaled by the standard deviation). Scaling is generally good practice to avoid undue influence of attributes that have unduly large numeric values. This choice requires domain knowledge.

We now use the `svm()` function to fit the support vector classifier for a given value of the `cost` parameter. Here we 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), pch=19,
     main="Simulated values",
     xlab="x1", ylab="x2")
```



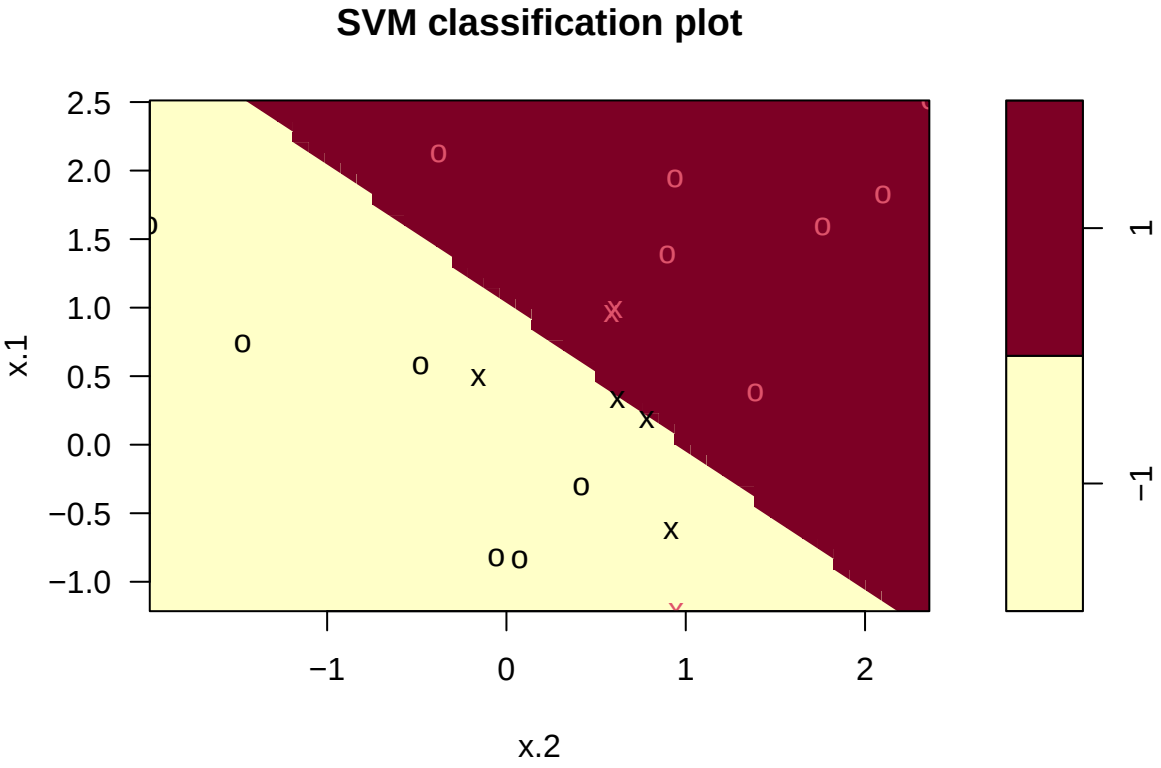
They are not. Next, we fit the support vector classifier. Note that in order for the `svm()` function to perform classification (as opposed to SVM-based regression), we **must** encode the response as a factor variable. We now create a data frame with the response coded as a factor.

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

As you can see, the argument `scale = FALSE` tells the `svm()` function **not** to scale each feature to have mean zero or standard deviation one; depending on the application, one might prefer to use `scale = TRUE`.

We can now plot the support vector classifier obtained:

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



Note that the two arguments to the SVM `plot()` function are the output of the call to `svm()`, as well as the data used in the call to `svm()`. 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 (because we used the argument `kernel = "linear"`), though due to the way in which the plotting function is implemented in this library the decision boundary looks somewhat jagged in the plot. (Note that here the second feature is plotted on the x-axis, i.e., the horizontal axis, and the first feature is plotted on the y-axis, i.e., the vertical axis in contrast to the behavior of the usual `plot()` function in `R` (and common sense).) The support vectors are plotted as crosses and the remaining observations are plotted as circles; we see here that there are seven support vectors. We can determine their identities 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)
```

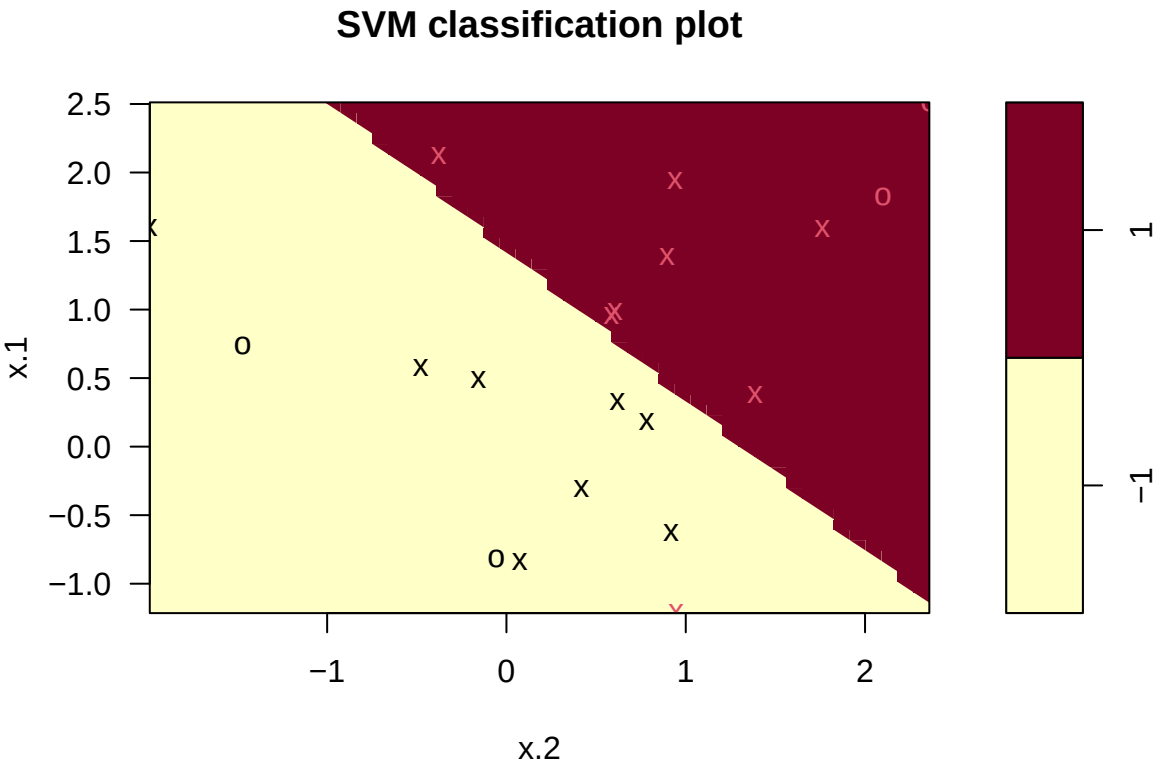
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
```

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

This tells us, for instance, that a linear kernel was used with `cost = 10`, and that there were seven support vectors, four in one class and three in the other.

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

```
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
```

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. Unfortunately, the `svm()` function does not explicitly output the coefficients of the linear decision boundary obtained when the support vector classifier is fit, nor does it output the width of the margin.

The `e1071` library includes a built-in function, `tune()`, to perform cross-validation. By default, `tune()` performs ten-fold cross-validation on a set of models of interest. In order to use this function, we pass in relevant information about the set of models that are under consideration. 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)))
```

We can easily access the cross-validation errors for each of these models using the `summary()` command:

```
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   0.001  0.55  0.4377975
## 2   0.010  0.55  0.4377975
## 3   0.100  0.05  0.1581139
## 4   1.000  0.15  0.2415229
## 5   5.000  0.15  0.2415229
## 6  10.000  0.15  0.2415229
## 7 100.000  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  9 1
##      1   2 8
```

Thus, with this value of `cost`, 17 of the test observations are correctly classified. What if we had instead used `cost = 0.01`?

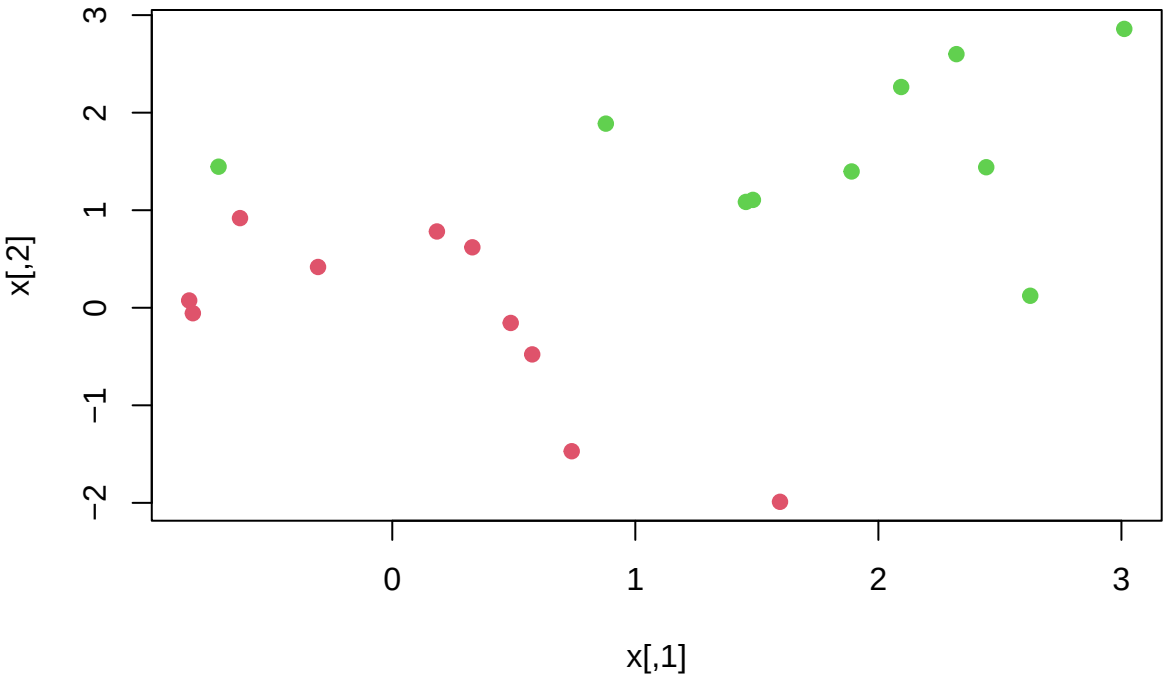
```
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
    cost = .01, scale = FALSE)
ypred <- predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)
```

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

In this case three additional observations are misclassified.

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)
```

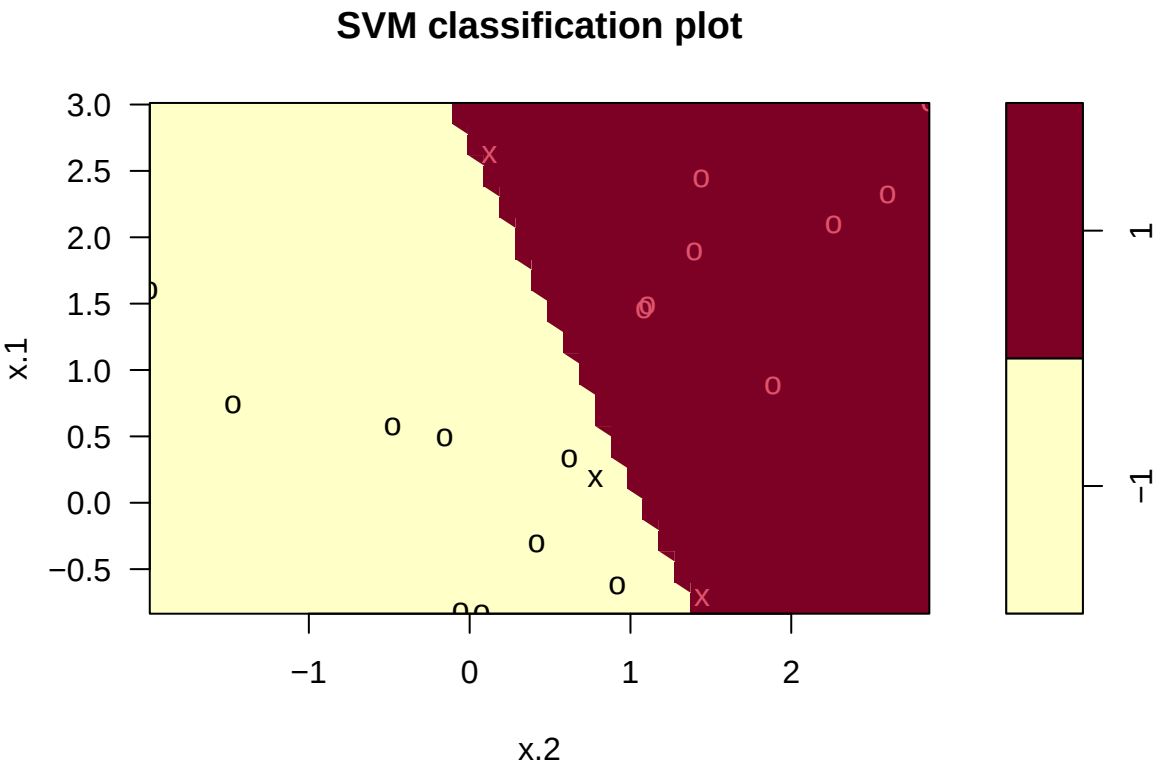


Now the observations are just barely linearly separable. We fit the support vector classifier and plot the resulting hyperplane, using a very large value of `cost` so that no observations are misclassified.

```
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 = 100000)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100000
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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



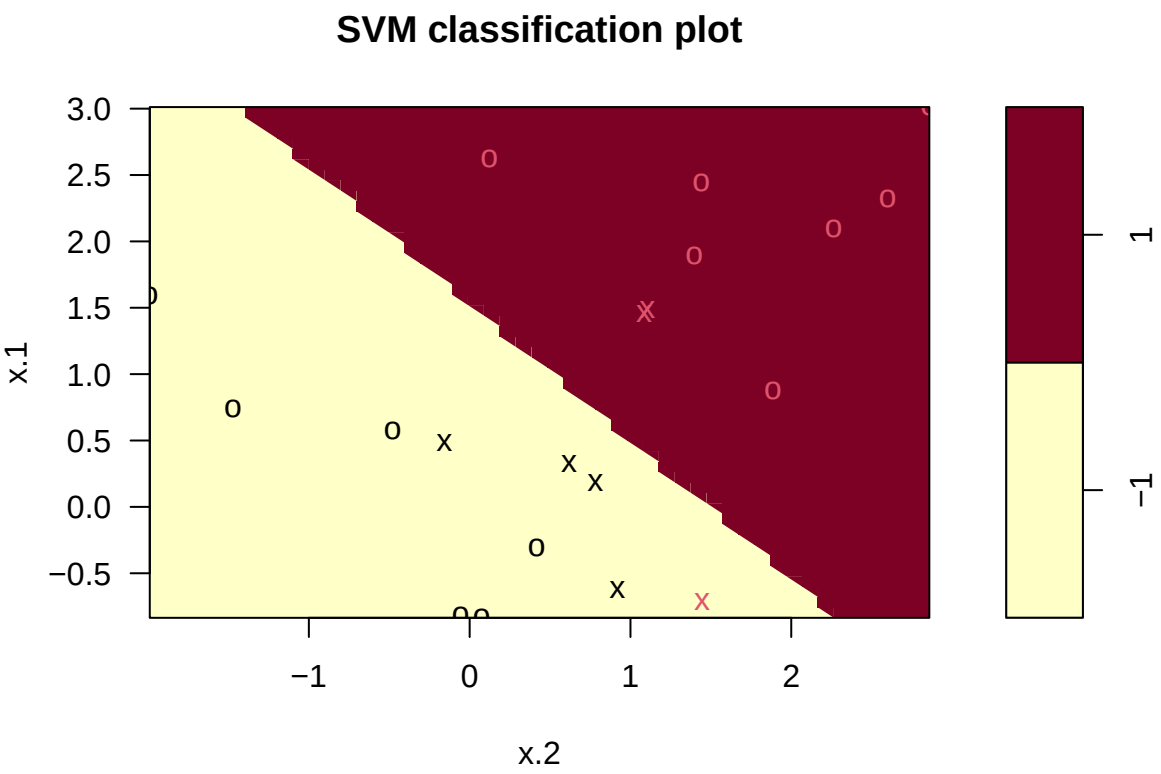
No training errors were made and only three support vectors were used. However, we can see from the figure

that the margin is very narrow (because the observations that are not support vectors, indicated as circles, are very close to the decision boundary). It seems likely that this model will perform poorly on test data. 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
```

```
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 = 1e5`.