Support Vector Machines

Nathan Sandford, Chris Donnay, and Mike Chaykowsky

To work through an example of support vector machines *SVM) in R, we will use the pacakge e1071, which contains several useful SVM functions. Our sample datasets are synth.tr and synth.te from the MASS package. They are synthetic datasets involving two dimensions and 2 classes made to train and test classification methods.

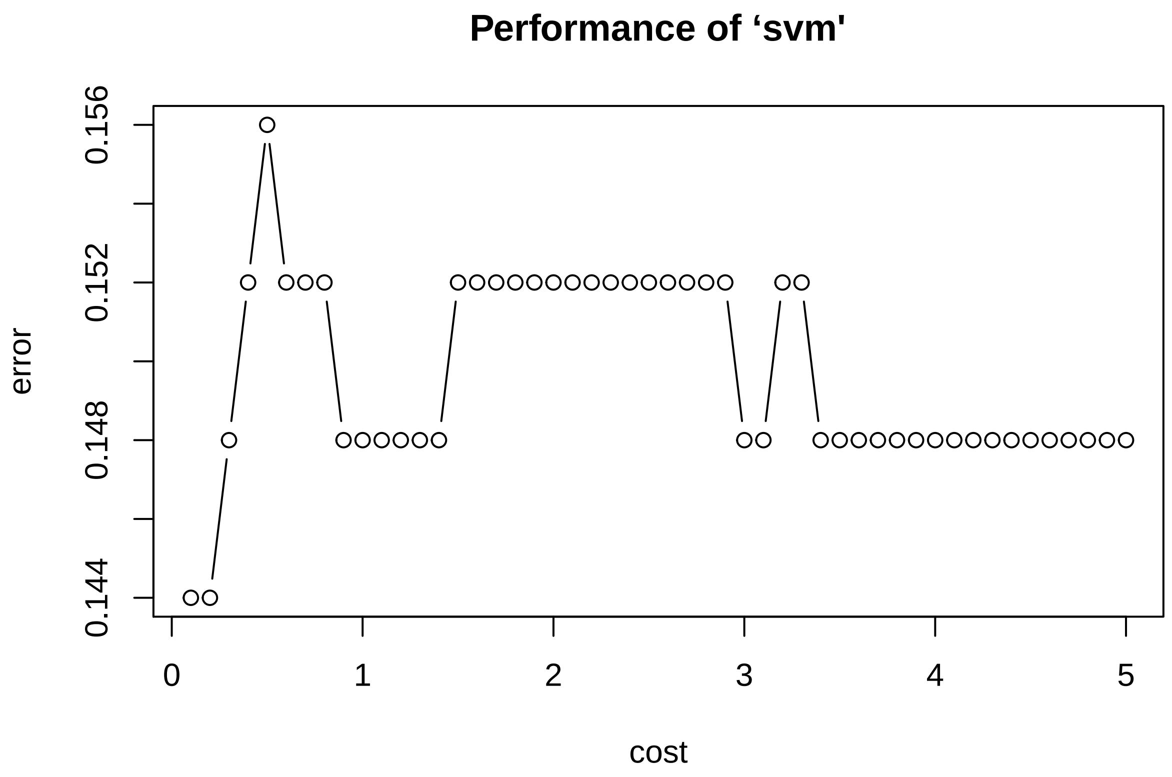
Reading in training and testing data:

```r
library(e1071)
library(MASS)

# Training Data
data(synth.tr)
train <- synth.tr
train$yc <- as.factor(train$yc) # make class a factor

# Testing data
data(synth.te)
test <- synth.te
test$yc <- as.factor(test$yc)# make class a factor
```

e1071 features a SVM function, svm() that can handle both classification and regression problems using 4 different basis kernels: linear, polynomial, radial, and sigmoid. We will train each of these kernels on the synth.tr dataset and then test the model with the synth.te data set.

Before we train the model, we use the tune.svm() function to determine the best parameters of the model. For the simple linear case, this only includes the cost of constraints violation, which acts to regularize the model. The optimum parameter(s) is chosen to be the parameter(s) that minimizes the error for 10-fold cross validation. Once the parameters are tuned, we train the SVM on the training data.

```r
# Tune Parameters (Cost)
linear_tune <- tune.svm(yc ~ ., data = train, kernel = "linear", cost=seq(0.1,5,0.1))
plot(linear_tune)
```



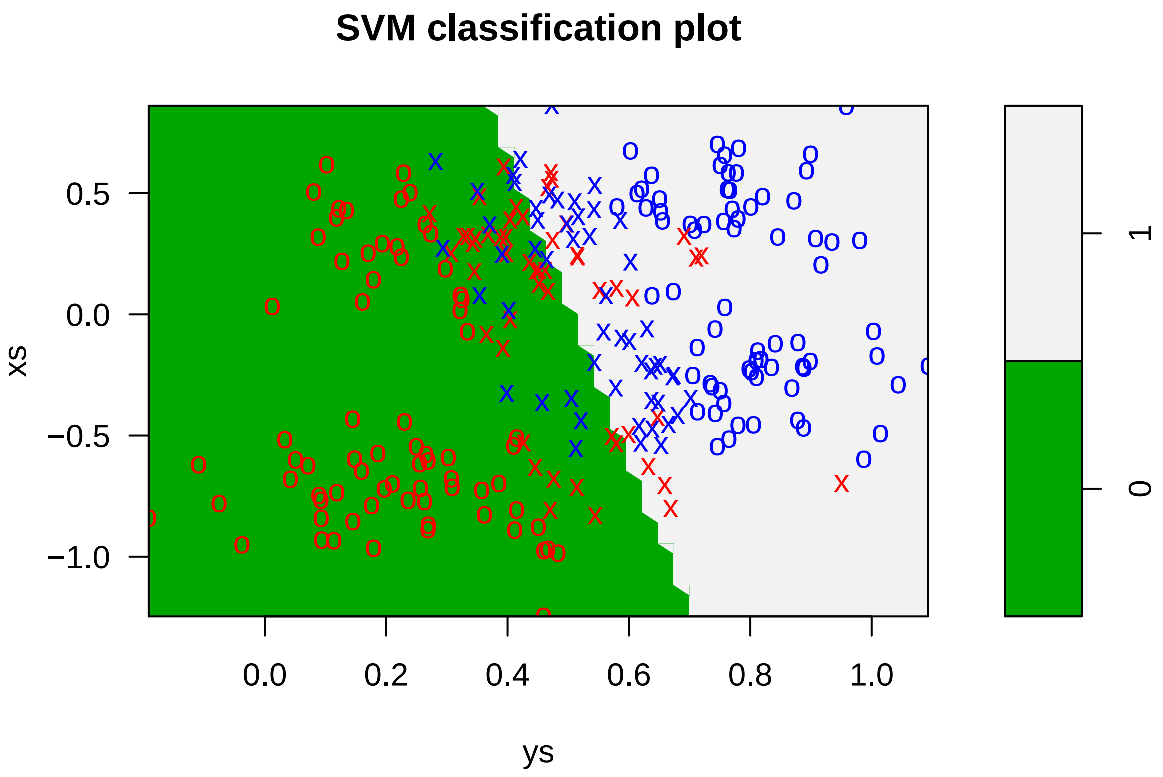
```r
# Optimum Cost (complicated way of determining first set of parameters to yield minimum error)
c <- linear_tune$performances[min(which(linear_tune$performances[2] == min(linear_tune$performances[2]))

# Train Linear Model
linear_svm <- svm(yc ~ ., data = train, kernel = "linear", cost=c)
```

Testing the linear model back on the training data we see that the model does pretty well given the difficult overlapping populations of the two classes. The confusion matrix shows the number of misclassifications for each class. In the plot, X's mark the support vectors of the SVM while the O's mark the remaining data points.

```r
#Test on Training Data
linear_training_pred <- predict(linear_svm,train)
#Confusion Matrix
table(linear_training_pred, train$yc)
```

```
##
## linear_training_pred   0   1
##                    0 104  13
##                    1  21 112
```

```r
#Visual Representation
plot(linear_svm,train,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```

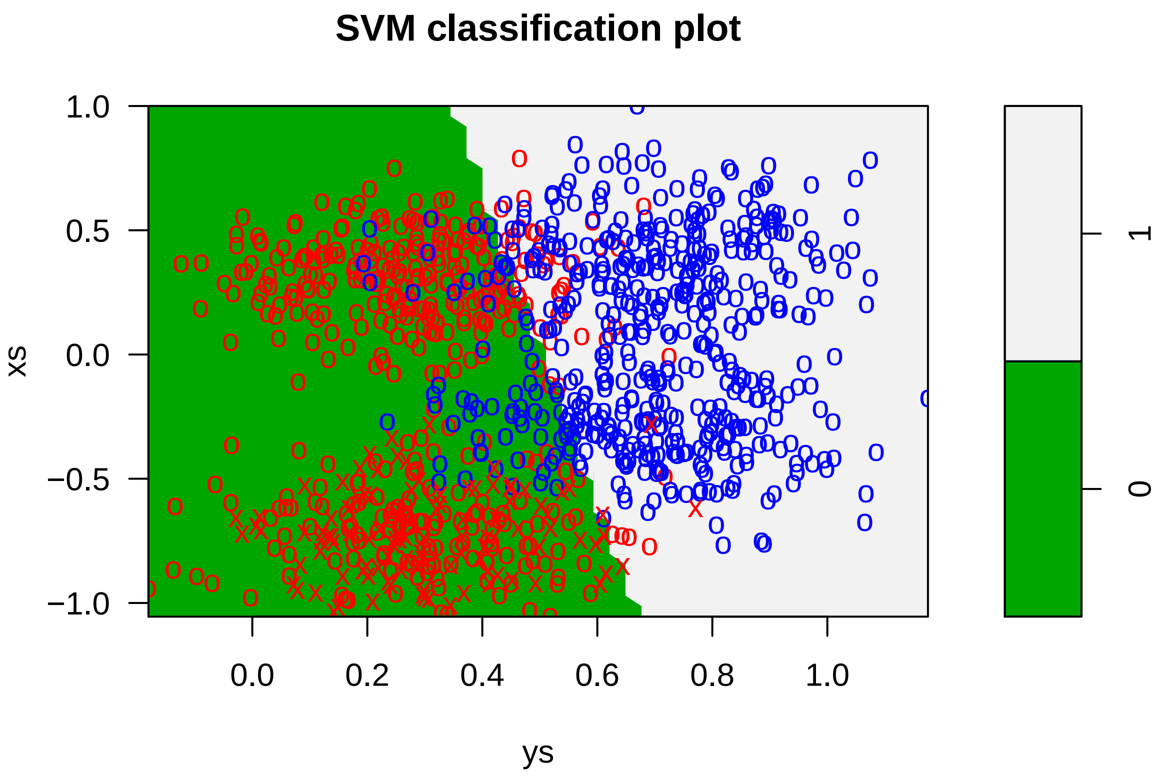


More importantly, we see that the model does similarly well on the test data set.

```
#Test on Test Data
linear_testing_pred <- predict(linear_svm,test)
#Confusion Matrix
table(linear_testing_pred, test$yc)
```

```
##
## linear_testing_pred   0   1
##                   0 451  54
##                   1  49 446
```

```
#Visual Representation
plot(linear_svm,test,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```



To see if more complicated basis kernels can yield better results, we turn next to the polynomial model. The steps from above are nearly repeated except that tuning now requires the tuning of several parameters. This turned out to be very computationally intensive and we have settled on fixing the cost at 0.2 (where it seems to have preferred for the linear model) and only varying the gamma and degree parameters. (We also used the default coefficient parameter.)

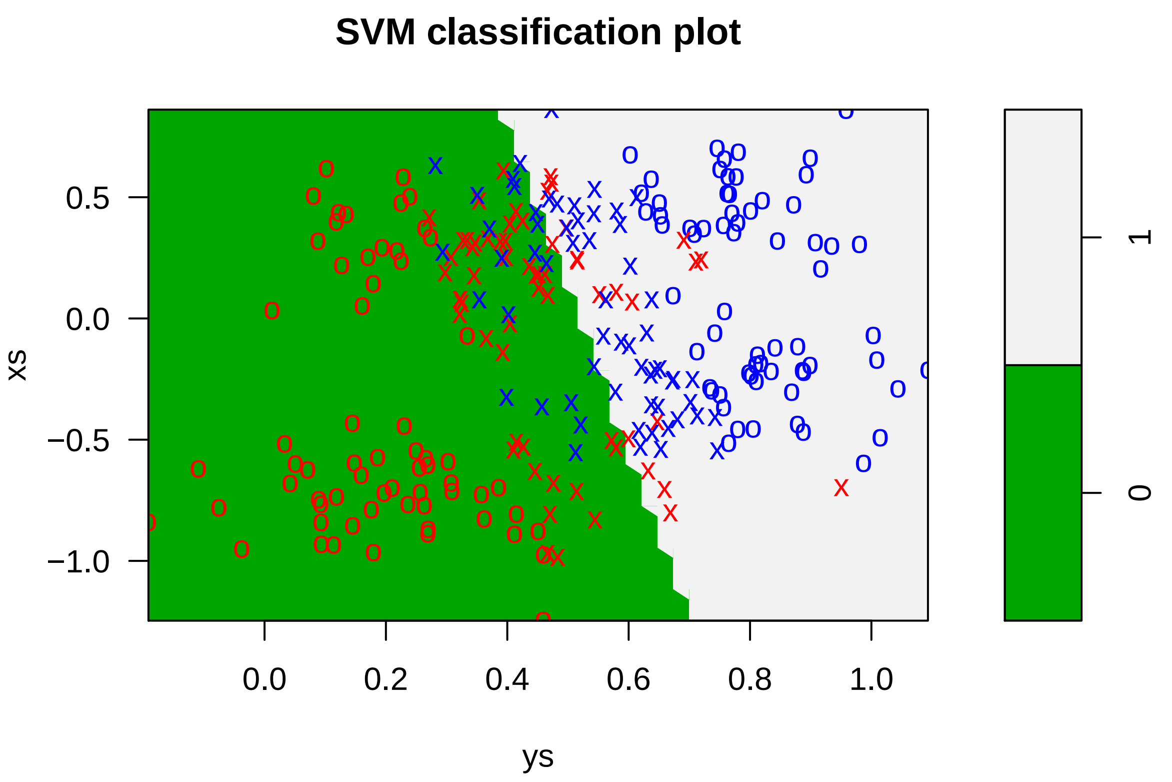
```
# Tune Parameters (gamma,degree)
polynomial_tune <- tune.svm(yc ~ ., data = train, kernel = "polynomial", degree=1:5,gamma=seq(0.2,5,0.2
# Cost
c <- 0.2
#degree
d <- polynomial_tune$performances[min(which(polynomial_tune$performances[4] == min(polynomial_tune$perf
#gamma
g <- polynomial_tune$performances[min(which(polynomial_tune$performances[4] == min(polynomial_tune$perf

# Train Polynomial Model
polynomial_svm <- svm(yc ~ ., data = train, kernel = "polynomial", cost=c, gamma=g, degree=d)
```

First testing on training data:

```
#Test on Training Data
polynomial_training_pred <- predict(polynomial_svm,train)
#Confusion Matrix
table(polynomial_training_pred, train$yc)
```

```
##
## polynomial_training_pred   0   1
##                        0 106  16
##                        1  19 109
```

```
#Visual Representation
plot(polynomial_svm,train,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```

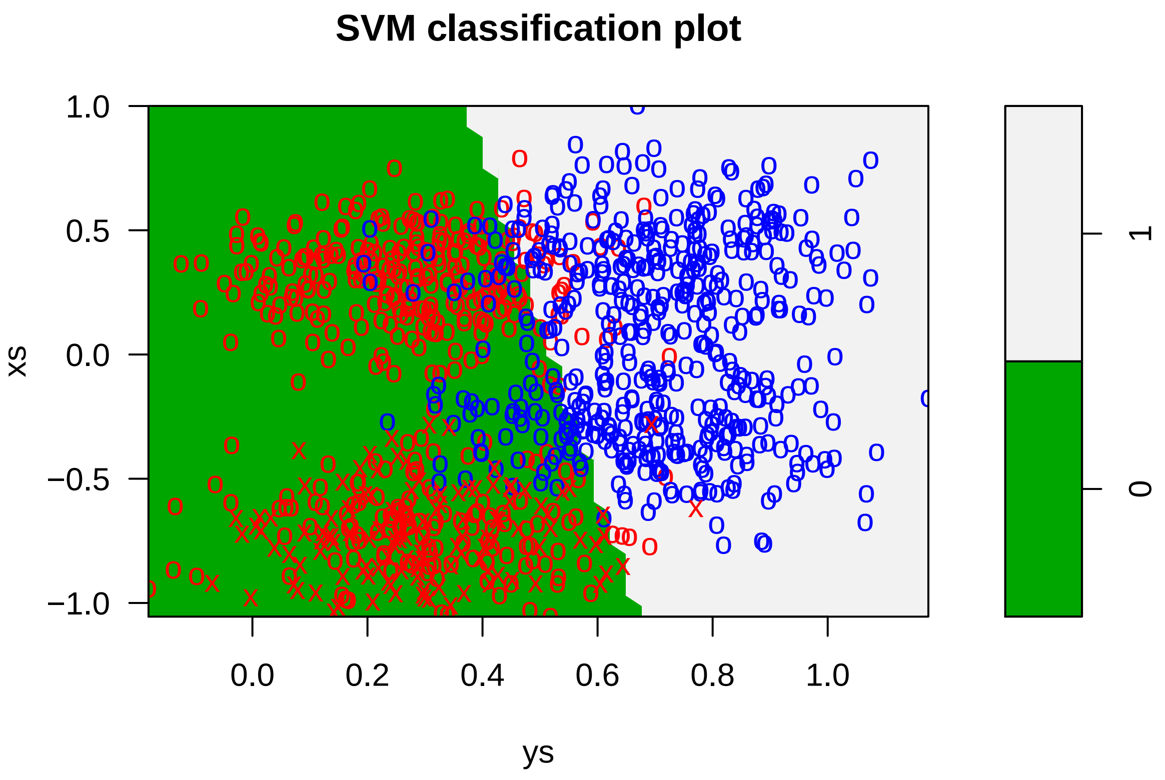


And then on testing data:

```
#Test on Test Data
polynomial_testing_pred <- predict(polynomial_svm,test)
#Confusion Matrix
table(polynomial_testing_pred, test$yc)
```

```
##
## polynomial_testing_pred   0   1
##                       0 457  67
##                       1  43 433
```

```
#Visual Representation
plot(polynomial_svm,test,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```



Comparing the polynomial model to the linear model, we see that they are very much the same. This should come at no surprise given that the tuning chose a polynomial of degree 1—i.e. a linear model. The only difference is that the polynomial model required a larger number of support vectors.

Moving on to the radial SVM model, we now have just the gamma parameter to tune (keeping the cost constant again)

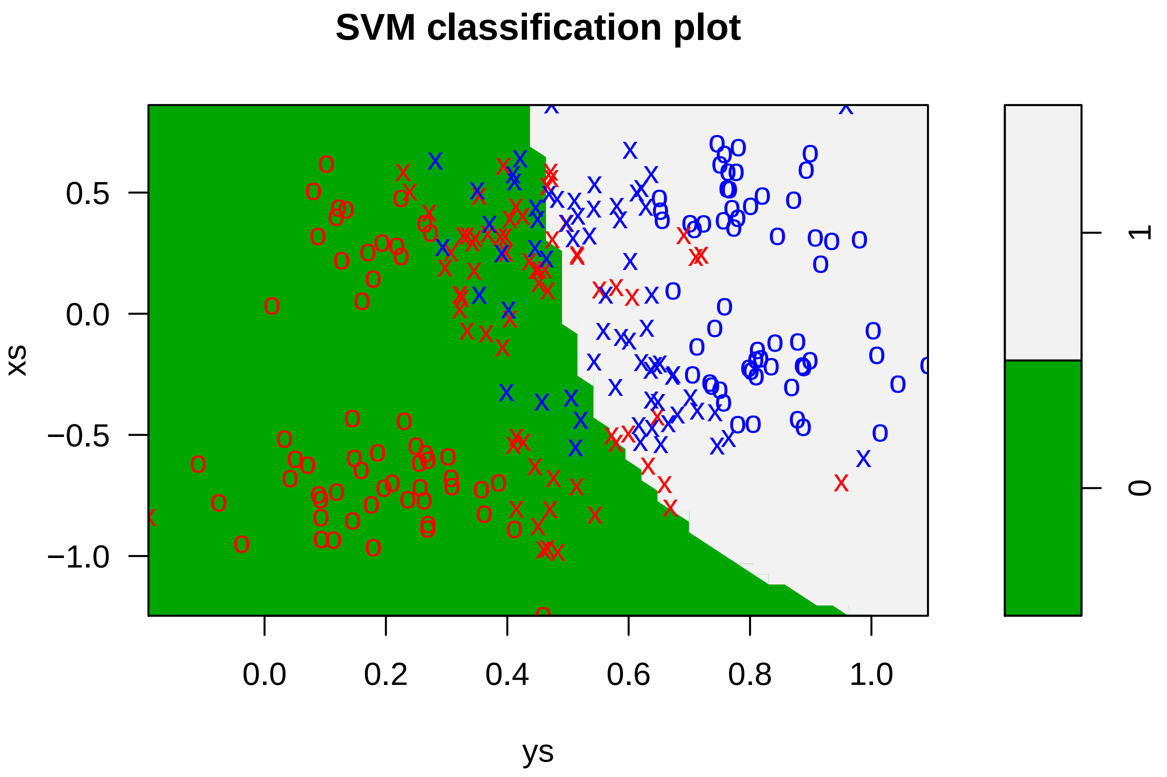
```r
# Tune Parameters (gamma)
radial_tune <- tune.svm(yc ~ ., data = train, kernel = "radial", gamma=seq(0.1,5,0.1),cost=0.2)
# Cost
c <- 0.2
#gamma
g <- radial_tune$performances[min(which(radial_tune$performances[4] == min(radial_tune$performances[4]))

# Train Radial Model
radial_svm <- svm(yc ~ ., data = train, kernel = "radial", cost=c, gamma=g)
```

Testing on the training set:

```r
#Test on Training Data
radial_training_pred <- predict(radial_svm,train)
#Confusion Matrix
table(radial_training_pred, train$yc)
```

```
##
## radial_training_pred   0   1
##                    0 104  19
##                    1  21 106
```

```r
#Visual Representation
plot(radial_svm,train,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```

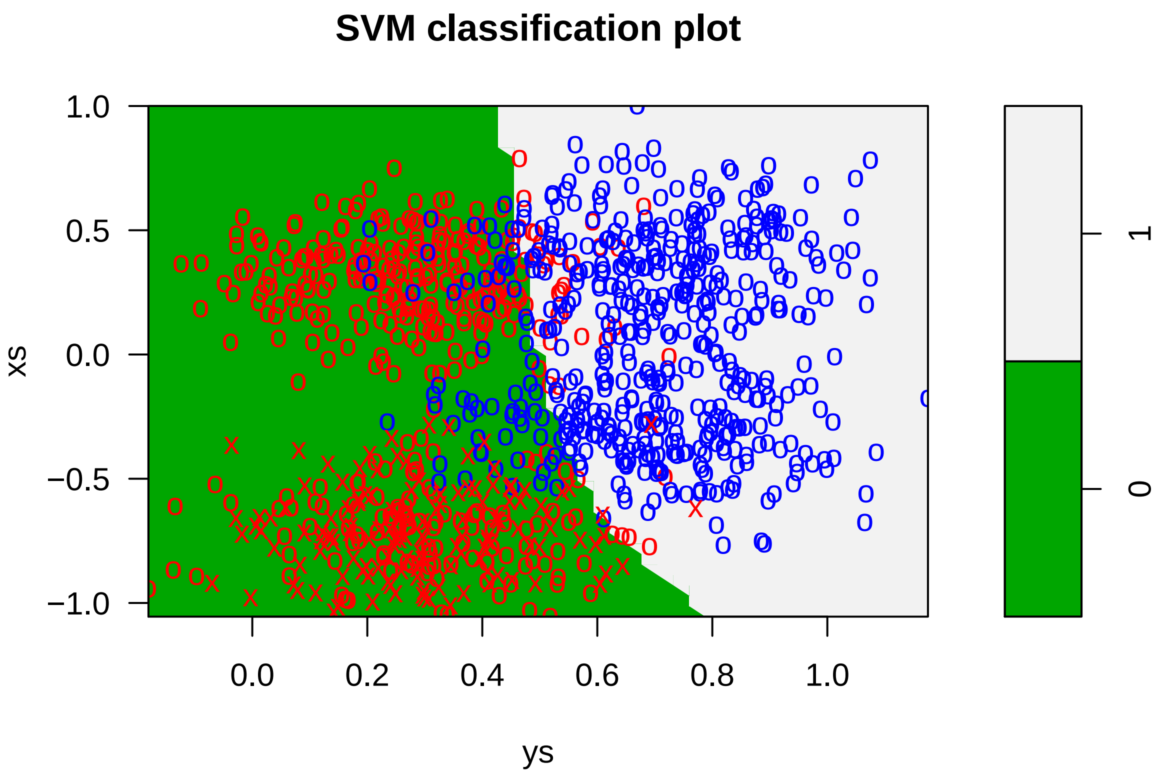


And on the test set:

```r
#Test on Test Data
radial_testing_pred <- predict(radial_svm,test)
#Confusion Matrix
table(radial_testing_pred, test$yc)
```

```
##
## radial_testing_pred   0   1
##                   0 457  58
##                   1  43 442
```

```r
#Visual Representation
plot(radial_svm,test,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```



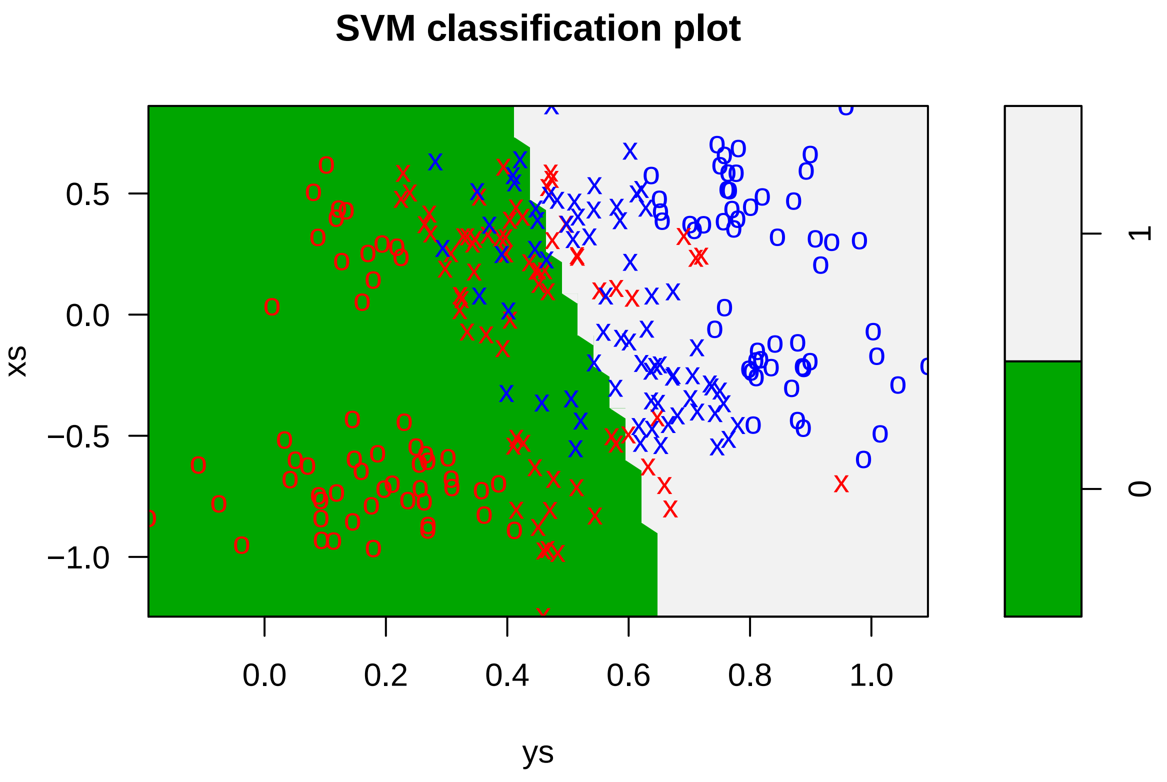
Though providing a distinctly different separator, the number of miscalculations the radial model produces is quite similar. It should be noted that when svm() isn't given a kernel to use for this dataset, it chooses the radial profile.

Lastly for the sigmoid model (jumping through the code all at once here):

```r
# Tune Parameters (gamma)
sigmoid_tune <- tune.svm(yc ~ ., data = train, kernel = "sigmoid", gamma=seq(0.1,5,0.1),cost=0.2)
# Cost
c <- 0.2
#gamma
g <- sigmoid_tune$performances[min(which(sigmoid_tune$performances[4] == min(sigmoid_tune$performances[

# Train Sigmoid Model
sigmoid_svm <- svm(yc ~ ., data = train, kernel = "sigmoid", cost=c, gamma=g)

#Test on Training Data
sigmoid_training_pred <- predict(sigmoid_svm,train)
#Confusion Matrix
table(sigmoid_training_pred, train$yc)
```

```
##
## sigmoid_training_pred   0   1
##                     0 106  16
##                     1  19 109
```

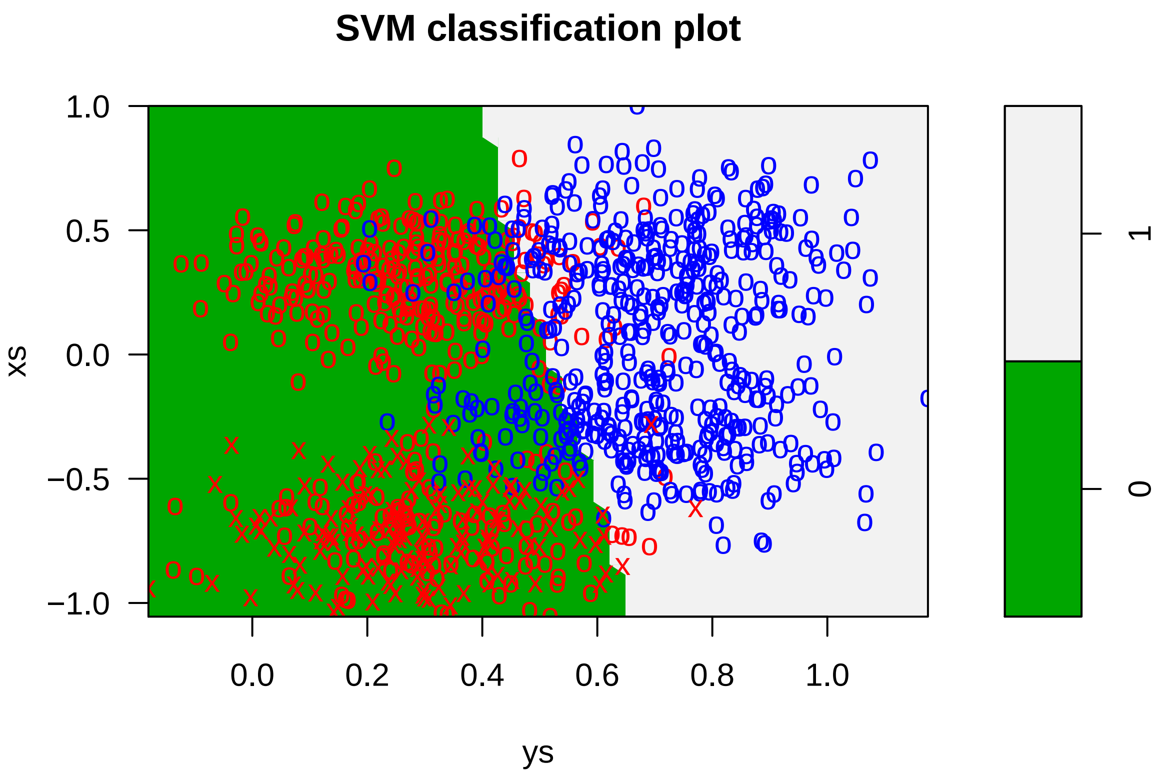
```r
#Visual Representation
plot(sigmoid_svm,train,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```



```r
#Test on Test Data
sigmoid_testing_pred <- predict(sigmoid_svm,test)
#Confusion Matrix
table(sigmoid_testing_pred, test$yc)
```

```
##
## sigmoid_testing_pred   0   1
##                    0 455  68
##                    1  45 432
```

```r
#Visual Representation
plot(sigmoid_svm,test,symbolPalette = c("red","blue"), color.palette = terrain.colors)
```



The sigmoid model performs (and appears) similar to the linear and polynomial models. Overall, the radial model might have a slightly lower misclassification rate, but with this set of data a linear model performs well. This exercise may have been more enlightening had the data set involved more than two dimensions. Nonetheless, the methods demonstrated here can be easily applied to the tuning and use of SVM for classification problems in higher dimensions.