```
- best parameters:
 cost gamma
  100   0.5

- best performance: 0.1

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.22 0.13984118
2  1e+00   0.5  0.11 0.12866839
3  1e+01   0.5  0.11 0.12866839
4  1e+02   0.5  0.10 0.12472191
5  1e+03   0.5  0.13 0.14181365
6  1e-01   1.0  0.22 0.13984118
7  1e+00   1.0  0.10 0.10540926
8  1e+01   1.0  0.11 0.12866839
9  1e+02   1.0  0.12 0.13165612
10 1e+03   1.0  0.16 0.10749677
11 1e-01   2.0  0.22 0.13984118
12 1e+00   2.0  0.10 0.12472191
13 1e+01   2.0  0.12 0.13165612
14 1e+02   2.0  0.17 0.10593499
15 1e+03   2.0  0.13 0.09486833
16 1e-01   3.0  0.22 0.13984118
17 1e+00   3.0  0.12 0.12292726
18 1e+01   3.0  0.13 0.10593499
19 1e+02   3.0  0.13 0.08232726
20 1e+03   3.0  0.18 0.11352924
21 1e-01   4.0  0.22 0.13984118
22 1e+00   4.0  0.13 0.12516656
23 1e+01   4.0  0.14 0.10749677
24 1e+02   4.0  0.14 0.08432740
25 1e+03   4.0  0.18 0.11352924
```

Therefore, the best choice of parameters involves cost = 100 and gamma = 0.5. We can view the test set predictions for this model by applying the predict() function to the data.

```
> table(
+   true = dat[-train, "y"], pred = predict(
+     tune_out$best.model, newdata = dat[-train, ]
+     )
+   )
    pred
true   1  2
   1 69  3
   2 13 15
```

test err. rate for this SVM $= \frac{16}{100}$

ROC Curves:

The ROCR package can be used to produce ROC curves. We first write a short function to plot an ROC curve and calculate the associated AUC given a vector containing a numerical score for each observation, pred, and a vector containing the class label for each observation, truth.

```
> library(ROCR)
> rocplot_auc <- function(pred, truth, ...) {
+    predob <- prediction(pred, truth)
+    perf <- performance(predob, "tpr", "fpr")
+    plot(perf, ...)
+    auc <- performance(predob,"auc")
+    paste("auc is:", auc@y.values[[1]])
+ }
```

SVMs and support vector classifiers output class labels for each observation. However, it is also possible to obtain fitted values for each observation, which are the numerical scores used to obtain the class labels. In order to obtain the fitted values for a given SVM model fit, we use decision.values = TRUE when fitting svm(). Then the predict() function will output the fitted values.

```
> svmfit_opt <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 2,
   cost = 1, decision.values = T)
```

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

```
> fitted <- attributes(
+   predict(svmfit_opt, dat[train, ], decision.values = TRUE)
+ )$decision.values
```

Now we can produce the ROC plot. Note we use the negative of the fitted values so that negative values correspond to class 1 and positive values to class 2.

```
> par(mfrow = c(1, 2))
> rocplot_auc(-fitted, dat[train, "y"], main = "Training Data", col = "blue")
[1] "auc is: 0.958624708624708"
```

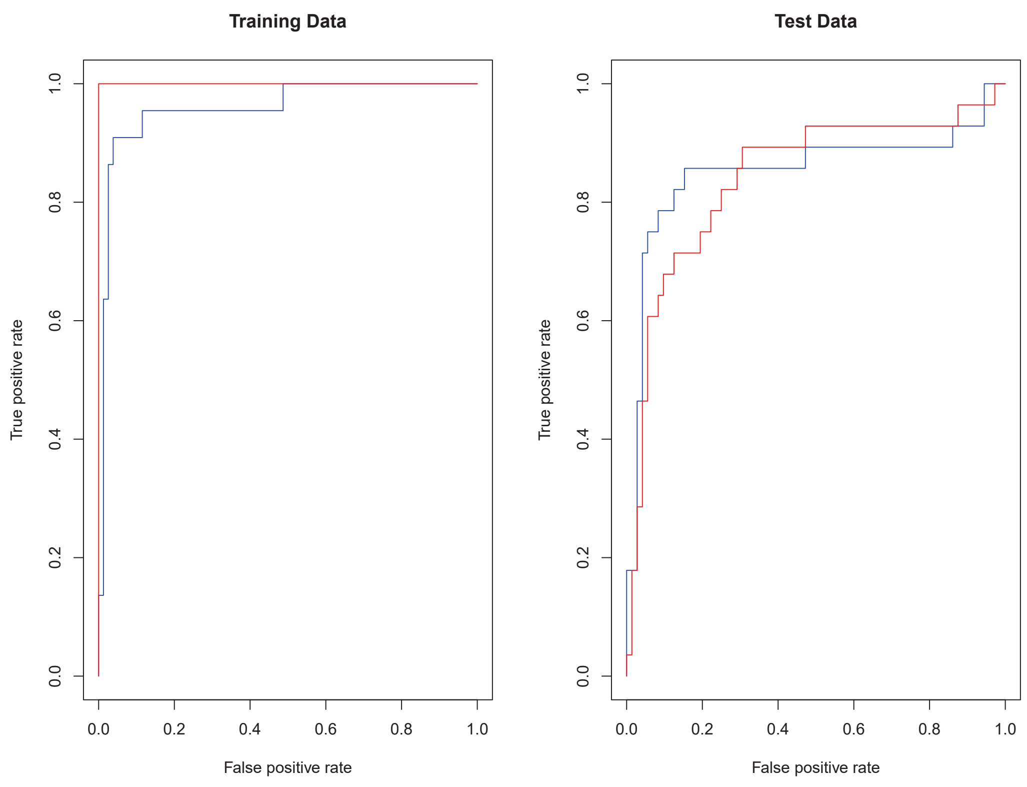
SVM appears to be producing accurate predictions. By increasing γ we can produce a more flexible fit and generate further improvements in accuracy.

```
> svmfit_flex <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 50,
   cost = 1, decision.values = T)
> fitted <- attributes(
+   predict(svmfit_flex, dat[train, ], decision.values = T)
+ )$decision.values
> rocplot_auc(-fitted, dat[train, "y"], add = T, col = "red")
[1] "auc is: 1"
```

However, these ROC curves are all on the training data. We are really more interested in the level of prediction accuracy on the test data. When we compute the ROC curves on the test data, the model with $\gamma = 2$ appears to provide more accurate results.

```
> fitted <- attributes(
+   predict(svmfit_opt, dat[-train, ], decision.values = T)
+ )$decision.values
> rocplot_auc(-fitted, dat[-train, "y"], main = "Test Data", col = "blue")
[1] "auc is: 0.851686507936508"
> fitted <- attributes(
+   predict(svmfit_flex, dat[-train, ], decision.values = T) )$decision.values
```

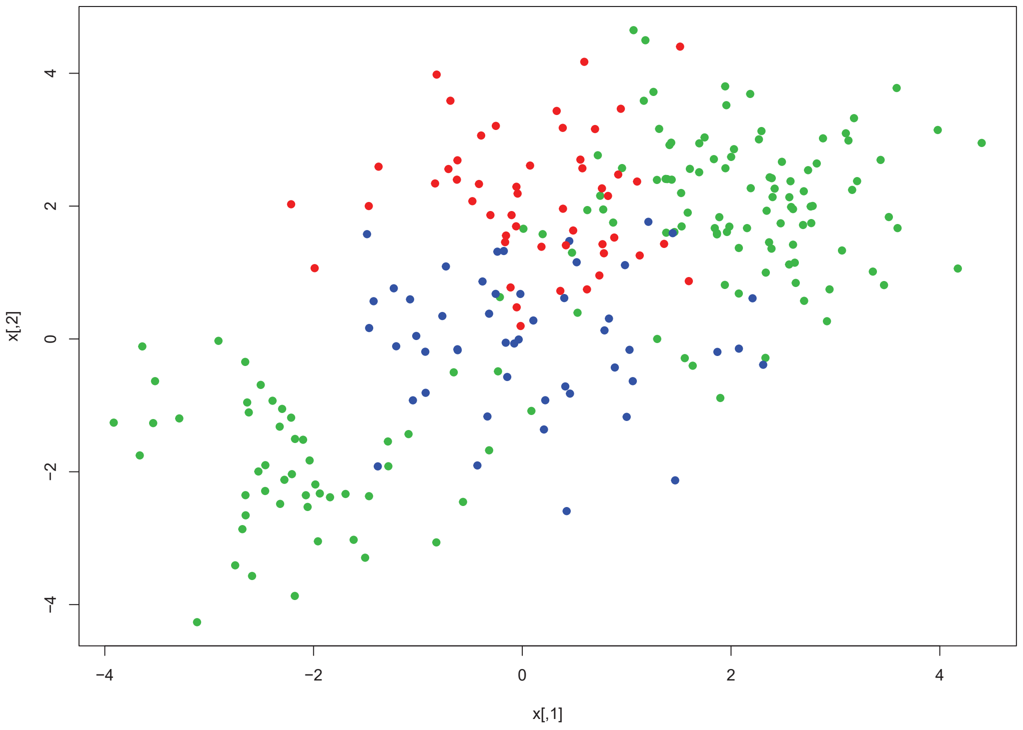
```
> rocplot_auc(-fitted, dat[-train, "y"], add = T, col = "red")
[1] "auc is: 0.84077380952381"
```



SVM with Multiple Classes:

If the response is a factor containing more than two levels, then the svm() function will perform multi-class classification using the one-versus-one approach. We explore that setting here by generating a third class of observations.

```
> set.seed(1)
> x <- rbind(x, matrix(rnorm(50 * 2), ncol = 2))
> y <- c(y, rep(0, 50))
> x[y == 0, 2] <- x[y == 0, 2] + 2
> dat <- data.frame(x = x, y = as.factor(y))
> par(mfrow = c(1, 1))
> plot(x, col = (y + 2), pch = 19)
```

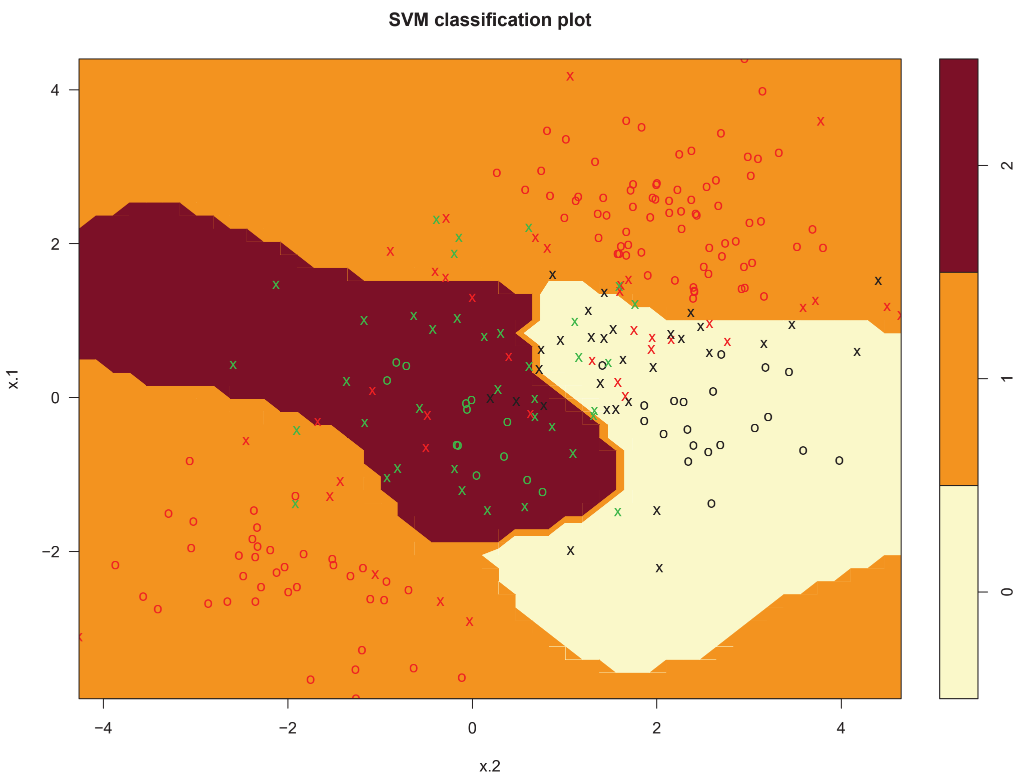


We now fit an SVM with radial kernel to the data:

```
> svmfit_radial <- svm(y ~ ., data = dat, kernel = "radial", cost = 10,
   gamma = 1)
> plot(svmfit_radial, dat)
> table(predict = svmfit_radial$fitted, truth = dat$y)
```

```
       truth
predict   0   1   2
      0  42   8   6
      1   4 134   8
      2   4   8  36
```

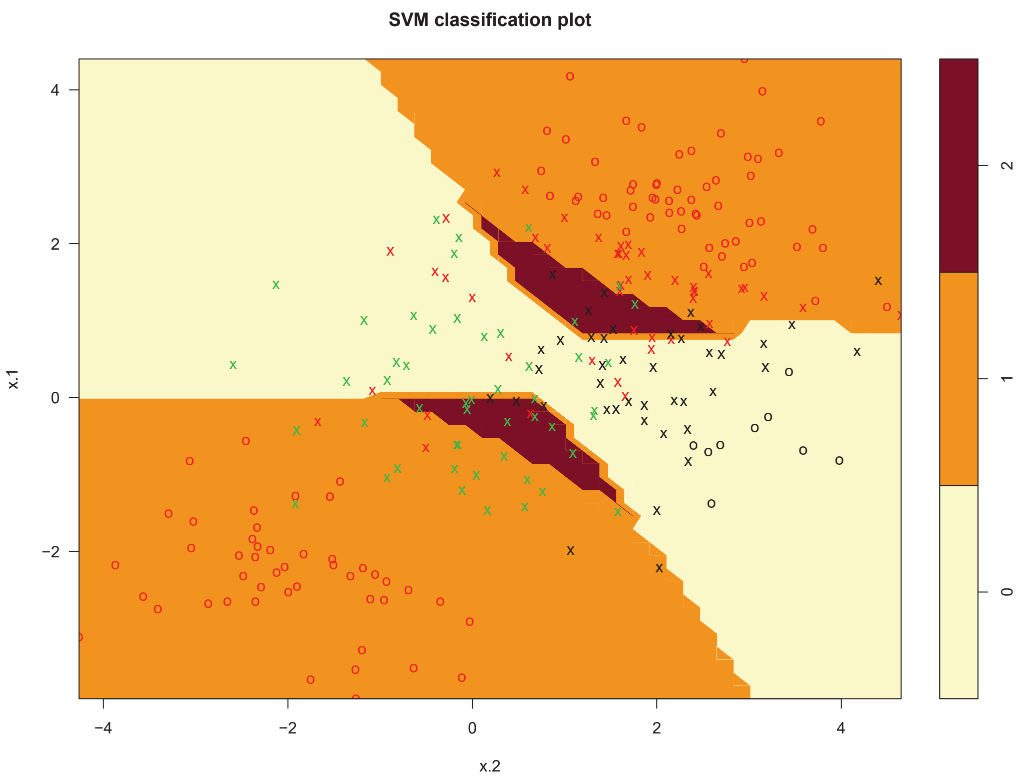
38 training errors for this SVM



We also fit an SVM with polynomial kernel to the data:

```
> svmfit_poly <- svm(y ~ ., data = dat, kernel = "polynomial", cost = 10,
   degree=2)
> plot(svmfit_poly, dat)
> table(predict = svmfit_poly$fitted, truth = dat$y)
       truth
predict   0   1   2
      0  36  13  20
      1   4 130  16
      2  10   7  14
```

For this svm there are 70 training errors



Application to Gene Expression Data:

We now examine the Khan data set, which consists of a number of tissue samples corresponding to four distinct types of small round blue cell tumors. For each tissue sample, gene expression measurements are available. The data set consists of training data, xtrain and ytrain, and testing data, xtest and ytest.

We examine the dimension of the data:

```
> library(ISLR2)
> names(Khan)
[1] "xtrain" "xtest"  "ytrain" "ytest"
> dim(Khan$xtrain)
[1]   63 2308
```

```
> dim(Khan$xtest)
[1]   20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
```

This data set consists of expression measurements for 2,308 genes. The training and test sets consist of 63 and 20 observations respectively.

```
> table(Khan$ytrain)

 1  2  3  4
 8 23 12 20

> table(Khan$ytest)

1 2 3 4
3 6 6 5
```

We will use a support vector approach to predict cancer subtype using gene expression measurements. In this data set, there are a very large number of features relative to the number of observations. This suggests that we should use a linear kernel, because the additional flexibility that will result from using a polynomial or radial kernel is unnecessary.

```
> dat <- data.frame( x = Khan$xtrain, y = as.factor(Khan$ytrain) )
> out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)
> summary(out)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
```

```
       cost:  10

Number of Support Vectors:  58

 ( 20 20 11 7 )


Number of Classes:  4

Levels:
 1 2 3 4

> table(predict =  out$fitted, truth = dat$y)
       truth
predict  1  2  3  4
      1  8  0  0  0
      2  0 23  0  0
      3  0  0 12  0
      4  0  0  0 20
```

We see that there are no training errors. In fact, this is not surprising, because the large number of variables relative to the number of observations implies that it is easy to find hyperplanes that fully separate the classes. We are most interested not in the support vector classifier's performance on the training observations, but rather its performance on the test observations.

```
> dat_test <- data.frame( x = Khan$xtest, y = as.factor(Khan$ytest))
> pred_test <- predict(out, newdata = dat_test)
> table(predict = pred_test, truth = dat_test$y)

       truth
predict 1 2 3 4
      1 3 0 0 0
      2 0 6 2 0
      3 0 0 4 0
      4 0 0 0 5
```

We see that using cost = 10 yields two test set errors on this data.