STATS 415 Lab 11

Nick Seewald

March 30, 2018

1 Today's objectives

1. Learn how to implement a support vector classifier and a support vector machine with two classes
2. Extend SVM to multiple classes

2 Support Vector Classifier

Our goal is to build a classifier which classifies an observation based on which side of a hyperplane it lies on. The *support vector classifier* solves the following optimization problem:

$$\begin{aligned} &\max_{\beta_0,\ldots,\beta_p,\xi_1,\ldots,\xi_n} M \\ &\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \\ &\quad y_i\left(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}\right) \geq M(1-\xi_i), \\ &\quad \xi_i \geq 0,\ \sum_{i=1}^{p} \xi_i \leq B. \end{aligned}$$

The goal, then, is to find the largest margin M, allowing for some violations, controlled by the "budget" C.

- The ξ_i are "slack variables", the value of which tells us where the ith observation is located relative to the hyperplane and the margin: if $\xi_i = 0$, the ith point is on the correct side of the margin; if $0 < \xi_i < 1$ the point *violates* the margin (is on the wrong side of the margin); if $\xi_i > 1$ the point is on the wrong side of the hyperplane.
- B is a "budget" for the amount that the margin can be violated. If $B = 0$, the support vector classifier is the maximal margin classifier If $B > 0$, at most B points can be misclassified. B is generally chosen via cross-validation.

2.1 R Implementation

We'll use the `svm()` function from the `e1071` library:

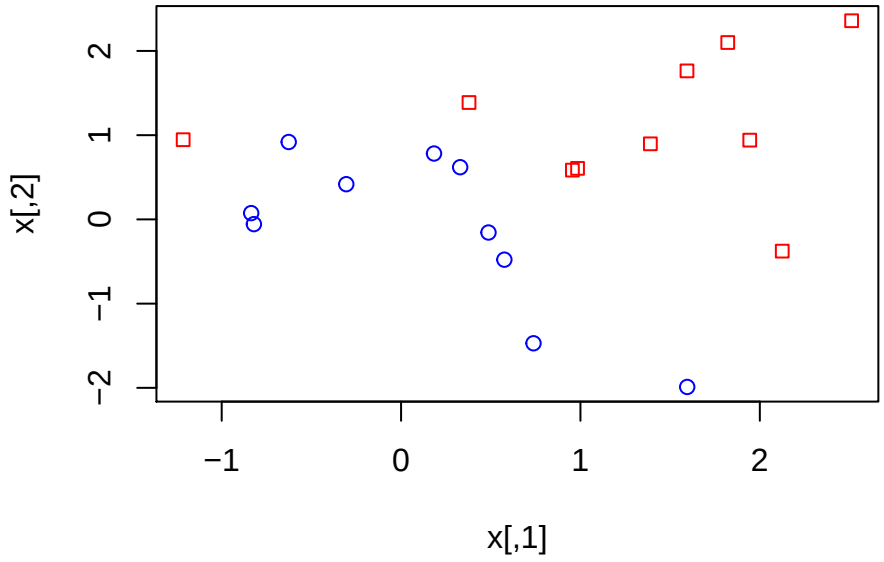
```
library(e1071)
```

`svm()` solves a different, but equivalent, formulation of the problem above, and uses a `cost` argument to specify the severity of a violation of the margin (see slide 12 in the SVM notes). High `cost` will enforce narrow margins and result in few support vectors on the margin or violating it; low `cost` will lead to large margins and many support vectors on or violating the margin.

2.1.1 Example 1:

Let's start by generating some data. **Note:** We're generating this data specifically for exploratory purposes for this lab, to achieve specific properties. Real data analysis does not involve generating data like this.

```
set.seed(1)
x <- matrix(rnorm(20*2), ncol = 2)  # 20x2 matrix of draws from a standard normal dist'n
y <- c(rep(-1, 10), rep(1, 10))
x[y==1, ] <- x[y==1, ] + 1 # Add 1 to x when y = 1
plot(x, col = (3-y), pch = c(21, 22)[as.factor(y)])
```



Question: Are the classes linearly separable?

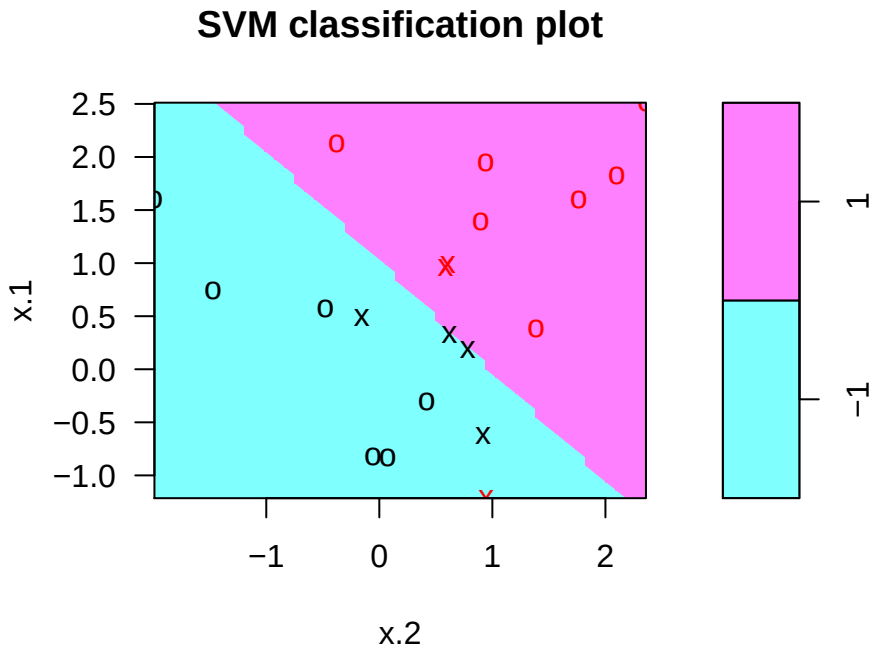
Now let's fit a support vector classifier. *In order to do classification, the response must be a factor.*

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

- `svm()` will fit a support vector classifier when the `kernel="linear"` argument is used
- `scale = FALSE` tells `svm()` not to scale each predictor to have mean zero and standard deviation one. *Depending on the application, you might want to set* `scale = T`: This is an example in which we are choosing not to scale (in particular, because `x` consists of draws from the *standard* normal distribution). You should think carefully about your particular goals when running this on the homework and deciding whether or not to scale the data.

Let's plot the classifier we've fit:

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



Question: How many points were misclassified? How many support vectors are there?

We can figure out which observations are the support vectors by looking at the `index` element of `svmfit` and get basic summary information using `summary()`:

```
svmfit$index
```

```
## [1]  1  2  5  7 14 16 17
```

```
summary(svmfit)
```

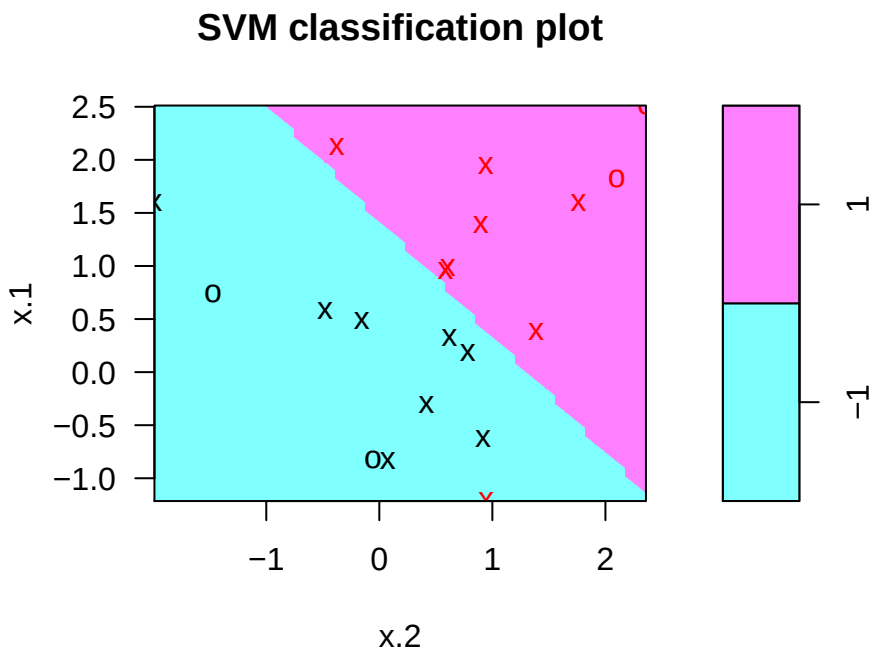
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  7
##
```

```
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Question: How many support vectors were in each class?

Let's try changing the value of `cost`:

```
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
```

Question: Is the margin wider or narrower than when `cost=10`? How can you tell?

Note: `svm()` does *not* explicitly output the coefficients of the linear decision boundary or the width of the margin.

Let's now cross-validate the `cost`. To do this, we'll use the built-in `tune()` function from `e1071`.

```
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)))
```

- The first argument of `tune()` is called `method`; here, we want to use `svm()`, since that's the function we want to tune.
- Subsequent arguments are passed to `svm()`
- `ranges` is a named list of parameter vectors, the values of which you want to use in the `method`.

```
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
```

Question: Which `cost` led to the lowest cross-validation error rate?

We can access the best model found by `tune`:

```
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
##       gamma:  0.5
```

```
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Now let's generate some test data. **Note:** The test data is generated from the same process as before. In facto, if we used `set.seed(1)` before this code, we would regenerate the training data. (This is why we're not setting the seed).

```
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))
```

We now predict class labels of the test observations using the best model as obtained from cross-validation:

```
ypred <- predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
```

```
##        truth
## predict -1  1
##      -1 11  1
##      1   0  8
```

Question: With the cross-validated choice of `cost`, how many observations are correctly classified?

What if we used `cost = 0.01` instead?

```
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 0.01, scale = FALSE)
ypred <- predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)
```

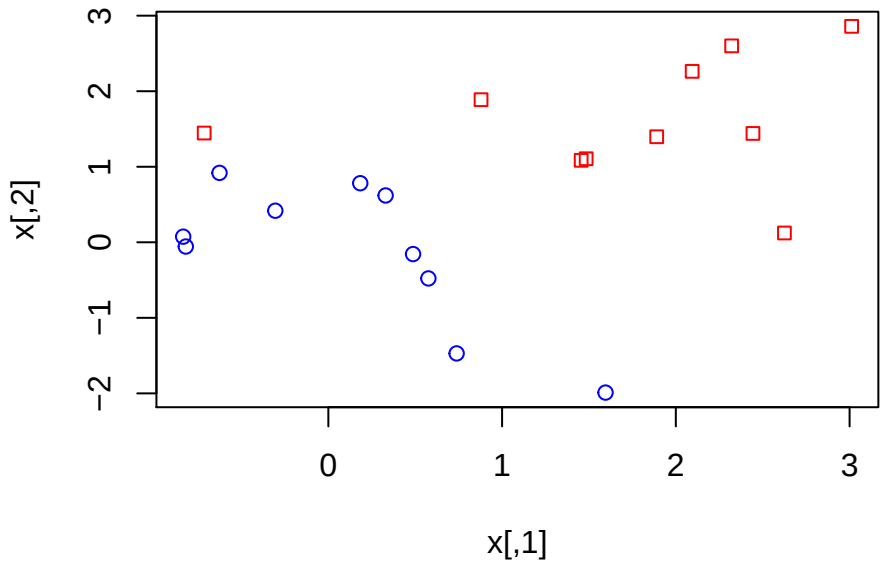
```
##        truth
## predict -1  1
##      -1 11  2
##      1   0  7
```

Question: Now how many observations are correctly classified?

2.1.2 Example 2

Let's modify our data a bit:

```
x[y == 1, ] <- x[y == 1, ] + 0.5
plot(x, col = (3-y), pch = c(21, 22)[as.factor(y)])
```



Question: Now are the observations linearly separable?

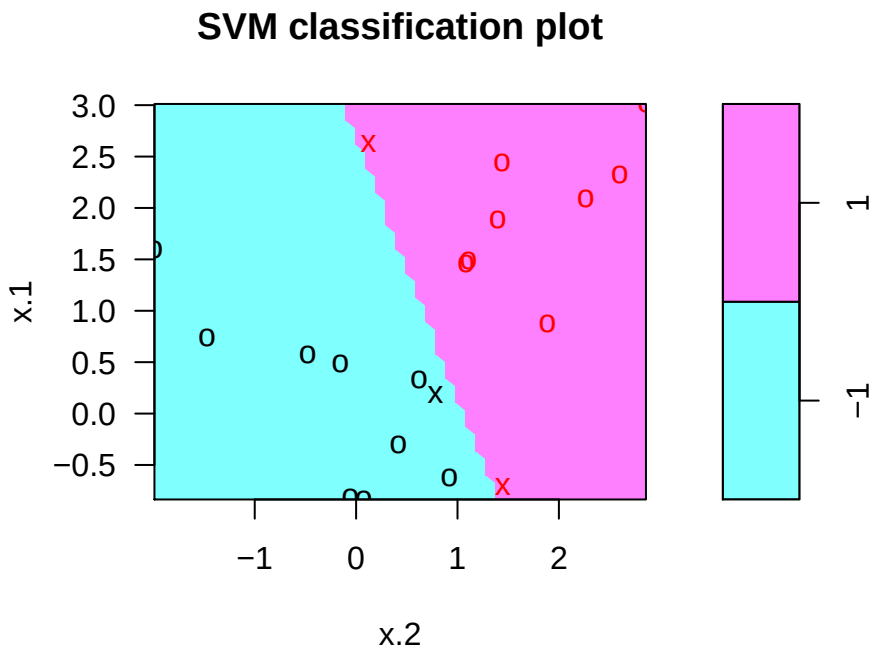
We can now fit a support vector classifer with a very large value of `cost` so that no observations are misclassified.

```
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
##       gamma:  0.5
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
```

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

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



Question: How many training errors were made?

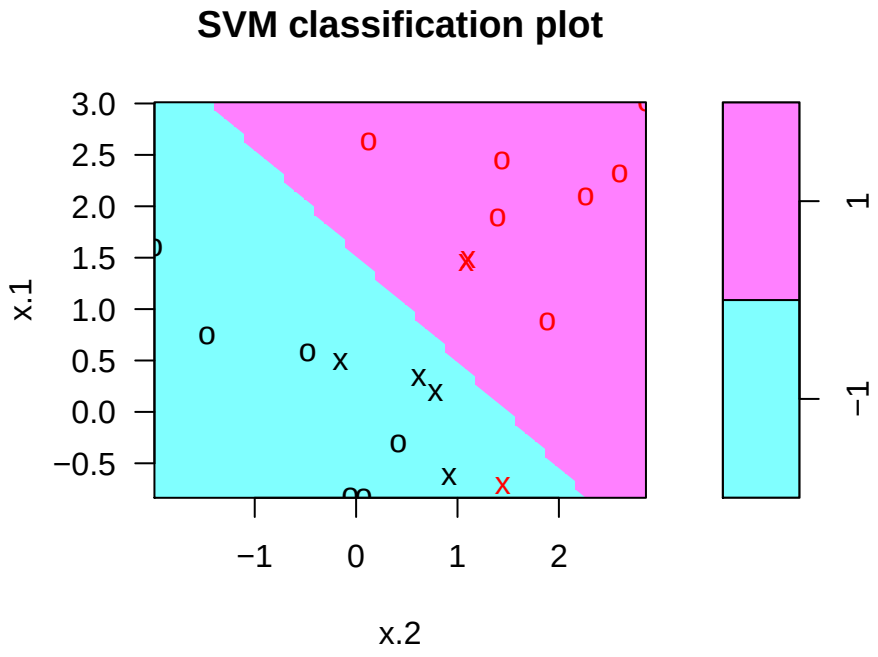
Question: Is the margin wide? Narrow? How can you tell? How do you think this will perform on test data?

Let's try reducing `cost`.

```
dat <- data.frame(x = x, y = as.factor(y))
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
##       gamma:  0.5
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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



Question: How many training errors were made?

Question: Is the margin wide? Narrow? How can you tell? How do you think this will perform on test

data?

3 Support Vector Machines

The support vector machine generalizes the support vector classifier to non-linear boundaries by enlarging the feature space with polynomials, interactions, or other functions of the predictors, in an efficient way. To do this, we consider *kernel functions*.

A *kernel function* $K(x, x^*)$ generalizes the notion of an inner product (e.g., the dot product) and is used to quantify the similarity between two observations. For popular examples of kernel functions, see slide 24 of the SVM notes. We'll focus on two: polynomial and radial kernels.

- The dth degree polynomial kernel is $K(x, x^*) = (1 + \langle x, x^* \rangle)^d$
- The radial basis kernel is $K(x, x^*) = \exp\left(-\gamma \|x - x^*\|^2\right)$

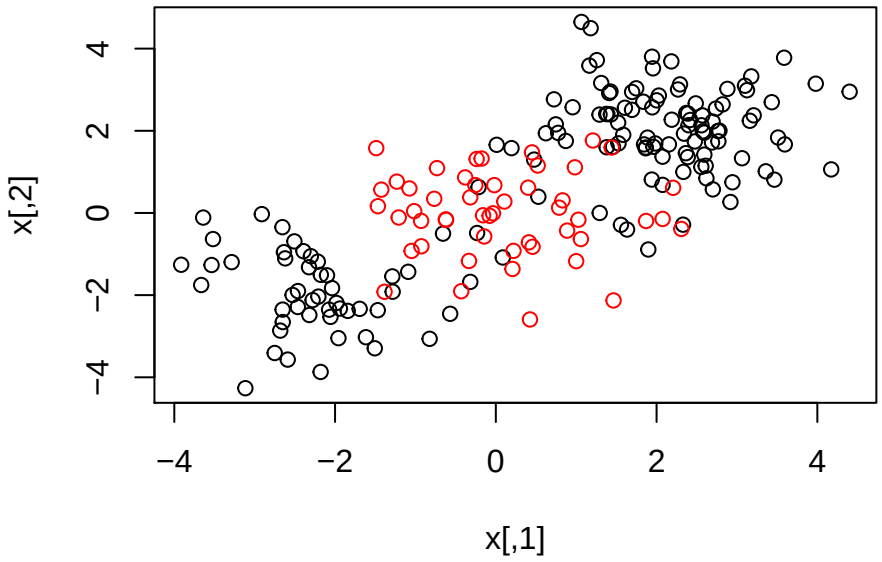
Note: The radial basis kernel defined by R is different than the one defined in the notes! Above is the kernel as defined *by R*. To achieve the one in the notes, set $\gamma = 1/\sigma^2$.

3.1 R implementation

We still use `svm()`. To implement a polynomial kernel, we use `kernel = polynomial` combined with a `degree` argument; for a radial basis kernel, we use `kernel = radial` and a `gamma` argument.

We start by generating data with a non-linear class boundary:

```
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
plot(x, col = y)
```



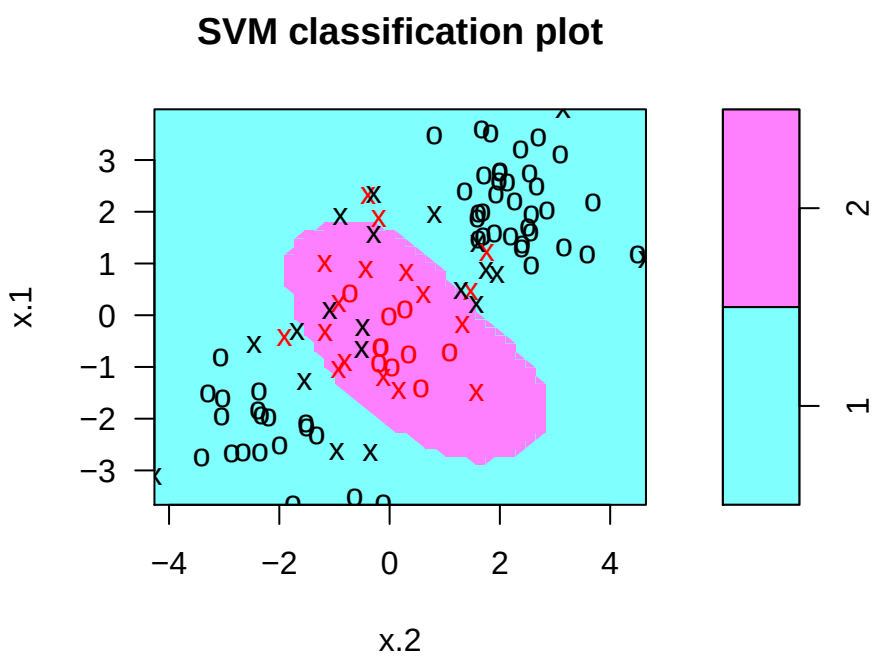
Question: How did we know the data would have a non-linear decision boundary?

Split the data into training and test sets:

```
train <- sample(200, 100)
```

We'll fit an SVM to the training data using a radial kernel with $\gamma = 1$.

```
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)
plot(svmfit, dat[train, ])
```




```
summary(svmfit)
```

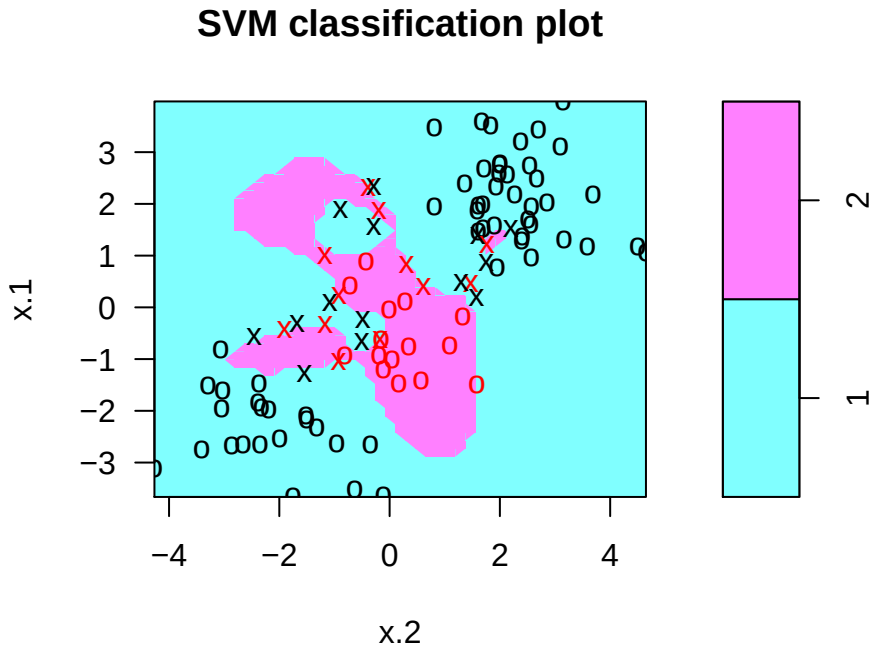
```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
##     gamma = 1, cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  37
##
##  ( 17 20 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

Question: Heuristically, how did this model perform on the training data?

Question: How might we reduce the number of training errors? At what cost?

Let's try our strategy from the above question.

```
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1e5)
plot(svmfit, dat[train, ])
```



Question: Does this irregular decision boundary worry you? Why?

overfit cost太大margin太小

Let's cross-validate to choose the best choice of `cost` and `gamma`.

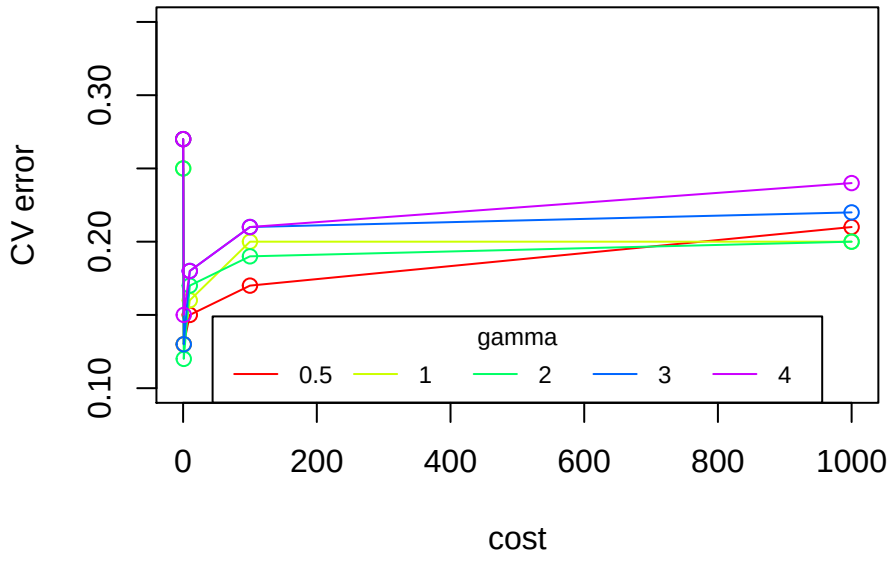
```
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ], kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                               gamma = c(0.5, 1, 2, 3, 4)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
```

```
##  cost gamma
##     1     2
##
## - best performance: 0.12
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.27 0.11595018
## 2  1e+00   0.5  0.13 0.08232726
## 3  1e+01   0.5  0.15 0.07071068
## 4  1e+02   0.5  0.17 0.08232726
## 5  1e+03   0.5  0.21 0.09944289
## 6  1e-01   1.0  0.25 0.13540064
## 7  1e+00   1.0  0.13 0.08232726
## 8  1e+01   1.0  0.16 0.06992059
## 9  1e+02   1.0  0.20 0.09428090
## 10 1e+03   1.0  0.20 0.08164966
## 11 1e-01   2.0  0.25 0.12692955
## 12 1e+00   2.0  0.12 0.09189366
## 13 1e+01   2.0  0.17 0.09486833
## 14 1e+02   2.0  0.19 0.09944289
## 15 1e+03   2.0  0.20 0.09428090
## 16 1e-01   3.0  0.27 0.11595018
## 17 1e+00   3.0  0.13 0.09486833
## 18 1e+01   3.0  0.18 0.10327956
## 19 1e+02   3.0  0.21 0.08755950
## 20 1e+03   3.0  0.22 0.10327956
## 21 1e-01   4.0  0.27 0.11595018
## 22 1e+00   4.0  0.15 0.10801234
## 23 1e+01   4.0  0.18 0.11352924
## 24 1e+02   4.0  0.21 0.08755950
## 25 1e+03   4.0  0.24 0.10749677
```

That summary is tough to read. Let's plot the training errors as a function of cost, with one line per value of gamma:

```
with(tune.out$performances, {
  plot(error[gamma == 0.5] ~ cost[gamma == 0.5], ylim = c(.1, .35),
       type = "o", col = rainbow(5)[1], ylab = "CV error", xlab = "cost")
  lines(error[gamma == 1] ~ cost[gamma == 1],
       type = "o", col = rainbow(5)[2])
  lines(error[gamma == 2] ~ cost[gamma == 2],
       type = "o", col = rainbow(5)[3])
  lines(error[gamma == 3] ~ cost[gamma == 3],
       type = "o", col = rainbow(5)[4])
  lines(error[gamma == 4] ~ cost[gamma == 4],
       type = "o", col = rainbow(5)[5])
})
legend("bottom", horiz = T, legend = c(0.5, 1:4), col = rainbow(5),
       lty = 1, cex = .75, title = "gamma")
```




```
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat[train, ],
##     ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5,
##         1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  2
##
## Number of Support Vectors:  43
##
##  ( 17 26 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

Question: What is the best choice of parameters according to 10-fold cross-validation?

Let's examine test error for the model chosen by CV.

```
table(true = dat[-train, "y"], pred = predict(bestmod, newdata = dat[-train, ]))
```

```
##     pred
## true  1  2
##    1 74  3
##    2  7 16
```

Question: What's the test error for this model?

4 SVM with Multiple Classes

The idea of a separating hyperplane doesn't lend itself well to classification in settings with more the one class. To extend SVM to multiple classes, we consider the *one-versus-one* approach.

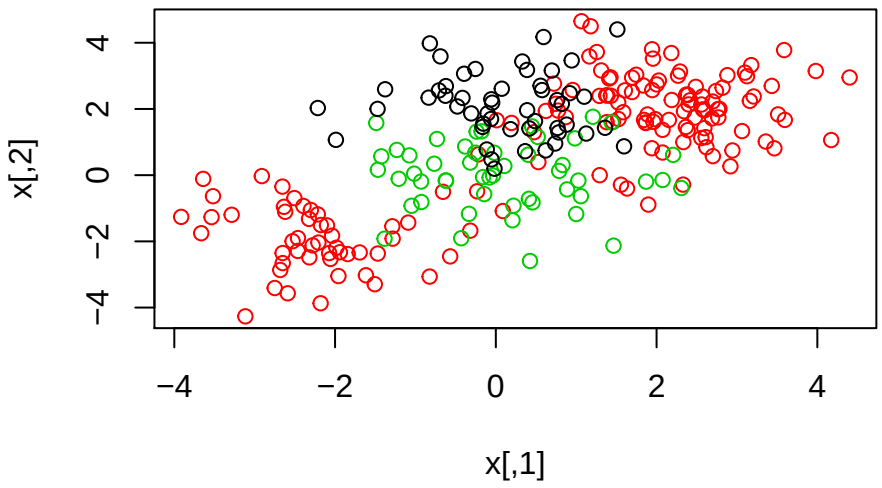
4.1 One-Versus-One Classification

Suppose we have K classes, $K > 2$. The idea is to fit $\binom{K}{2}$ SVMs, each of which compares two of the classes. We'll classify a test observation using each of the classifiers, and count the number of times the it's assigned to each class. The class to which it was most frequently assigned is our final classification.

4.2 R implementation

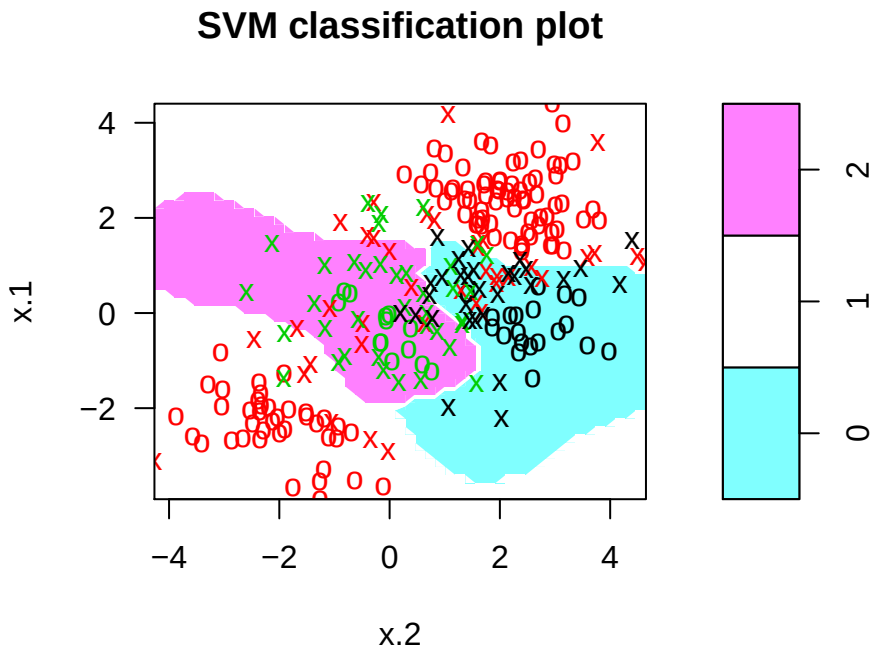
Let's generate an additional 100 data points which belong to a third class.

```
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))
plot(x, col = (y+1))
```



Now we fit an SVM as usual:

```
svmfit <- svm(y ~ ., data = dat, kernel = "radial", cost = 10, gamma = 1)
plot(svmfit, dat)
```



Note: We didn't split the data into training and test for this last example. This is just for illustrative purposes.