Classifying Cancer Subtype with Support Vector Machines

Our Problem

- Doctors diagnose what kind of cancer their patients have using genetic tests. Can we use data to help doctors classify cancer type?
- multiclass classification rather than binary
- **class boundaries may not be linear**
- use Support Vector Machines (SVM)

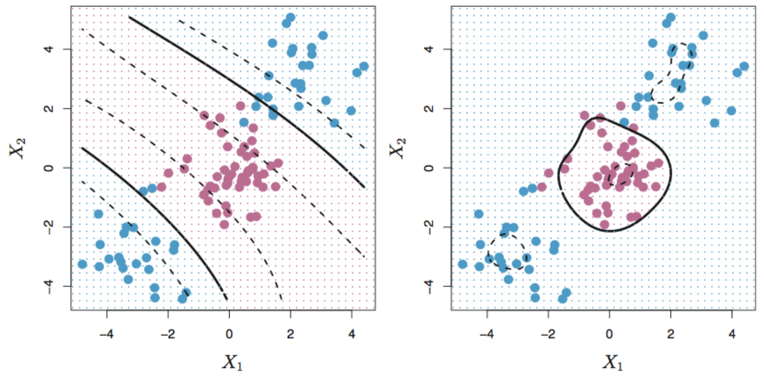

FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

How do SVMs work?

A simple example: maximal margin hyperplane

- a maximal margin hyperplane is a line that has the farthest minimum distance to the training observations.
- This allows us to separate observations into different classes

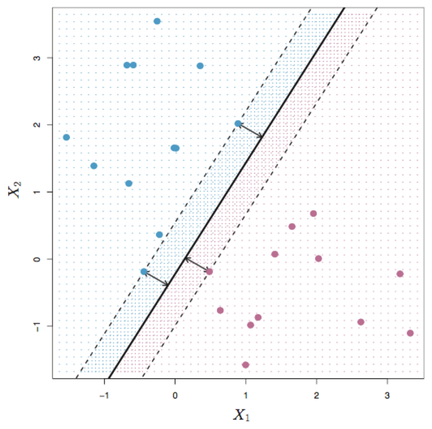

FIGURE 9.3. *There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the margin is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.*

Extending our simple example: using kernels

- A kernel calculates the similarity of two observations.
- A linear kernel will give us a decision boundary like the one above.
- A **polynomial kernel** will give us a more flexible decision boundary. Hence we can apply the kernel to non-linear data.
- Kernels are only restricted to polynomials. Below is a **radial kernel**

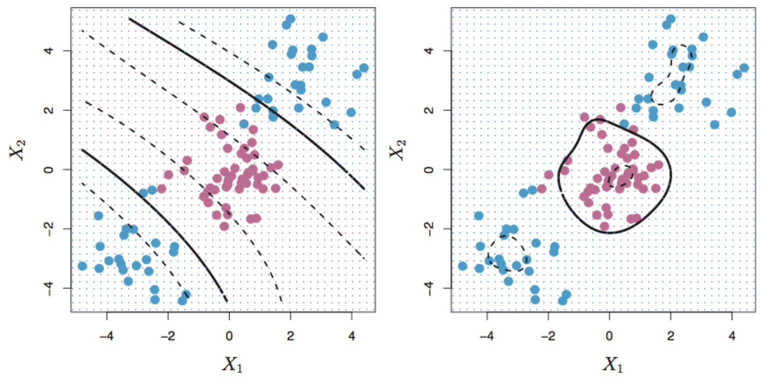

FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

Our Dataset

- Khan dataset: tissue samples corresponding to four types of cell tumours
- Each sample contains gene expression measurements.

Code

```
In [6]: library(ISLR)
        library(MASS)
        library(caret)
        names(Khan)
```

```
Loading required package: lattice
Loading required package: ggplot2
```

```
    'xtrain'   'xtest'   'ytrain'   'ytest'
```

Features

- We have gene expression measurements for 2308 genes. The training set has 63 observations, while the test set has 20 observations

- A dataset with many features like this is an example of a high-dimensional dataset.

```
In [10]: dim(Khan$xtrain)
```

```
    63   2308
```

```
In [11]: dim(Khan$xtest)
```

```
    20   2308
```

Are our datasets balanced?

```
In [12]: table(Khan$ytrain)
```

```
 1  2  3  4 
 8 23 12 20 
```

```
In [20]: table(Khan$ytest)
```

```
    3  2  4  1
```

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.

This time, we will use the e1071 package that is included in caret, to show how to implement analysis outside of caret. We will place the x variables and y variables together into one data frame, then apply the svm( ) function to the data.

```
In [42]: data_train <- data.frame(x=Khan$xtrain, y=as.factor(Khan$ytrain))
         head(data_train)
```

	x.1	x.2	x.3	x.4	x.5	x.6	x.7
V1	0.77334370	-2.438405	-0.4825622	-2.7211350	-1.2170580	0.82780920	1.342604
V2	-0.07817778	-2.415754	0.4127717	-2.8251460	-0.6262365	0.05448819	1.429498
V3	-0.08446916	-1.649739	-0.2413075	-2.8752860	-0.8894054	-0.02747398	1.159300
V4	0.96561400	-2.380547	0.6252965	-1.7412560	-0.8453664	0.94968680	1.093801
V5	0.07566390	-1.728785	0.8526265	0.2726953	-1.8413700	0.32793590	1.251219
V6	0.45881630	-2.875286	0.1358412	0.4053984	-2.0826470	0.13784710	1.733530

When the cost argument is small, then the mar- gins 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.

In [51]:
```
library(e1071)
out = svm(y ~ ., data=data_train, kernel="linear",cost=10)
summary(out)
```

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.0004332756

Number of Support Vectors:  58

 ( 20 20 11 7 )


Number of Classes:  4

Levels:
 1 2 3 4
```

In [52]:
```
# manually compute a confusion matrix
table(out$fitted , data_train$y)
```

```
     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.

In [59]:
```
data_test=data.frame(x=Khan$xtest , y=as.factor(Khan$ytest))
head(data_test)
```

	x.1	**x.2**	**x.3**	**x.4**	**x.5**	**x.6**	**x.7**
V1	0.1395010	-1.1689275	0.5649728	-3.366796	-1.323132	-0.692547360	2.327395
V2	1.1642752	-2.0181583	1.1035335	-2.165435	-1.440117	-0.437420279	2.661587
V4	0.8410929	0.2547197	-0.2087477	-2.148149	-1.512765	-1.263722809	2.946642
V6	0.6850646	-1.9275792	-0.2330676	-1.640413	-1.008954	0.774450632	1.617168
V7	-1.9561625	-2.2349264	0.2815634	-2.695628	-1.214697	-1.059872460	2.498070
V8	-0.2586412	-1.6847004	0.1758003	-2.323809	-1.692276	-0.008637193	2.302135

In [60]:
```
# run the predict function
pred_test=predict(out, newdata=data_test)
# output confusion matrix
table(pred_test, data_test$y)
```

```
pred_test 1 2 3 4
        1 3 0 0 0
        2 0 6 2 0
        3 0 0 4 0
        4 0 0 0 5
```

Only two errors!