Statistical Learning: Chapter 9
Support Vector Machines

Before we start, I'll remark that SVM are a little bit like the following strategy for two-class classification:

1. Expand your X's to a higher dimensional set
   - for example polynomials of X, including cross-products, or some other expansion

2. Fit a "linear decision bondary" classifier in this high dimensional space (e.g. LDA or logistic)

Remarks:
- Step 1 may involve an infinite dimensional expansion
- Step 2 involves a different criterion than LDA or logistic. It seeks to maximize class separation.
- Step 2 also involves efficient computational (kernels) for high- or infinite- dimensional spaces
- Both steps 1 & 2 will have complexity parameters that we can adjust (by CV)

The above is not 100% accurate, but it is the main idea.

The details will be developed in 3 stages:

9.1, the "Maximum Margin Classifier" focuses on the case where we can perfectly separate classes with a linear decision boundary.

9.2, "Support Vector Classifiers" considers a relaxing of the requirement of perfect class separation

9.3, "Support Vector Machines" considers the expansion of Xs to a high-dimensional set

9.1 Maximum Margin Classifier
9.1.1 What is a hyperplane?

In p=2 dimensions, it is defined by the equation:

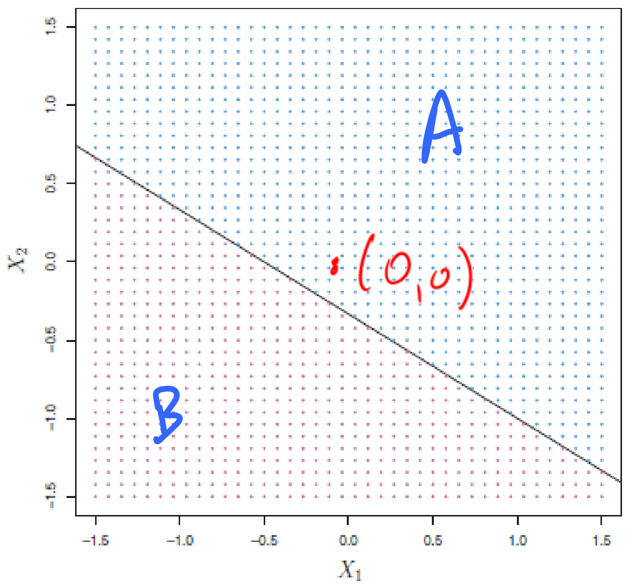
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

In the plot, the line represents the hyperplane

$$1 + 2X_1 + 3X_2 = 0$$

In area "A", 1 + 2X1 + 3X2 > 0 while in "B" it is < 0. By plugging X=(X1,X2) into the plane we can see which side of the hyperplane a point lies.

$$1 + 0 + 0 > 0 \Rightarrow A$$



Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

In p dimensions, the definition of a hyperplane generalizes to:

$$f(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

As in the 2-dimensional case, the hyperplane divides the p-space into 2 subspaces. We can tell what side an X is on by the sign of the above equation.

so sign $(f(x))$ = predicted class

9.1.2 Classification Using a Separating Hyperplane

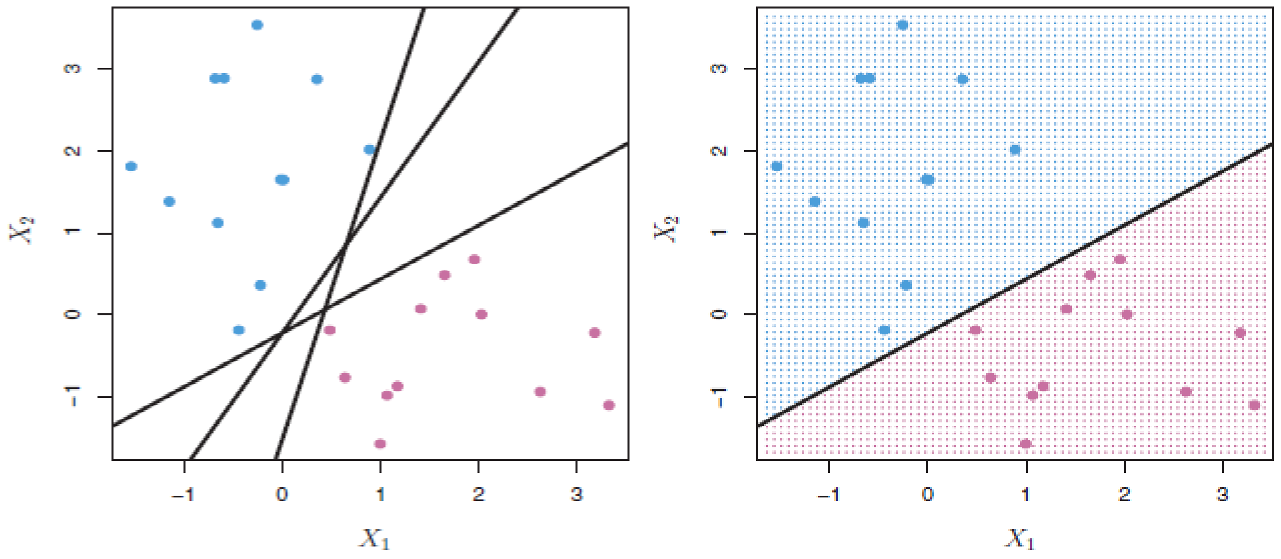
If we assume that we observe each data point as a p-vector X = (X1, X2, ..., Xp) and a response Y = -1 or +1 (class labels), we can use the idea of a separating hyperplane as a classifier.

If the classes can be separated by a hyperplane, there ~~may be~~ usually is more than one hyperplane.

Example below,

Left: Blue / Purple class labels, with 3 possible separating hyperplanes

Right: one of the hyperplanes from the left, with corresponding classifications



We saw above that the sign of

$$f(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

... tells us which side of the plane we're on. That is the predicted class label will simply be the sign of the equation.

Note that for the Y=1 class, the following is always true: (for separable classes)

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1,$$

Similarly for the Y= -1 class,

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1.$$

So a separating hyperplane has the property that for all i=1, ..., n

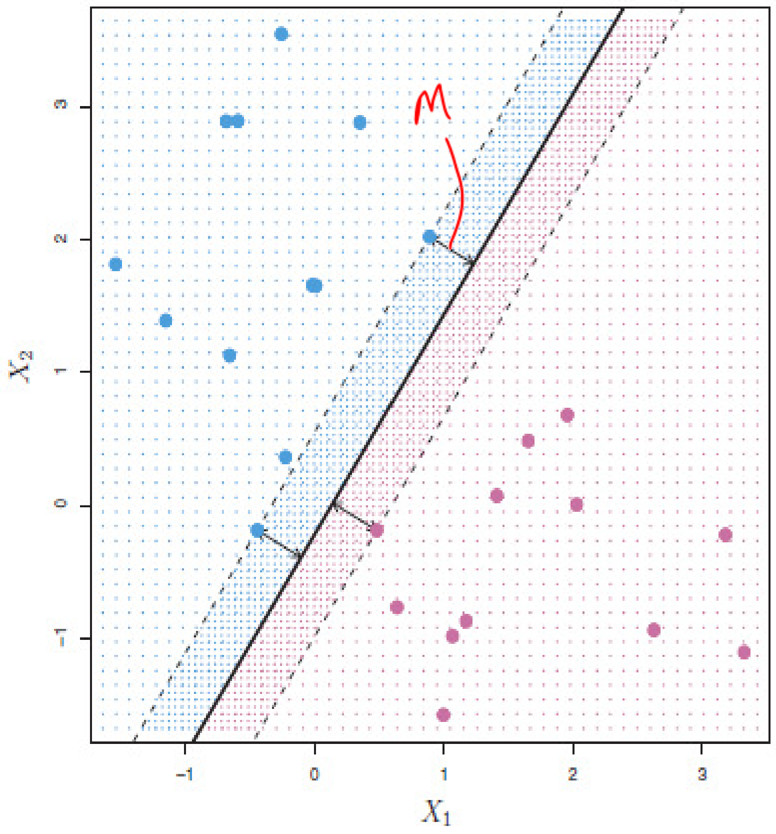
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0$$

One additional note: The distance from a point x* to the hyperplane is given by |f(x*)|, and equivalently by yf(x*), which is always positive for a separating hyperplane.

- If f(x*) is close to 0, we're near the plane and there's more class uncertainty.

9.1.3 The Maximal Margin Classifier
As noted above, if the classes can be separated by a hyperplane, there are (infinitely) many other hyperplanes that will also separate the classes.

We will choose the hyperplane that separates classes and maximizes the distance to the nearest points.



In the plot, the arrows indicate the distance from the plane to the nearest points, called the "margin" of the hyperplane.

Constrained optimization problem

1. $$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\; M$$

subject to 2. $$\sum_{j=1}^{p} \beta_j^2 = 1,$$

3. $$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$
for all $i = 1, \ldots, N.$

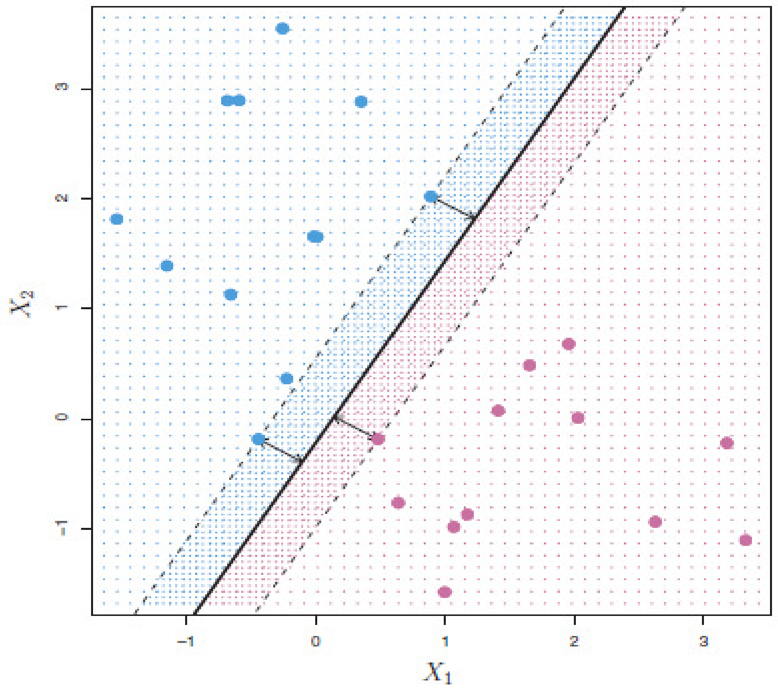
The "constrained optimization" looks harder than it is.

The 3rd equation ensures that each point is on the right side of the boundary ← ie outside shaded region
The 2nd condition (on squared betas) is for identifiability.
That is, if b0 + b1x1 + b2x2 = 0 defines a hyperplane then
a(b0 + b1x1 + b2x2) = 0 defines the same hyperplane, for any nonzero constant a.

So we're really trying to maximize the margin subject to simple constraints.

The actual optimization algorithm is outside the scope of the book / course. It can be cast as a convex quadratic programming problem.

But note that you are advised to "Be careful with large datasets as training times may increase rather fast" (http://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf)



One other note:

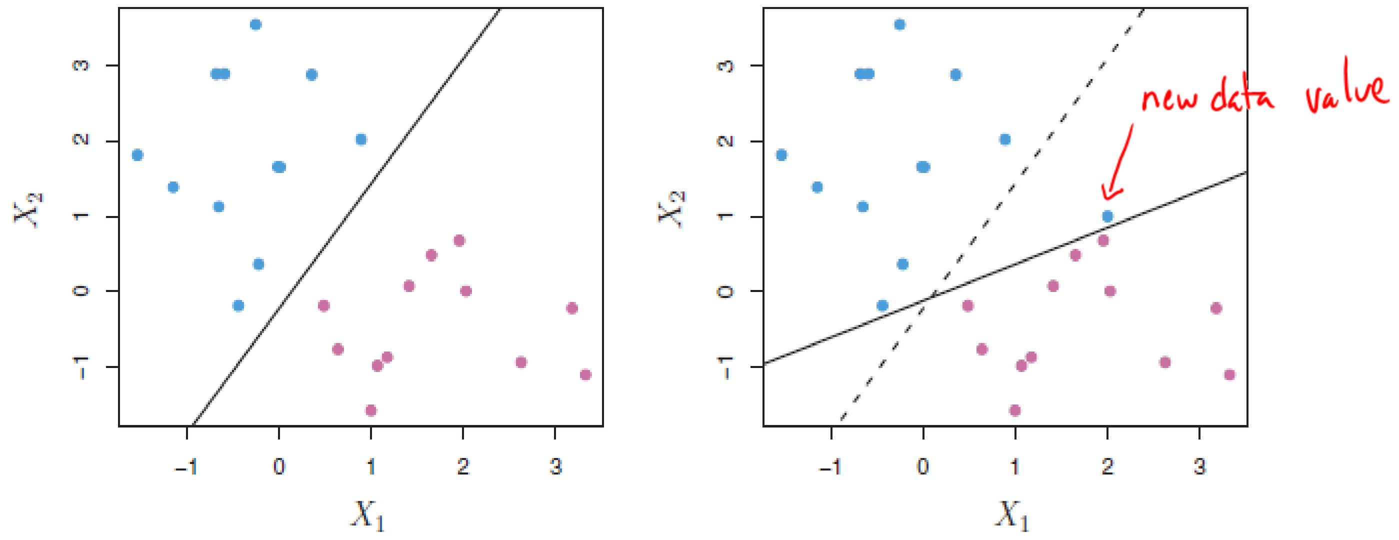
The 3 points (with arrows) in the plot on the left are called "**support vectors".**

They "support" the maximum margin hyperplane, in the sense that if we move any of these points a little bit, the maximum margin plane will move.

It's possible to move non-support vector points, without forcing the plane to move.

9.2 Support Vector Classifiers
Most classification problems are not separable, so we must modify the above procedure.

Furthermore, even if we had separability of classes, the maximum margin hyperplane can be sensitive to variation, as Figure 9.5 below demonstrates with the addition of a single point.



Left: maximum margin hyperplane
~~The~~

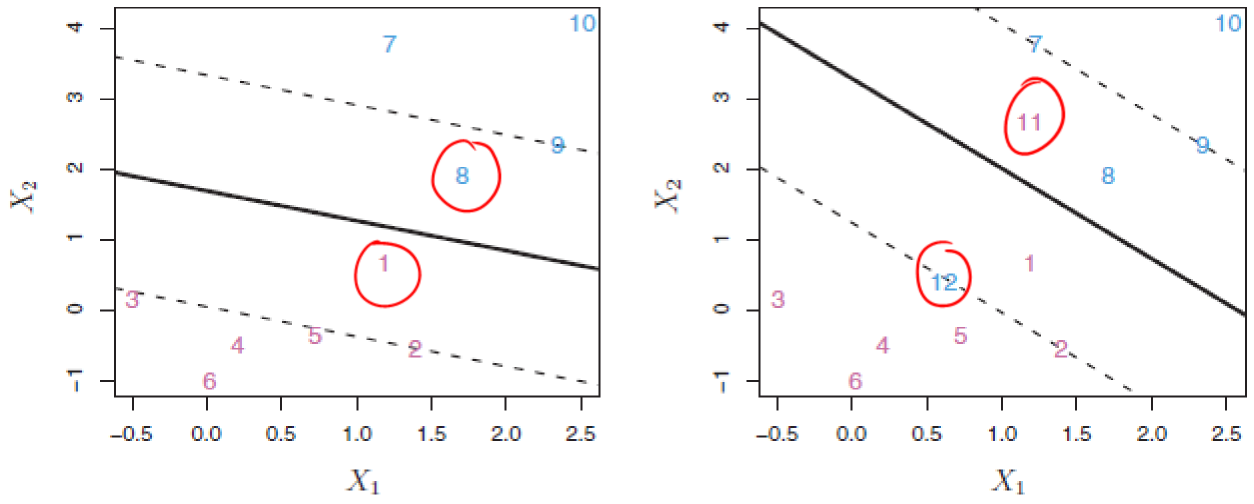
Right: same data with one new (blue) point.

new maximum margin hyperplane (solid line)

So we might consider a hyperplane that does not perfectly separate classes, in the interest of:

1. greater robustness to individual observations
2. Better classification of most of the training observations.

The **support vector classifier**, introduced next, accomplishes this. It will allow some points to be
- inside the margin, but on the right side of the hyperplane(1, 8)
- on the wrong side of the hyperplane (additional points 11, 12 in right plot).



These allowances can be represented by the following modified optimization problem:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \tag{9.12}$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.13}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{9.15}$$

M is still margin width, which we want to maximize

In (9.14) the epsilon_i's are "slack variables" allowing points to be on the wrong side of the margin, or of the hyperplane.
- If epsilon_i = 0, the observation i is on the right side of the margin
- If 0 < epsilon_i < 1, the observation is inside the margin, but on the right side of the hyperplane
- If epsilon_i > 1, the observation is on the wrong side of the hyperplane.

So the constant "C" in (9.15) is a bound on the number of training points that can be on the wrong side of the margin (i.e. misclassified).

The constant "C" is a tuning parameter that will affect the flexibility of the fitted model. We will choose it by cross-validation.

C=0 is a special case reducing to the maximum margin classifier, and does not permit any misclassification.

Small C leads to classifiers with few training set misclassifications, and greater sensitivity to variation.

Large C allows for many misclassifications, and less sensitivity.

In terms of the bias/variance tradeoff:
- small C gives a flexible model with low bias and high variance
- large C gives an inflexible model with high bias and low variance.

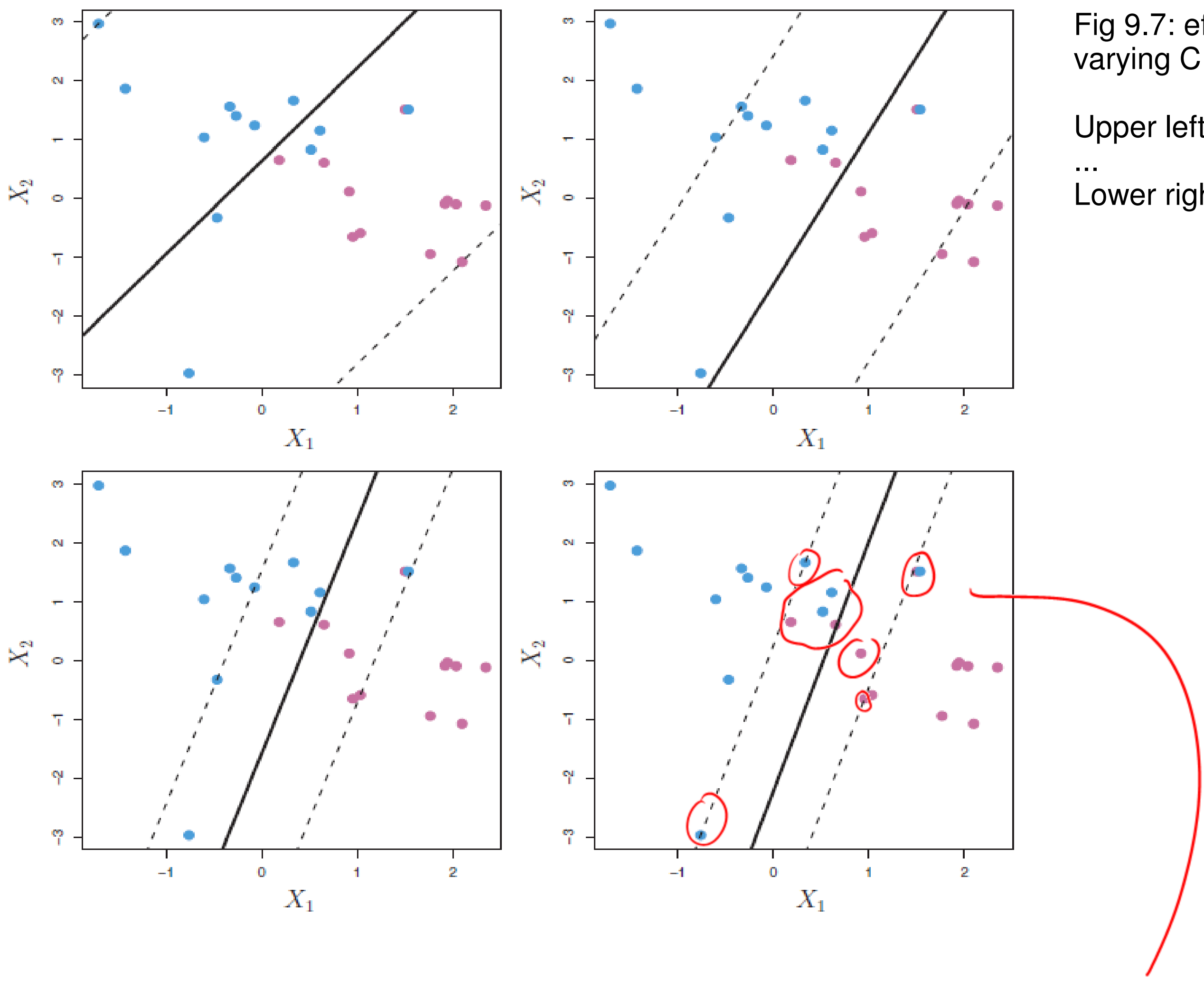


Fig 9.7: effect of varying C

Upper left: large C
...
Lower right: small C

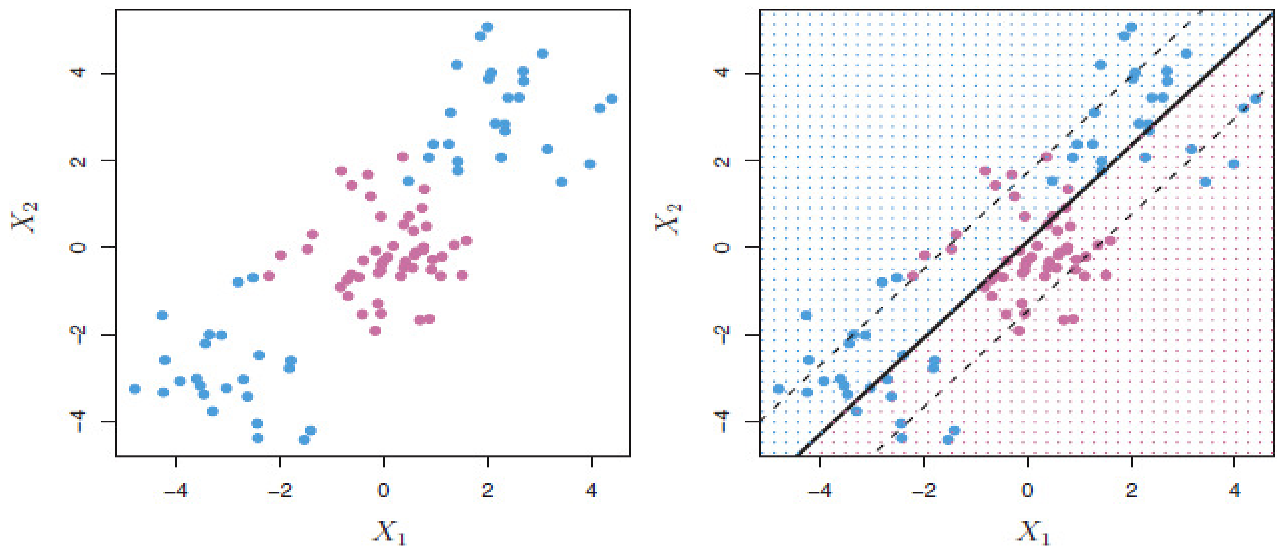
Note also that the "support points" turn out to now be only those points on the margin or on the wrong side of the margin.

9.3 Support Vector Machines

But we're not finished yet!

The linear boundary can fail altogether, even with an allowance parameter "C"

For example (Fig 9.8 below), we need a nonlinear boundary to separate the classes.



The solution is the "support vector machine" which enlarges the feature space.

We've seen this in Ch 3 and Ch 4 for linear regression - inclusion of quadratic, cubic, interaction terms, and so on. The same idea is a starting point here:

Enlarge the X space by including transformations such as

$$(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$$

Instead of just (X1,X2).

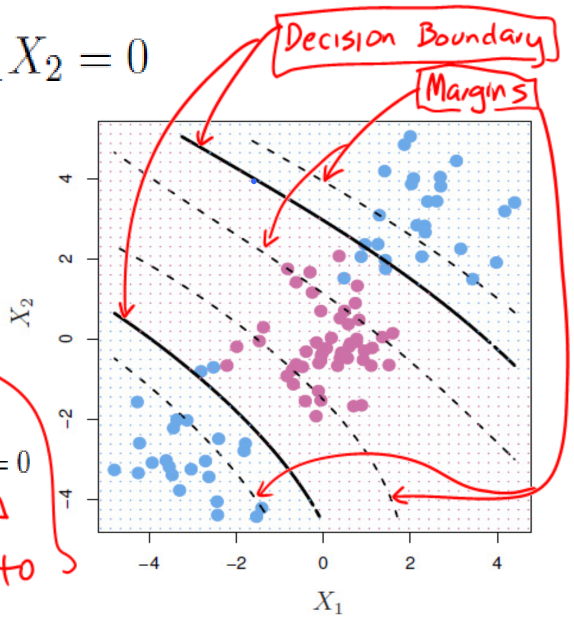
The the decision boundary is of the nonlinear form:

$$f(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

Example: all terms and cross products up to cubic order

So we have 9 terms instead of 2

$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\underbrace{\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2}_{\text{extra, relative to}} = 0$$



A better way to enlarge the predictor set: kernels

There's a more general and efficient way to introduce nonlinear terms in the model, using **kernels**.

But first we need to understand the role of inner products in the support-vector classifier.

We write the inner product between vectors as:

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

The linear support vector classifier can be represented as (Not obvious, can be proven)

$$\underset{\text{new prediction point}}{f(x)} = \beta_0 + \underbrace{\sum_{i=1}^{n}}_{\text{training points}} \alpha_i \langle x, x_i \rangle$$

To estimate the n "alpha" parameters, and beta_0, all we need are the (n choose 2) = n*(n-1)/2 inner products <x_i,x_j> over all (i,j) pairs of training observations.

It turns out that most of the alpha_i can be zero, and in fact the only nonzero ones are the ones that correspond to "support points".

So the classifier becomes

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

Where *S* is the support set of indicies i such that alpha_hat_i >0.

Generalizing from a linear boundary:

Suppose that everywhere we see the inner product <x_i,x_i'>, we replace it by a general function ~~k~~ K(x_i,x_i').

Choosing $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}, = \langle x_i, x_{i'} \rangle$

Will just give us the linear support vector classifier back.

But we can use other forms.

Note that this works because the support vector classifier only needs the inner products.

The kernel below, for example, computes all ((p+d) choose d) products and is called the polynomial kernel

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d. = (1 + \langle x_i, x_{i'} \rangle)^d$$

d = degree of polynomial

If you expand the above function for p=2, d=2, you get 6 terms corresponding to ↑ constant, linear, quadratic, and a 2-way interaction.

Using such a kernel function in a support vector classifier turns it into a "support vector machine"

Predictions can be produced, for example, we can calculate the value of the classification function at a new point "x", using the kernel and the support points, as:

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

In the polynomial kernel case we would consider choosing degree parameter "d" as a second "tuning constant", again via CV.

Other kernels exist. A popular one is the "radial kernel" which has the form

$$K(x_i, x_{i'}) = \exp(-\gamma \overbrace{\sum_{j=1}^{p} (x_{ij} - x_{i'j})^2}^{\text{Squared Euclidean distance between } x_i,\ x_{i'}}).$$

Again, there is a tuning parameter, in this case $\gamma > 0$, which once again can be chosen by CV.

Although it's difficult to see exactly how the radial kernel function works, note that the summation in the K function above is the squared Euclidean distance between point x_i and x_i'.

The book argues that since $\gamma > 0$, for a test observation x* and one training point x_i, if the Euclidean distance is large, then K(x*,x_i) will be very close to 0, and the point x_i will play little role in predicting x*.

The choice of γ controls the meaning of "close". A small γ will lead to more distant points being called "close", that is a less flexible classifier.

Two comments on computation:
- The use of a kernel function means that no matter how high-dimensional our transformed space is, we still only need to calculate the (n choose 2) = n*(n-1)/2 values of the kernel at all pairs of training points.

- $\binom{n}{2}$ does not scale well with "n"

$d=3$

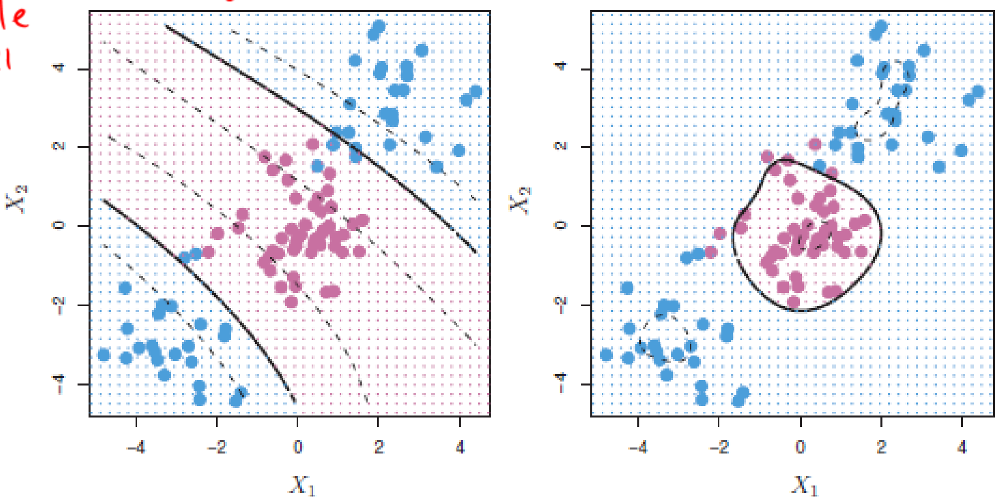


Figure 9.9 compares a SVM with the polynomial kernel (left) to one with the radial kernel (right)

Example: South African Heart Disease data (Heart data)

The text describes the fitting of various SVM models to the Heart data
- n = 297 observations (after removing NAs)
- binary outcome ("heart disease")
- 13 predictors, some categorical and some numeric

R will make indicator variables

We use the "svm" function in the R library "e1071" to fit the model.

```
> library(e1071)
> Heart = read.csv('http://www-bcf.usc.edu/~gareth/ISL/Heart.csv',
+                   head=TRUE,row.names=1)
> Heart = na.omit(Heart)
> set.seed(1)
> train = sample(nrow(Heart),207)
> svm1 = svm(AHD~.,data=Heart,kernel='linear',cost=100,subset=train)
```

I should have said "validate" not "test".

```
> svm1.test.class = predict(svm1,newdata=Heart[-train,])
> table(svm1.test.class,Heart$AHD[-train])
                   Actual
svm1.test.class No Yes
            No  41  10
            Yes  7  32
```

```
> svm2 = svm(AHD~.,data=Heart,kernel='radial',cost=10,gamma=.001,subset=train)
> svm2.test.class = predict(svm2,newdata=Heart[-train,])
> table(svm2.test.class,Heart$AHD[-train])

svm2.test.class No Yes
            No  44  11
            Yes  4  31
```

We can consider choosing the best value of the "cost" and "gamma" parameters using cross-validation.

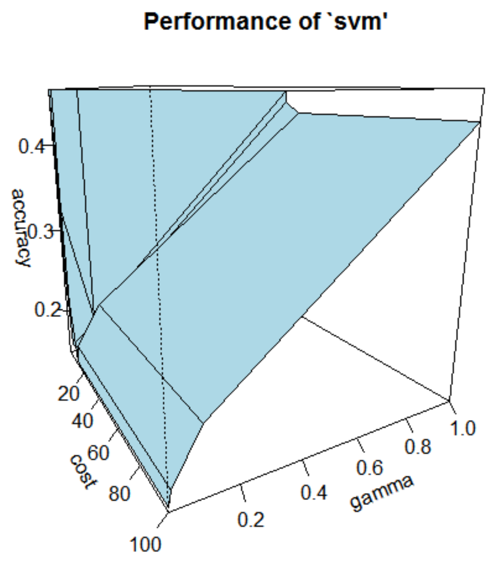
The generic "tune" function, also in e1071 library, will use k-fold CV (k=10 by default) to estimate a test error rate (MSE or misclassfication rate). It can do this for practically any model. Here we use it for svm:

```
> tune.out1 = tune(svm,AHD~.,data=Heart[train,],kernel='radial',
+ ranges=list(cost=c(.001,.01,.1,1,10,100),gamma=c(.001,.01,.1,1)))
> plot(tune.out1,type='persp')
> summary(tune.out1)
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10 0.001

- best performance: 0.145
```

$6 \times 4 = 24$ combinations of (cost, γ)

Misclass (CV)

```
- Detailed performance results:
    cost gamma     error dispersion
1  1e-03 0.001 0.4600000 0.09695412
2  1e-02 0.001 0.4600000 0.09695412
3  1e-01 0.001 0.4600000 0.09695412
4  1e+00 0.001 0.2419048 0.10234219
5  1e+01 0.001 0.1450000 0.05992405
6  1e+02 0.001 0.1492857 0.05133700
7  1e-03 0.010 0.4600000 0.09695412
8  1e-02 0.010 0.4600000 0.09695412
9  1e-01 0.010 0.3247619 0.12025944
10 1e+00 0.010 0.1547619 0.06402554
11 1e+01 0.010 0.1638095 0.04466781
12 1e+02 0.010 0.1633333 0.08949059
13 1e-03 0.100 0.4600000 0.09695412
14 1e-02 0.100 0.4600000 0.09695412
15 1e-01 0.100 0.1838095 0.08888747
16 1e+00 0.100 0.1783333 0.07928925
17 1e+01 0.100 0.2116667 0.10742438
18 1e+02 0.100 0.2116667 0.10505281
19 1e-03 1.000 0.4600000 0.09695412
20 1e-02 1.000 0.4600000 0.09695412
21 1e-01 1.000 0.4600000 0.09695412
22 1e+00 1.000 0.4457143 0.10924626
23 1e+01 1.000 0.4311905 0.11087532
24 1e+02 1.000 0.4311905 0.11087532
```



With tuning a SVM with CV, we're looking for the low point in a 2-dimensional "U" (in the directions of "cost" and "gamma").

Notice that I'm choosing costs and gamma values over a wide range.

The tune function will automatically evaluate all combinations of the two parameters given.

Here the values cost = 10, gamma = 0.001 give the best CV MSE, a value of 14.5%.
- Given that in tuning, we considered values of gamma and cost which ranged over orders of magnitude, we might want to consider additional exploration of the MSE surface.

```
> tune.out2 = tune(svm,AHD~.,data=Heart[train,],kernel='radial',
+ ranges=list(cost=c(1, 5, 10, 50),gamma=c(.0005,.001,.005,.01)))
> summary(tune.out2)
```

This suggests cost=5, gamma = .005

I think that some of the variation here is due to the different cross-validation runs, and that values of cost= 10 and gamma = .001 are probably fine.

See the lab sections of the textbook for additional R examples, including generation of the ROC curves.

SVMs: More than 2 classes?

SVMs work in 2-class problems. What if we have K>2 classes?

OVA: One Versus All: Fit K different 2-class SVMs, each predicting one class vs. the rest.
Classify an observation x* to the class for which the svm function f_k(x*) is largest.

OVO: One vs. One: Fit all (K choose 2) pairwise classifiers.
Classify x* to the class that wins the most pairwise competitions

OVO is recommended for small-to-moderate K. For large K it will become computationally expensive.