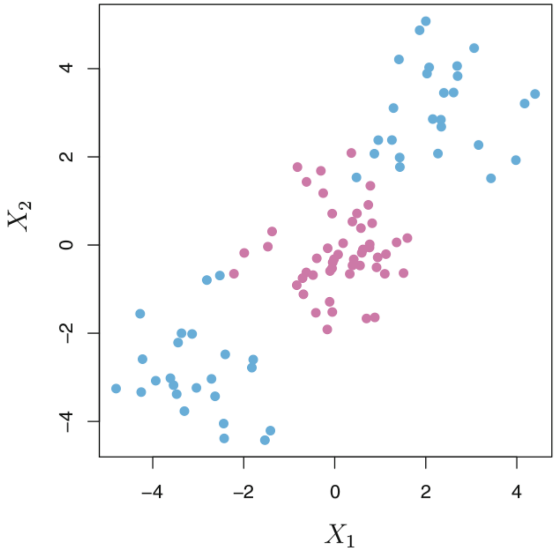
Support Vector Machines.

W. Wang[1]

[1] Department of Mathematics
University of Houston

MATH 4323

Data Example: What boundary is appropriate?



Classification with Non-linear Decision Boundaries.

The support vector classifier is a natural classification approach in the

- two-class setting,
- when the boundary between the two classes is linear.

In practice, we are sometimes faced with **non-linear** class boundaries.

Example. Next slide shows:

- (**Left**) a two-dimensional data example,
- (**Right**) separating hyperplane & support vectors result of applying a support vector classifier.

It is clear that a support vector classifier or any linear classifier will perform poorly here.

Question: How to deal with non-linear boundaries?

Classification with Non-linear Decision Boundaries.

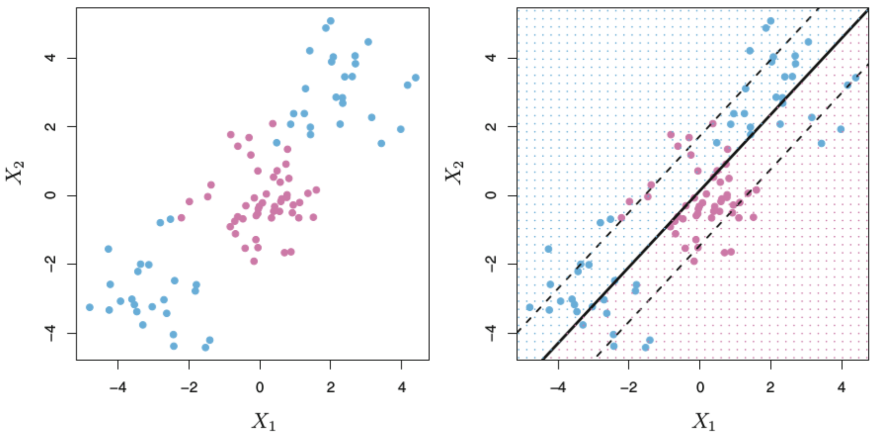


FIGURE 9.8. Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

Linear Regression: Enlarging Predictor Space.

Example. In linear regression with

- predictors X_1, X_2, and
- response Y

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

Question: How do we model potentially **non-linear** relationship between predictors X_1, X_2 and response Y?
Answer: **polynomial regression**, where we **enlarge** the predictor space to include quadratic, cubic & other terms. E.g. we could fit

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + \epsilon$$

extending the original predictor space as follows

$$\{X_1, X_2\} \Rightarrow \{X_1, X_1^2, X_2, X_2^2\}$$

Support Vector Classifier: Enlarging Predictor Space.

For support vector classifier, the problem of possible non-linear boundaries is addressed the **same way**:

We **enlarge** the feature space using

- quadratic,
- cubic or higher-order polynomial,
- general non-linear (e.g. $\log$ or *exp*)

functions of the predictors.

Example. In *p*-dimensional case,

- instead of fitting a support vector classifier using *p* features

$$\{X_1, X_2, \ldots, X_p\},$$

- we could fit a support vector classifier using 2*p* features

$$\{X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2\}$$

Support Vector Classifier: Enlarging Predictor Space.

Example (cont'd). Logic of obtaining separating hyperplane equation remains the same:

- while in the p-feature space of $\{X_1, X_2, \ldots, X_p\}$ we had

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{1}$$

- in the extended $2p$-feature space we have

$$\beta_0 + \beta_{11} X_1 + \beta_{12} X_1^2 + \beta_{21} X_2 + \beta_{22} X_2^2 + \cdots + \beta_{p1} X_p + \beta_{p2} X_p^2 = 0 \tag{2}$$

Why does (2) lead to a non-linear decision boundary?

- While in the **enlarged** feature space ($\{X_1, X_1^2, X_2, X_2^2, \ldots\}$), the resulting decision boundary is, in fact, linear,
- But in the **original** feature space, the decision boundary is of the form $q(x) = 0$, where q is a quadratic polynomial $\Longrightarrow$ its solutions are generally **non-linear**.

Enlarging Predictor Space: Example.

Example. Below is a similar case of non-linearly separable scenario:

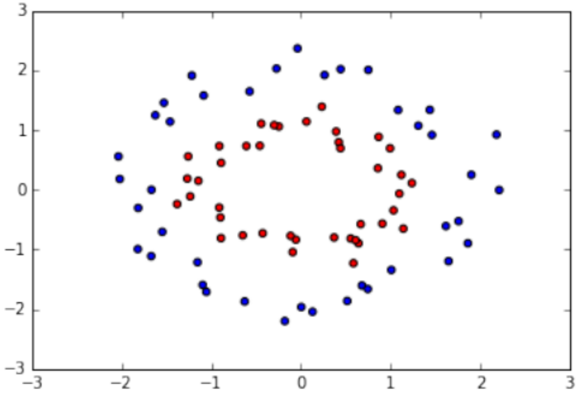


FIGURE 2.3: A not linearly separable dataset.

Enlarging Predictor Space: Example.

Example (cont'd). That's what happens when we enlarge the predictor space by adding a third predictor $Z = X_1^2 + X_2^2$:

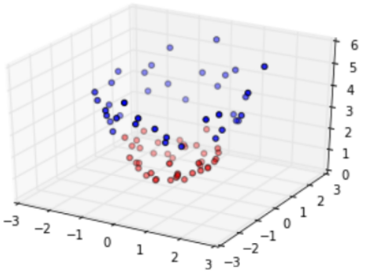


FIGURE 2.4: The dataset with the new z dimension.

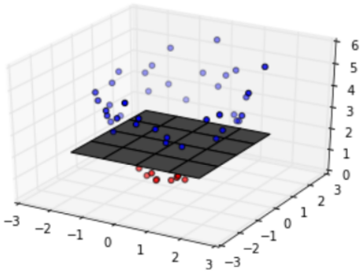


FIGURE 2.5: A hyperplane dividing the dataset.

There is a separating hyperplane

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 Z = 0$$

in the extended $3D$ predictor space.

Enlarging Predictor Space: Example.

Example (cont'd). In the meantime, if we were to project that hyperplane back onto the original $2D$ space of $\{X_1, X_2\}$:

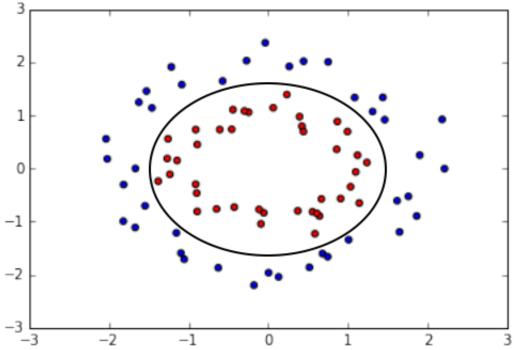


FIGURE 2.3: A not linearly separable dataset.

We get a circle shape as a result of equation that is not linear in X_1, X_2:

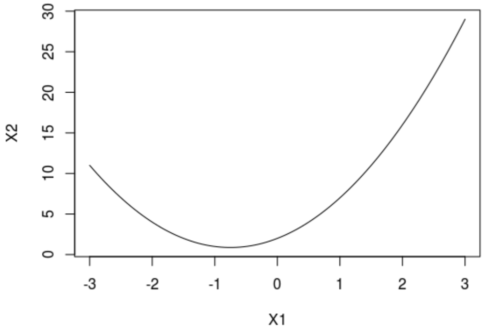
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1^2 + X_2^2) = 0$$

Support Vector Classifier: Enlarging Predictor Space.

Example. Let's look at $p = 2$, with extended space $\{X_1, X_1^2, X_2, X_2^2\}$ and the following fitted hyperplane equation:

$$2 - X_1 + 0 \times X_1^2 + 3X_2 + 2X_2^2 = 0$$

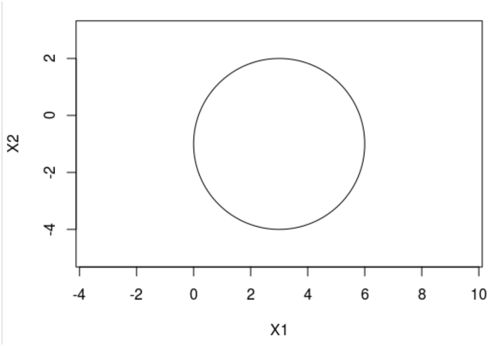
This corresponds to the following separating $2D$ hyperplane:



Support Vector Classifier: Enlarging Predictor Space.

Example. Let's look at $p = 2$, with extended space $\{X_1, X_1^2, X_2, X_2^2\}$ and the following fitted hyperplane equation:

$$1 - 6 \times X_1 + X_1^2 + 2 \times X_2 + X_2^2 = 0$$



Looks familiar, and after some rearrangement:

$$(X_1 - 3)^2 + (X_2 + 1)^2 = 9$$

Support Vector Classifier: Enlarging Predictor Space.

When enlarging original space of p features with quadratic terms, which leads to separating hyperplane equation of the form

$$\beta_0 + \beta_{11}X_1 + \beta_{12}X_1^2 + \beta_{21}X_2 + \beta_{22}X_2^2 + \cdots + \beta_{p1}X_p + \beta_{p2}X_p^2 = 0$$

the initial optimization task (6) – (9) (from "Maximal Margin & Support Vector Classifier" sides) becomes

$$\underset{\beta_0,\beta_{11},\beta_{12},\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \tag{3}$$

$$\text{subject to } \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2 = 1 \tag{4}$$

$$y_i(\beta_0 + \sum_{j=1}^{p}\beta_{j1}x_{ij} + \sum_{j=1}^{p}\beta_{j2}x_{ij}^2) \geq M(1-\epsilon_i), \tag{5}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n}\epsilon_i \leq C, \tag{6}$$

Support Vector Classifier: Enlarging Predictor Space.

One could also enlarge the feature space with

- higher-order polynomial terms ($X_1^3, X_1^4, \ldots$)

  Example. For $2D$ case of (X_1, X_2), one could extend to

  $$\{X_1, X_1^2, X_1^3, X_1^4, X_2, X_2^2, X_2^3, X_2^4\}$$

- interaction terms ($X_i X_j, i \neq j$),

  Example. For $3D$ case of (X_1, X_2, X_3), one could extend to

  $$\{X_1, X_1^2, X_2, X_2^2, X_3, X_3^2, X_1X_2, X_1X_3, X_2X_3\}$$

- non-polynomial functions of predictors ($\log(X_1), exp(X_1), \ldots$)

  Example. For $2D$ case of (X_1, X_2), one could extend to

  $$\{X_1, \log(X_1), X_2, \log(X_2)\}$$

Support Vector Machines.

There are infinitely many ways to enlarge the feature space, and one doesn't want to end up with a huge number of features, as it renders computations unmanageable.

The support vector **machine**, which we present next, allows us a

- general framework of enlarging the feature space for a support vector classifier,
- in a way that leads to efficient computations.

Support Vector Machines: Optimization Task.

REMINDER: For a given set of features $\{X_1, \ldots, X_p\}$, support vector classifier is obtained by finding the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ of hyperplane equation

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0$$

that solve the following optimization task

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \tag{7}$$

$$\text{subject to}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{8}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9}$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C \tag{10}$$

Support Vector Machines: Inner Products.

It turns out that the solution to the support vector classifier optimization task (7) – (10) involves only the **inner products** of the observations $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

The inner product of two r-dimensional vectors $\mathbf{a} = (a_1, \ldots, a_r) \in \mathbb{R}^r$ and $\mathbf{b} = (b_1, \ldots, b_r) \in \mathbb{R}^r$ is defined as

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{k=1}^{r} a_k b_k = \mathbf{a}^T \mathbf{b}$$

Thus the inner product of two observations

- $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip}) \in \mathbb{R}^p$, and
- $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jp}) \in \mathbb{R}^p$

is given by

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{k=1}^{p} x_{ik} x_{jk}$$

Support Vector Machines: Inner Products.

It can be shown that (derivation is beyond the scope of class)

1. The solution to support vector classifier optimization task (7) – (10) can be represented as

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \equiv \beta_0 + \sum_{k=1}^{p} \beta_k x_k =$$

$$= \beta_0 + \sum_{i=1}^{n} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle \tag{11}$$

   where we have

   - $\mathbf{x} = (x_1, \ldots, x_p)^\intercal, \mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^\intercal, i = 1, \ldots, n,$
   - n parameters $\alpha_i, i = 1, \ldots, n$, one per training observation $\mathbf{x}_i$.

   Note: keep in mind, $\{X_1, \ldots, X_p\}$ is the original, not transformed, feature space.

Support Vector Machines: Inner Products.

2. To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0 in eq. (11), all we need are the $\binom{n}{2}$ inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ between all pairs of training observations, meaning

$$\hat{\beta}_0 = g_0(\langle \mathbf{x}_1, \mathbf{x}_2 \rangle, \langle \mathbf{x}_1, \mathbf{x}_3 \rangle, \ldots, \langle \mathbf{x}_{n-1}, \mathbf{x}_n \rangle)$$

$$\hat{\alpha}_i = g_i(\langle \mathbf{x}_1, \mathbf{x}_2 \rangle, \langle \mathbf{x}_1, \mathbf{x}_3 \rangle, \ldots, \langle \mathbf{x}_{n-1}, \mathbf{x}_n \rangle), \ i = 1, \ldots, n$$

where $g_0, g_1, \ldots, g_n$ are functions resulting from solving the optimization task (7) – (10).

In our scenario,

$$\binom{n}{2} \equiv \frac{n(n-1)}{2} \equiv$$

$$\equiv \{\text{the number of pairs among a set of } n \text{ training observations}\}$$

Support Vector Machines: Inner Products.

3. However, it turns out that
   - $\alpha_i \neq 0 \Leftrightarrow$ training observation $\mathbf{x}_i$ is a support vector,
   - $\alpha_i = 0 \Leftrightarrow$ training observation $\mathbf{x}_i$ is not a support vector,

   So if S is the collection of indices for training observations that are support vectors, then

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = \beta_0 + \sum_{i \in S} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle \tag{12}$$

   So, in the end of the day, $\hat{\beta}_0$ and $\hat{\alpha}_i$ solving the (7) – (10) optimization task via the representation in eq. (11) are:

$$\begin{cases} \hat{\beta}_0 = g_0(\langle \mathbf{x}_1, \mathbf{x}_2 \rangle, \langle \mathbf{x}_1, \mathbf{x}_3 \rangle, \ldots, \langle \mathbf{x}_{n-1}, \mathbf{x}_n \rangle) \\ \hat{\alpha}_i = g_i(\langle \mathbf{x}_1, \mathbf{x}_2 \rangle, \langle \mathbf{x}_1, \mathbf{x}_3 \rangle, \ldots, \langle \mathbf{x}_{n-1}, \mathbf{x}_n \rangle), \; i \in S \\ \hat{\alpha}_i = 0, \; i \notin S \end{cases}$$

Support Vector Machines: Kernels.

Now, let's replace any appearance of inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ in (12) with a **generalization** of the inner product

$$K(\mathbf{x}_i, \mathbf{x}_j),$$

where K is a function referred to as **kernel**.

That leads to the following solution form:

$$f(\mathbf{x}) = \beta_0 + \sum_{i \in S} \alpha_i K(\mathbf{x}, \mathbf{x}_i) \tag{13}$$

with $\hat{\beta}_0$ and $\hat{\alpha}_i$ being functions of $K(\mathbf{x}_i, \mathbf{x}_j)$, instead of just $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$:

$$\begin{cases} \hat{\beta}_0 = g_0(K(\mathbf{x}_1, \mathbf{x}_2), K(\mathbf{x}_1, \mathbf{x}_3)), \ldots, K(\mathbf{x}_{n-1}, \mathbf{x}_n)) \\ \hat{\alpha}_i = g_i(K(\mathbf{x}_1, \mathbf{x}_2), K(\mathbf{x}_1, \mathbf{x}_3), \ldots, K(\mathbf{x}_{n-1}, \mathbf{x}_n)), \ i \in S \\ \hat{\alpha}_i = 0, \ i \notin S \end{cases}$$

Support Vector Machines: Linear Kernel.

A kernel is a function that quantifies the **similarity** of two observations.

Example. For two observations $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{p} x_{ik} x_{jk} \text{ - linear kernel.}$$

A few questions popping up on that:

1. Why is it called **linear**?

   If we plug it into (13):

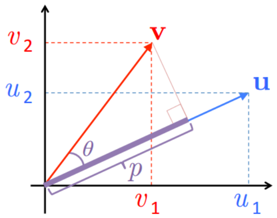
$$f(\mathbf{x}) = \beta_0 + \sum_{i \in S} \alpha_i K(\mathbf{x}, \mathbf{x}_i) = \beta_0 + \sum_{i \in S} \alpha_i (\sum_{k=1}^{p} x_k x_{ik}) =$$

$$= \beta_0 + \sum_{k=1}^{p} (\sum_{i \in S} \alpha_i x_{ik}) x_k \implies \textbf{linear} \text{ function in features } x_1, \ldots, x_p$$

Support Vector Machines: Linear Kernel.

2. Why does it quantify similarity of $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{x}_j \in \mathbb{R}^p$?

Vector Inner Product



$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\begin{aligned} \|\mathbf{u}\|_2 &= \text{length}(\mathbf{u}) \in \mathbb{R} \\ &= \sqrt{u_1^2 + u_2^2} \end{aligned}$$

$$\begin{aligned} \mathbf{u}^\intercal \mathbf{v} &= \mathbf{v}^\intercal \mathbf{u} \\ &= u_1 v_1 + u_2 v_2 \\ &= \|\mathbf{u}\|_2 \; \|\mathbf{v}\|_2 \cos\theta \end{aligned}$$

Quantifies **directional** similarity well - whether vectors point in the

- same direction ($cos(\theta) > 0$)
- opposite directions ($cos(\theta) < 0$)

Support Vector Machines: Polynomial Kernel.

To introduce **non-linearity** into the boundaries produced by solving optimization task (7) – (10), we can use the following kernels:

- $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^d = (1 + \sum_{k=1}^{p} x_{ik}x_{jk})^d$ - polynomial kernel of degree d (d - positive integer). It is a non-linear kernel.

  Question: What happened to the idea of manually adding extra features, e.g. $X_1^2, X_2^2, \ldots$?

  Answer: Polynomial kernel implicitly adds those for us, depending on value d specified. E.g. for $d = 2$:

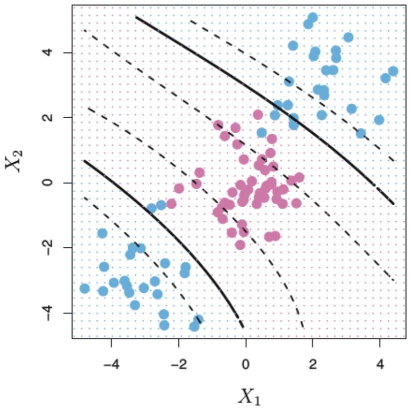
$$f(\mathbf{x}) = \beta_0 + \sum_{i \in S} \alpha_i K(\mathbf{x}, \mathbf{x}_i) = \beta_0 + \sum_{i \in S} \alpha_i (1 + \sum_{k=1}^{p} x_k x_{ik})^2 =$$

$$\beta_0 + \sum_{i \in S} \alpha_i (1 + 2\sum_{k=1}^{p} x_{ik} x_k + (\sum_{k=1}^{p} x_{ik} x_k)^2)$$

  $\Rightarrow$ **Task:** show explicitly that it's a linear function of $x_1, x_1^2, x_2, x_2^2, \ldots$

Support Vector Machines: Polynomial Kernel.

Example (cont'd). Back to our example from the very first slide: here's an SVM fit with polynomial kernel of degree 3.



It produces much more flexible boundary shapes, which appear appropriate for this data.

Support Vector Machines: Radial Kernel.

Another example of a kernel yielding flexible non-linear boundary is:

- $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \sum_{k=1}^{p}(x_{ik} - x_{jk})^2), \gamma > 0$ - radial kernel.

Intuition:

If obs. $\mathbf{x}^* = (x_1^*, \ldots, x_p^*)^T$ is far from obs. $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^T \implies$
Euclidean distance $\sum_{k=1}^{p}(x_k^* - x_{ik})^2$ is large $\implies$
$K(\mathbf{x}^*, \mathbf{x}_i) = exp(-\gamma \sum_{k=1}^{p}(x_{ik}^* - x_{ik})^2)$ is very tiny.

This goes on to show that

- Radial kernel is a similarity measure between $\mathbf{x}^*$ & $\mathbf{x}_i$.
- A "far away" support vector $\mathbf{x}_i$ plays virtually no role in $f(\mathbf{x}^*)$

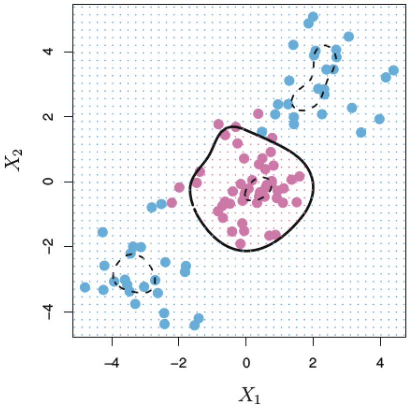
$$f(\mathbf{x}^*) = \beta_0 + \sum_{i \in S} \alpha_i K(\mathbf{x}^*, \mathbf{x}_i)$$

due to tiny value of $K(\mathbf{x}^*, \mathbf{x}_i)$.

Support Vector Machines: Radial Kernel.

Hence, the radial kernel has very local behavior: only **nearby** training observations determine the class label of a test observation.

Example (cont'd). Below is the SVM fit for radial kernel with $\gamma = 0.5$.



Support Vector Machines: Radial Kernel.

- γ is a parameter of the radial based kernel. It determines the spread of the kernel and therefore the decision region.
- When γ is small, the curvature of the decision boundary is very low and thus the decision region is broad.
- When γ is large, the curvature of the decision boundary is high, which may lead to islands of decision-boundaries around data points.

Kernel vs Enlarging Feature Space: Advantages?

Biggest advantage of kernels as opposed to simply enlarging feature space is **computational**:

- With kernels, we only compute $K(\mathbf{x}_i, \mathbf{x}_j)$ for all $\binom{n}{2}$ distinct pairs $\mathbf{x}_i, \mathbf{x}_j$ of training observations, in the original p-dimensional space.
- Kernels are computed without explicitly working in the enlarged feature space, which in many SVM applications can be so large that computations are **intractable**.

  Example. Imagine working in 10-dimensional space:

  $$\{X_1, X_2, \ldots, X_{10}\}$$

  and extending it to include quadratic, cubic terms + interactions:

  $$\{X_1, X_1^2, X_1^3, \ldots, X_1X_2, X_1X_3, \ldots, X_1X_2X_3\}$$

  You end up with $10 \times 3 + \frac{10(10-1)}{2} + 1 = 76$ features.

SVM: *Heart* data example.

Example. In *Heart* data we have

- 13 predictors (such as age, sex, cholesterol level, etc), and
- predict whether individual has heart disease or not ($AHD = Yes/No$, binary response)

```
> Heart <- read.csv("~/Downloads/Heart.csv")
> head(Heart)
  X Age Sex    ChestPain RestBP Chol ...  AHD
1 1  63   1      typical    145  233 ...   No
2 2  67   1 asymptomatic    160  286 ...  Yes
3 3  67   1 asymptomatic    120  229 ...  Yes
4 4  37   1   nonanginal    130  250 ...   No
5 5  41   0   nontypical    130  204 ...   No
6 6  56   1   nontypical    120  236 ...   No
```

We will use support vector machines (SVM) to build a classifier diagnosing the heart disease status of a patient given other characteristics.

Heart data example: EDA.

Example (cont'd). But first, some exploratory data analysis:

```
> dim(Heart)
 303  15
> summary(Heart)
      X               Age         ..               Ca            Thal
 Min.   :  1.0   Min.   :29.00   ..   Min.   :0.0000   fixed : 18
 1st Qu.: 76.5   1st Qu.:48.00   ..   1st Qu.:0.0000   normal:166
 Median :152.0   Median :56.00   ..   Median :0.0000   revers:117
 Mean   :152.0   Mean   :54.44   ..   Mean   :0.6722   NA's  :  2
 3rd Qu.:227.5   3rd Qu.:61.00   ..   3rd Qu.:1.0000
 Max.   :303.0   Max.   :77.00   ..   Max.   :3.0000
                                 ..   NA's   :4
```

We may witness a few **missing** values among the *Ca* and *Thal* variables. Typically one may want to look into it in more detail, but given that it is just 6 observations out of 303, we may simply discard them:

```
Heart <- na.omit(Heart)
dim(Heart)
 297  15
summary(Heart)  # Check - does it have NAs now?
```

SVM in *R*: *svm*() function of *e*1071 library.

SVM in *R* can be performed via *svm*() function of *e*1071 library:

```
library(e1071)
?svm()
```

Example (cont'd). To build a support vector machine with linear kernel ($\Leftrightarrow$ support vector classifier) for *Heart* data:

```
svm.obj <- svm(AHD ~.,
               data=Heart,
               kernel='linear',
               cost=5)
```

In this *svm*() call:

- $AHD \sim .$ - classical *formula* argument,
- $data = Heart$ - classical *data* argument,
- $kernel =$ 'linear' - SVM kernel (other opt: 'polynomial', 'radial', ...)
- $cost = 5$ - the cost of a margin violation ($\equiv \frac{1}{Budget} = \frac{1}{C}$,).
- by **default** has $scale = TRUE$ - it scales the features

SVM in *R*: *predict*()

Example (cont'd). To evaluate quality of SVM classifications, we first obtain them via *predict*() function:

```
> predict(svm.obj)
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15 ...
 No Yes Yes  No  No  No Yes  No Yes Yes  No  No  No  No  No ...

> head(data.frame(True=Heart$AHD, Predicted=predict(svm.obj)))
  True Predicted
1   No        No
2  Yes       Yes
3  Yes       Yes
4   No        No
5   No        No
6   No        No

> mean(predict(svm.obj) != Heart$AHD)
[1] 0.1279461
```

This SVM with linear kernel yields 12.8% training error.

SVM in *R*: Radial Kernel.

Example (cont'd). Now, let's run SVM for radial basis kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \sum_{k=1}^{p} (x_{ik} - x_{jk})^2), \gamma > 0$$

with $\gamma = 10^{-3}, 10^{-2}$ and 10^{-1}. Value of γ is specified via *gamma* argument to *svm*() function:

```
for (j in 3:1){
  svm.obj <- svm(AHD ~.,
                 data=Heart,
                 kernel='radial',
                 gamma=10^{-j},
                 cost=5)

  print(mean(predict(svm.obj) != Heart$AHD))}
[1] 0.1481481
[1] 0.1178451
[1] 0.006734007
```

SVM in *R*: Radial Kernel.

Example (cont'd). As we could see above,

training error is decreasing (from 0.148 down to 0.007)
as
γ is increasing (from 10^{-3} up to 10^{-1})

BUT it might a sign of overfitting $\Longrightarrow$ we need a test error estimate.

Let's randomly divide data $70/30$ into

- 70% training data (207 observations), and
- 30% test set (90 observations).

Then let's compare the performance on test set for

- SVM with linear kernel,
- SVMs with radial kernal and γ values of 10^{-3}, 10^{-2}, 10^{-1}.

NOTE: these test error estimates are based just on one train/test data subdivision. **Cross-validation** should be done for reliable model comparisons (see *tune*() function used in Labs #4 and #5).

SVM in *R*: Test errors.

```
set.seed(1)
n <- nrow(Heart);   train <- sample(1:n, 0.7*n)

## Support Vector Classifier (Linear Kernel)
svm.obj <- svm(AHD ~., data=Heart, cost=5, kernel='linear',
               subset=train)
mean(predict(svm.obj, newdata=Heart[-train,])
     != Heart$AHD[-train])
[1] 0.2

## Support Vector Machines (Radial Kernel)
for (j in 3:1){
  svm.obj <- svm(AHD ~., data=Heart, cost=5,
                 kernel='radial', gamma=10^{-j},
                 subset=train)
  print(mean(predict(svm.obj, newdata=Heart[-train,])
        != Heart$AHD[-train]))}
[1] 0.1666667
[1] 0.2
[1] 0.1555556
```

False Postives/Negatives.

While misclassification error is an important metric of classification model performance, it doesn't give us a full picture of perfomance quality.

Example. In *Default* data set, describing whether a customer defaults on credit card payment or not, we had only $\approx$ 3% of customers that defaulted. Hence, if we simply predicted *Default* $=$ *No* for **each customer**, we'd obtain a train/test error of $\leq$ 3-4% just like that.

In general, $1/0$ (or *TRUE*/*FALSE*) binary classifiers can make **two types** of misclassification errors:

1. Misclassify a 1 for a 0 (*TRUE* for *FALSE*) $\implies$ false **negative**.
2. Misclassify a 0 for a 1 (*FALSE* for *TRUE*) $\implies$ false **positive**.

It is often of interest to determine exactly which of these two types of errors are being made.

False Postives/Negatives, Confusion Matrix.

Example (cont'd). For the *Default* example, our algorithm can

1. incorrectly assign an individual who actually defaults ($\textit{Default} = \textit{Yes}$) to the $\textit{Default} = \textit{No}$ category $\Longrightarrow$ false **negative**,
2. incorrectly assign an individual who does not default ($\textit{Default} = \textit{No}$) to the $\textit{Default} = \textit{Yes}$ category $\Longrightarrow$ false **positive**.

A **confusion matrix** is a convenient way to display that information:

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

False positive rate (FPR): for customers that didn't default, model predicts them incorrectly ($\textit{Default} = \textit{Yes}$) in just $\frac{23}{9667} = 0.02\%$ cases $\Longrightarrow$ $\textit{FPR} = 0.02$.

Question: What's the **false negative rate (FNR)** though?

False Postives/Negatives, Confusion Matrix.

Example. Back to the *Heart* data of predicting whether a patient has a heart disease or not. SVM with radial kernel and $\gamma = 10^{-1}$ yielded best test set performance (15.5% error). Let's check the confusion matrix of its predictions:

```
> table(pred=predict(svm.obj, newdata=Heart[-train,]),
        true=Heart$AHD[-train])
     true
pred  No Yes
  No  42   8
  Yes  6  34
```

Question #1: What is the false positive rate? False negative rate?

Question #2: Given that we are trying to detect a disease, which metric should be of more importance to us - false positive or false negative rate?

SVMs with More than Two Classes: One-vs-One.

So far, we've discussed SVMs for binary classification, the **two-class** setting. How to extend SVMs onto a general case of $K \geq 2$ classes?

One-versus-One Classification:

1. Construct $\binom{K}{2}$ $(\equiv \frac{K(K-1)}{2})$ SVMs, for all possible pairs of classes.
2. Use those SVMs to get $\binom{K}{2}$ class predictions for a test observation.
3. Assign this test observation the most frequent class among the $\binom{K}{2}$ predictions.

SVMs with More than Two Classes: One-vs-All.

One-versus-All Classification:

1. For class k, $k = 1, \dots, K$, fit SVM such that it compares
   - observations from class k (class #1, $y = +1$), with
   - those not belonging to class k (class #2, $y = -1$)

   Let $\beta_{0k}, \beta_{1k}, \dots, \beta_{pk}$ denote hyperplane parameters from such "One-vs-All" SVM for class $k, k = 1, \dots, K$. Given that $y_i = +1$ for arbitrary observation $\mathbf{x} \in$ class k:

$$\mathbf{x} \in \{\text{Class } k\} \Leftrightarrow \beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \cdots + \beta_{pk}x_p^* > 0$$

2. Given test observation $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_p^*)$, we assign it to class k such that

$$k = \max_{k, k=1,\dots,K} \beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \cdots + \beta_{pk}x_p^*$$

   as this amounts to a high level of confidence that the test observation belongs to the k^{th} **class** rather than to any of the other classes.

Which one to choose?

- **One-versus-One.** Fit all $\binom{K}{2}$ pairwise SVM classifiers. Classify test observation x^* to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.
- **One-versus-All.** Fit K different 2-class SVM classifiers $\hat{f}_k(x), k = 1, ..., K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x)$ is largest.
- Which one to choose? If K is not too large, use One-versus-One.

One-vs-One SVM example in *R*.

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.

Example. We create a simulated data example with $K = 3$ classes of observations in $2D$ predictor space. We first generate some data on $K = 2$ classes with a non-linear class boundary, as follows:

```
set.seed(1)
x1 <- rnorm(200)
x2 <- rnorm(200)

x1[1:100] <- x1[1:100] + 2; x2[1:100] <- x2[1:100] + 2;
x1[101:150] <- x1[101:150] - 2; x2[101:150] <- x2[101:150] - 2

y=c(rep(1,150),
    rep(2,50))

x <- cbind(x1,x2)
```

One-vs-One SVM example in *R*.

Example (cont'd). Then we add a *3rd* 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))
```

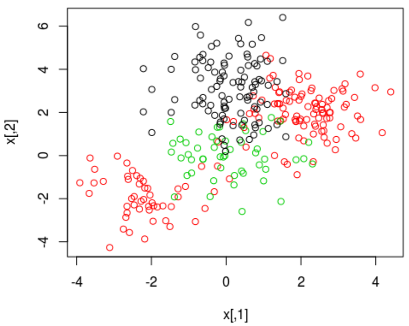
We now fit an SVM to the data:

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

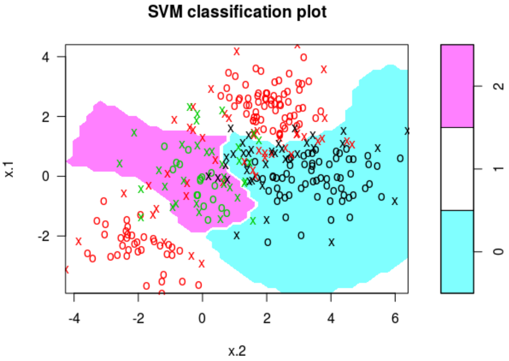
One-vs-One SVM example in *R*.

Example (cont'd). Below are the plots of original data (left) and the SVM fit (right).

```
par(mfrow=c(1,2))
plot(x,col=(y+1))
plot(svmfit, dat)
```



x[,2]
x[,1]
-4
-2
0
2
4
6


SVM classification plot
x.1
x.2