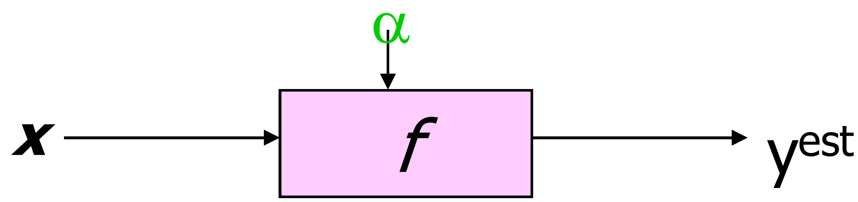
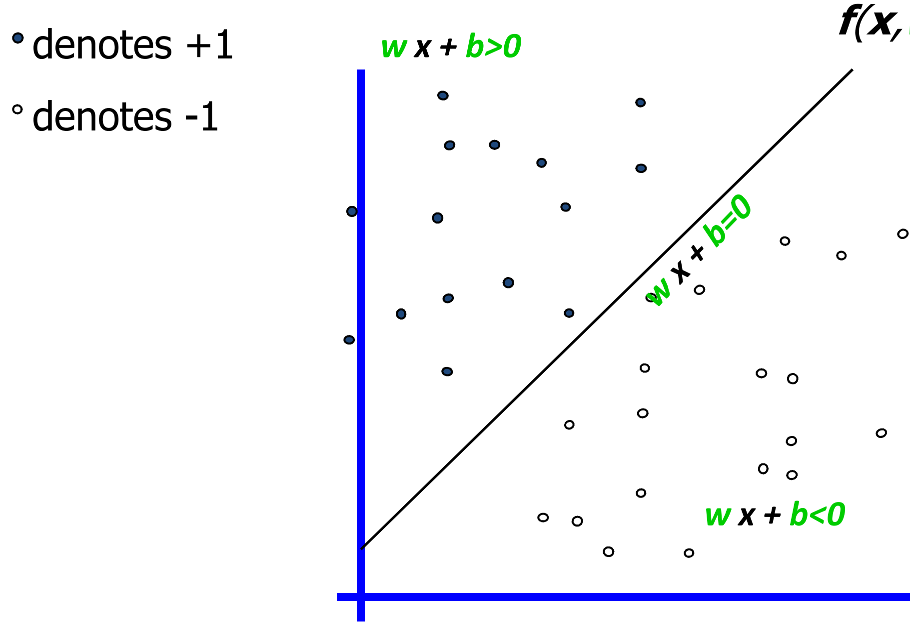
Support Vector Machines

- **Duration**: 2 hrs
- Support Vector Machine
  - Linear classifier
  - Maximum margins
  - Soft margin classification
  - Non-linear SVM
  - Kernel functions
  - Multiclass classification

Linear Classifiers

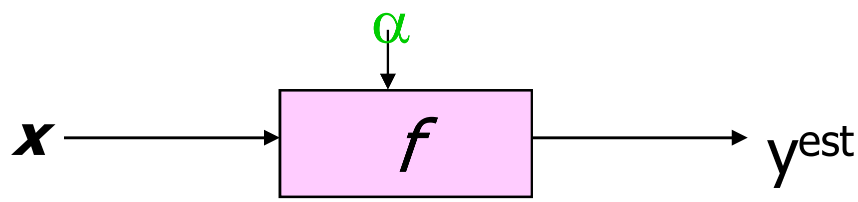


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

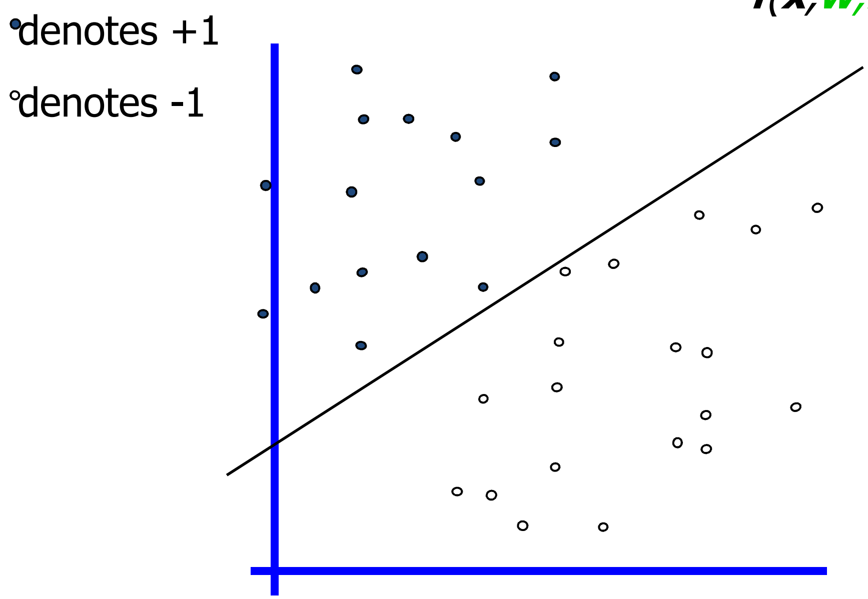


How would you classify this data?

Linear Classifiers

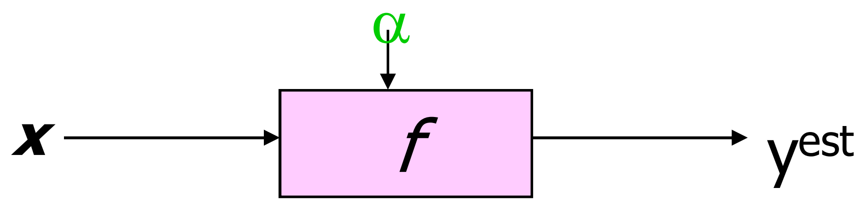


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

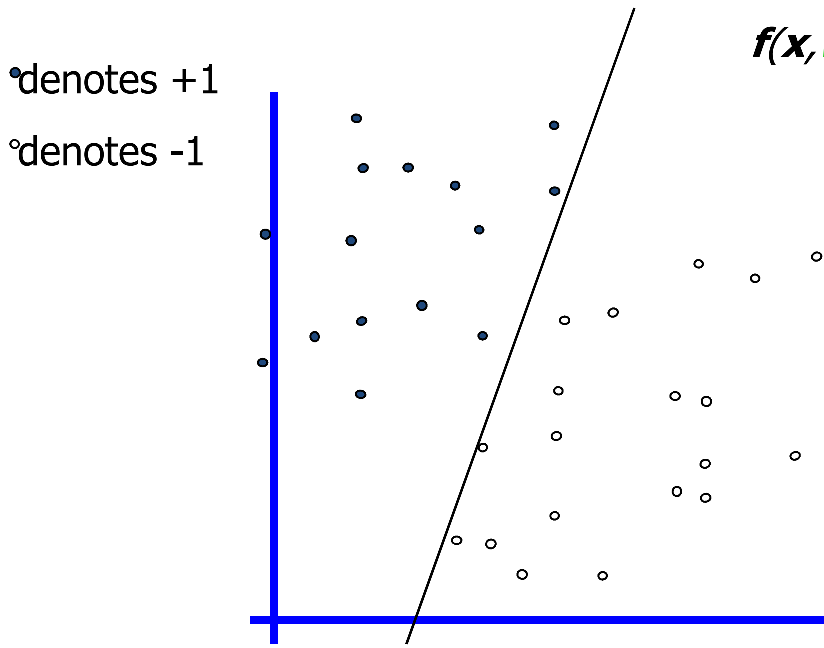


How would you classify this data?

Linear Classifiers

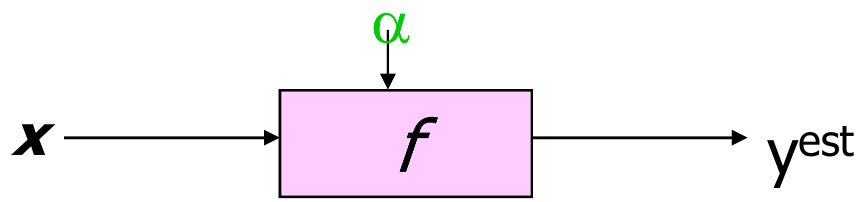


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$



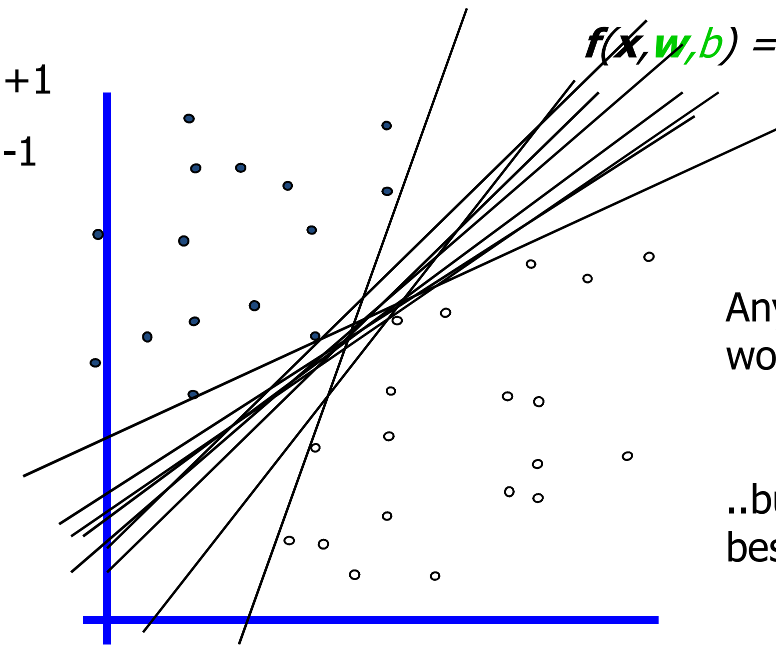
How would you classify this data?

Linear Classifiers



$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\ \boldsymbol{x} + b)$

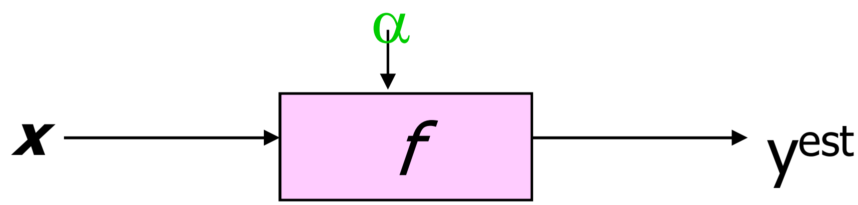
- denotes +1
- denotes -1



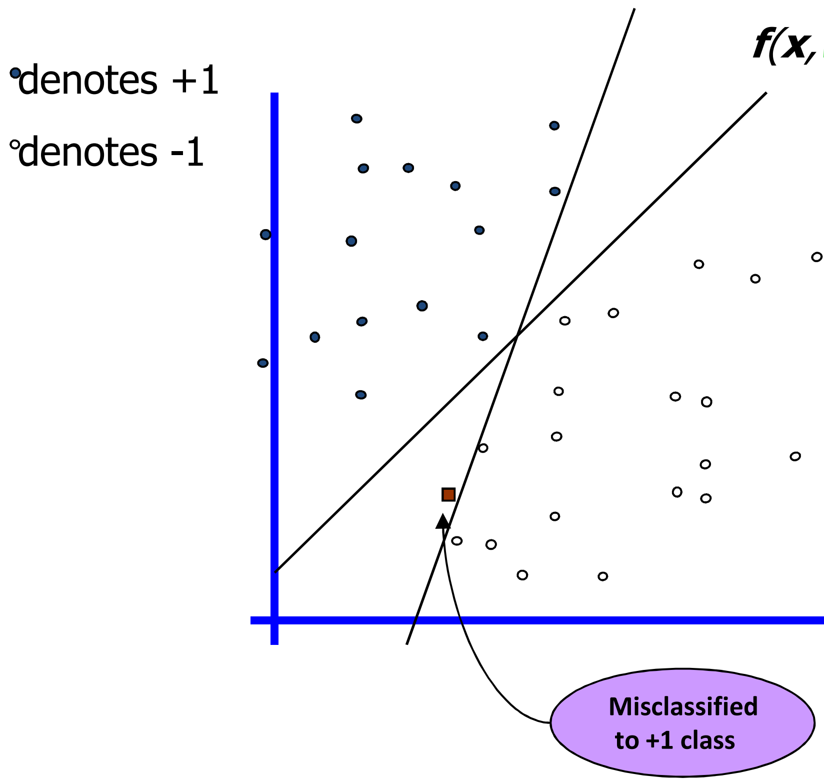
Any of these would be fine..

..but which is best?

Linear Classifiers

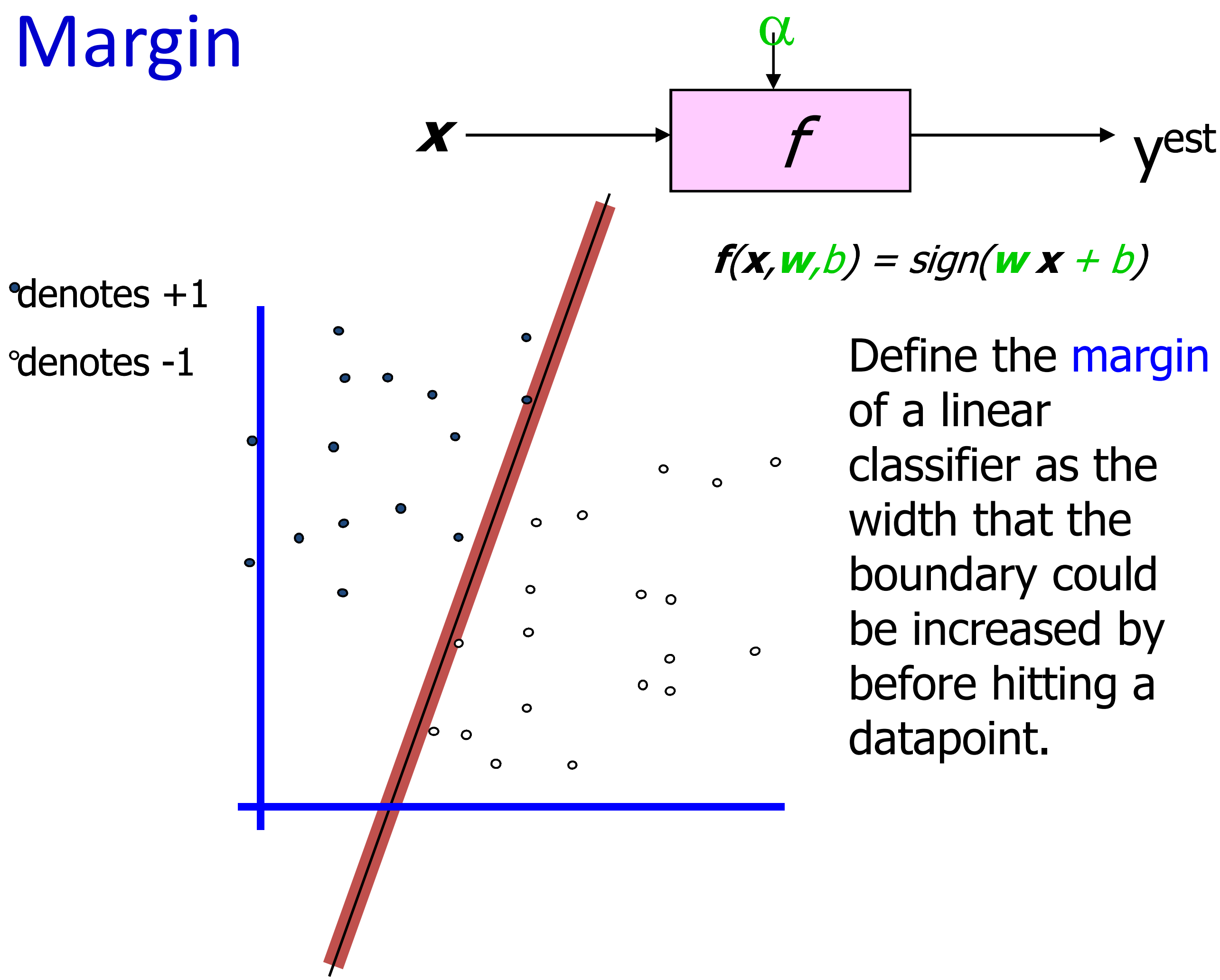


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$



How would you classify this data?

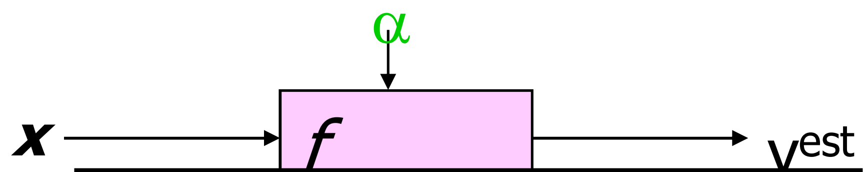
Classifiers Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin

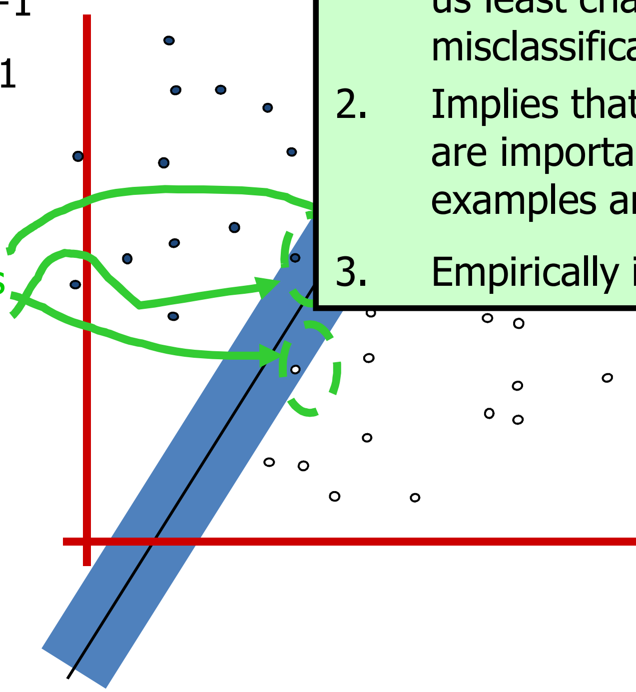


1. If we've made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

• denotes +1

° denotes -1

Support Vectors are those datapoints that the margin pushes up against



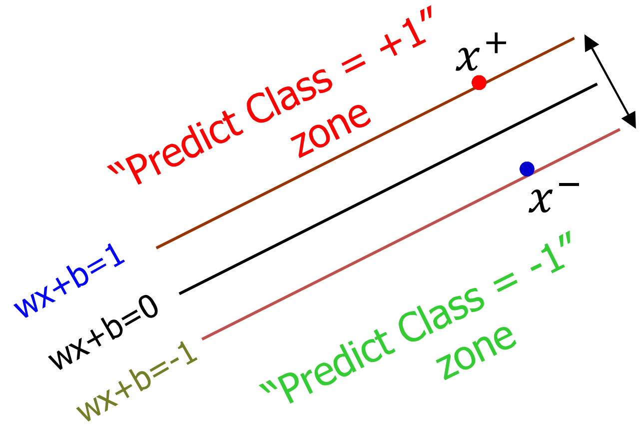
h

the maximum

margin.

This is the simplest kind of SVM (Linear SVM)

Linear SVM Mathematically



M=Margin Width

What we know:

- ***w*** *.* ***x***$^{+}$ *+ b = +1*
- ***w*** *.* ***x***$^{-}$ *+ b = -1*
- ***w*** *. (****x***$^{+}$***-x***$^{-)}$ *= 2*
- ***x***$^{+}$***-x***$^{-}$ *= λ* ***w*** *for some value of*

$$M = |x^+ - x^-| = \lambda|w|$$
$$= \frac{2}{w.w}|w| = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: **1) Correctly classify all training data**

$$wx_i + b \geq 1 \qquad \textit{if } y_i = +1$$
$$wx_i + b \leq 1 \qquad \textit{if } y_i = -1$$
$$y_i(wx_i + b) \geq 1 \qquad \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2}w^t w$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

- Minimize $\Phi(w) = \frac{1}{2}w^t w$

  subject to $y_i(wx_i + b) \geq 1 \qquad \forall i$

The Optimization Problem Solution

- The solution has the form:

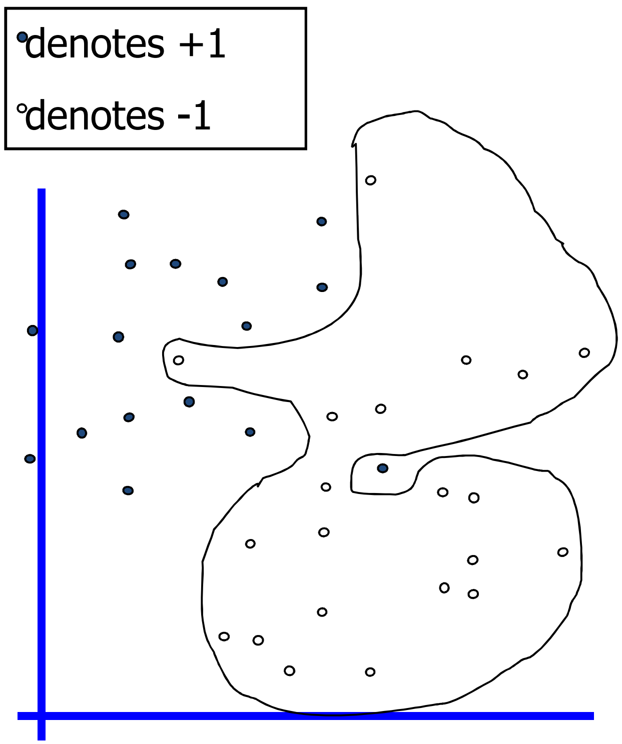
$$\boxed{\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w^T x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i}^{\mathbf{T}}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x_i}$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^{\mathbf{T}}\mathbf{x_j}$ between all pairs of training points.

Dataset with noise

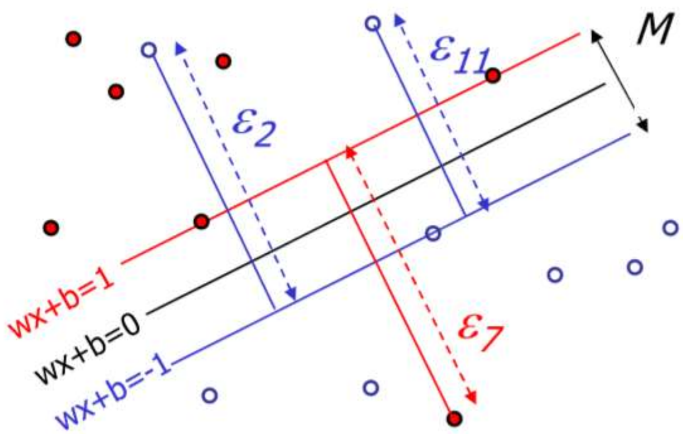


- Hard Margin: So far we require all data points be classified correctly
- - No training error
- What if the training set is noisy?
- - Solution 1: use very powerful kernels

OVERFITTING!

Soft Margin Classification

Slack variables ξi can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Hard Margin v.s. Soft Margin

- **The old formulation:**

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\,(\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + \mathrm{b}) \geq 1$

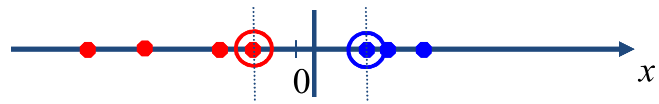
- **The new formulation incorporating slack variables:**

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\,(\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

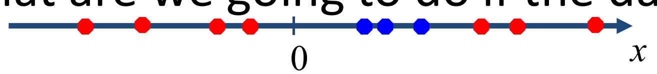
- **Parameter *C* can be viewed as a way to control overfitting.**

Non-linear SVMs

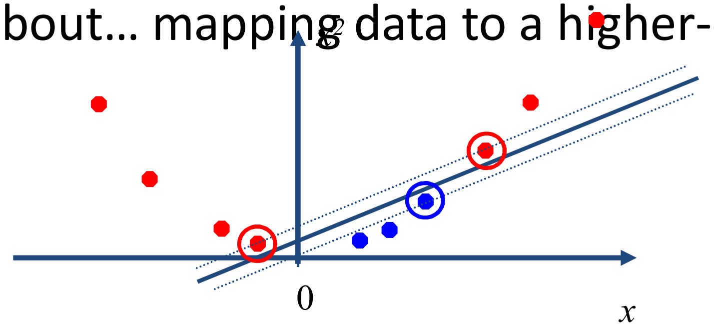
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

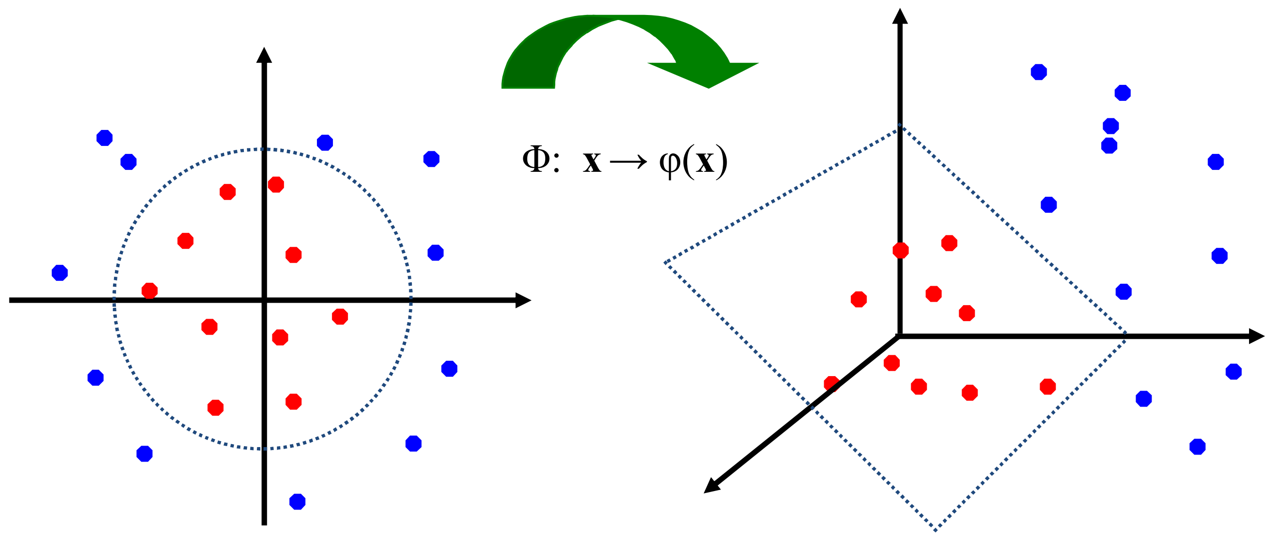


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



The “Kernel Trick”

- The linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^Tx_j$
- If every data point is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \phi(x)$, the dot product becomes:
    - $K(x_i,x_j)= \phi(x_i)^T\phi(x_j)$
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

2-dimensional vectors $x = [x_1 \ \ x_2]$; let $K(x_i,x_j) = (1 + x_i^Tx_j)^2$,

Need to show that $K(x_i,x_j) = \phi(x_i)^T\phi(x_j)$:

$$
\begin{aligned}
K(x_i,x_j) &= (1 + x_i^Tx_j)^2, \\
&= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} \\
&= [1 \ \ x_{i1}^2 \ \ \sqrt{2}\, x_{i1}x_{i2} \ \ x_{i2}^2 \ \ \sqrt{2}x_{i1} \ \ \sqrt{2}x_{i2}]^T [1 \ \ x_{j1}^2 \ \ \sqrt{2}\, x_{j1}x_{j2} \ \ x_{j2}^2 \ \ \sqrt{2}x_{j1} \ \ \sqrt{2}x_{j2}] \\
&= \phi(x_i)^T\phi(x_j), \quad \text{where } \phi(x) = [1 \ \ x_1^2 \ \ \sqrt{2}\, x_1x_2 \ \ x_2^2 \ \ \sqrt{2}x_1 \ \ \sqrt{2}x_2]
\end{aligned}
$$

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i},\mathbf{x_j}) = (1 + \mathbf{x_i}^T\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j}) = \tanh(\beta_0 \mathbf{x_i}^T\mathbf{x_j} + \beta_1)$

Use SVM software package (e.g. liblinear, libsvm, …)

$$\theta$$

Need to specify:

Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

Gaussian kernel:

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

Kernel (similarity) functions:

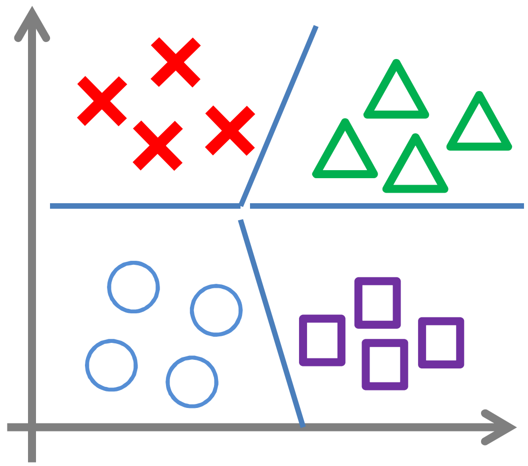
```
function f = kernel(x1,x2)
```

$$f = \exp\left(-\frac{||\,\mathbf{x1} - \mathbf{x2}\,||^2}{2\sigma^2}\right)$$

```
return
```

Note: Do perform feature scaling before using the Gaussian kernel.

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$)

Logistic regression vs. SVMs

$n =$number of features ($x \in \mathbb{R}^{n+1}$), $m =$ number of training examples
If n is large (relative to m): (E.g, $n \geq m, n = 10\ 000, m = 10 - 1000$)
Use logistic regression, or SVM without a kernel ("linear kernel")

If n is small, m is intermediate: $(n = 1 - 1000, m = 10 - 10000)$
Use SVM with Gaussian kernel

If n is small, m is large: $(n = 1 - 1000, m = 50000)$
Create/add more features, then use logistic regression or SVM without a kernel

Neural network likely to work well for most of these settings, but may be slower to train.

References

- Machine Learning, Andrew Ng, coursera.org
- SVM tutorial, Prof. Andrew Moore http://www.cs.cmu.edu/~awm/tutorials