Perceptrons, SVMs, and Friends:

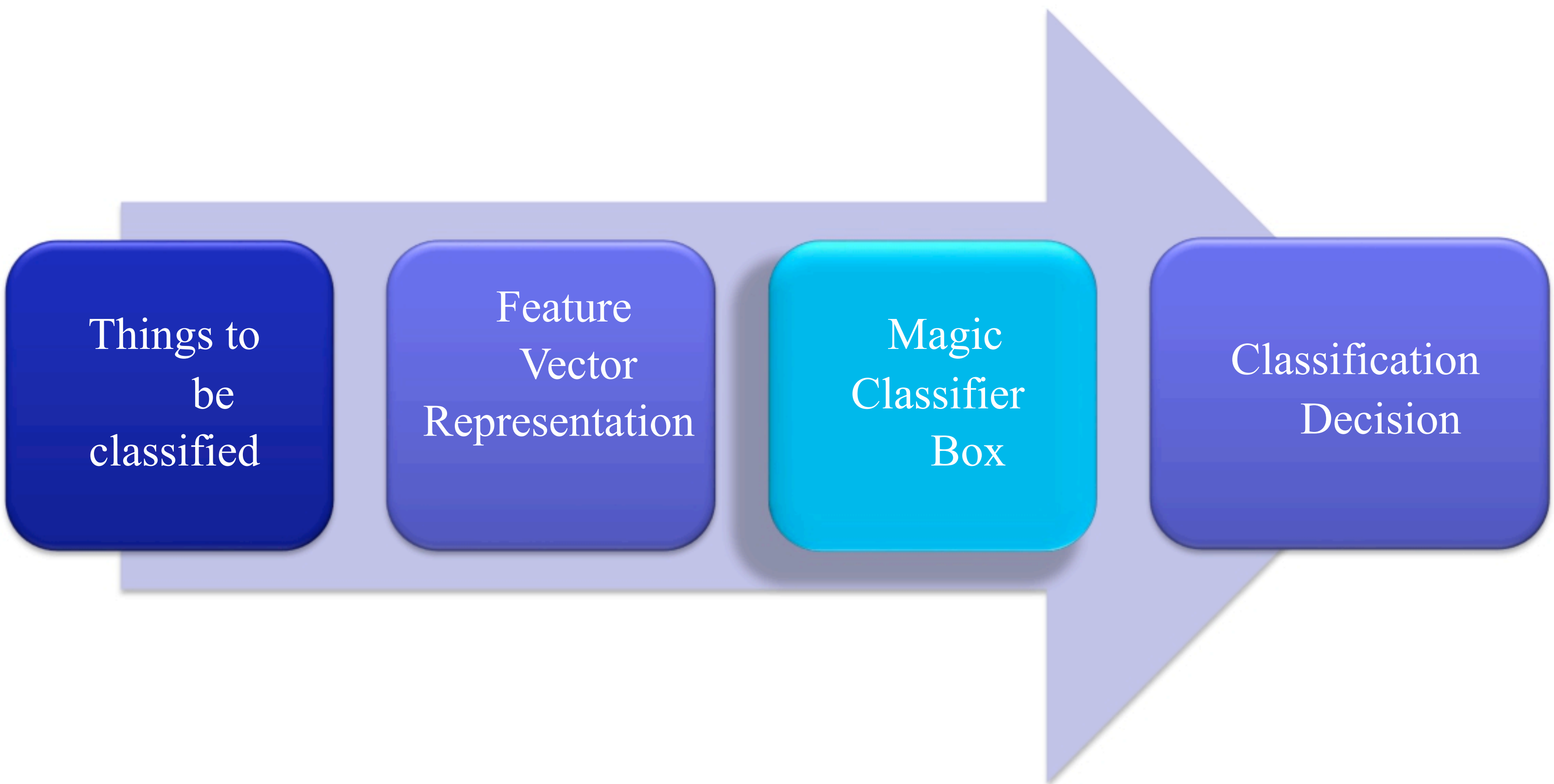
Some *Discriminative* Models for Classification

The Automatic Classification Problem

- **Assign object/event or sequence of objects/events to one of a given finite set of categories.**
  - *Fraud detection* for credit card transactions, telephone calls, etc.
  - *Worm detection* in network packets
  - *Spam filtering* in email
  - *Recommending articles* , books, movies, music
  - *Medical diagnosis*
  - *Speech recognition*
  - *OCR* of handwritten letters
  - Recognition of specific astronomical images
  - Recognition of specific DNA sequences
  - Financial investment

- **Machine Learning methods provide one set of approaches to this problem**

Universal Machine Learning Diagram



Example: handwritten digit recognition

Machine learning algorithms that

- **Automatically cluster these images**
- **Use a training set of labeled images to learn to classify new images**
- **Discover how to account for variability in writing style**

A machine learning algorithm development pipeline: minimization

Problem statement

↓

Mathematical description of
a cost function

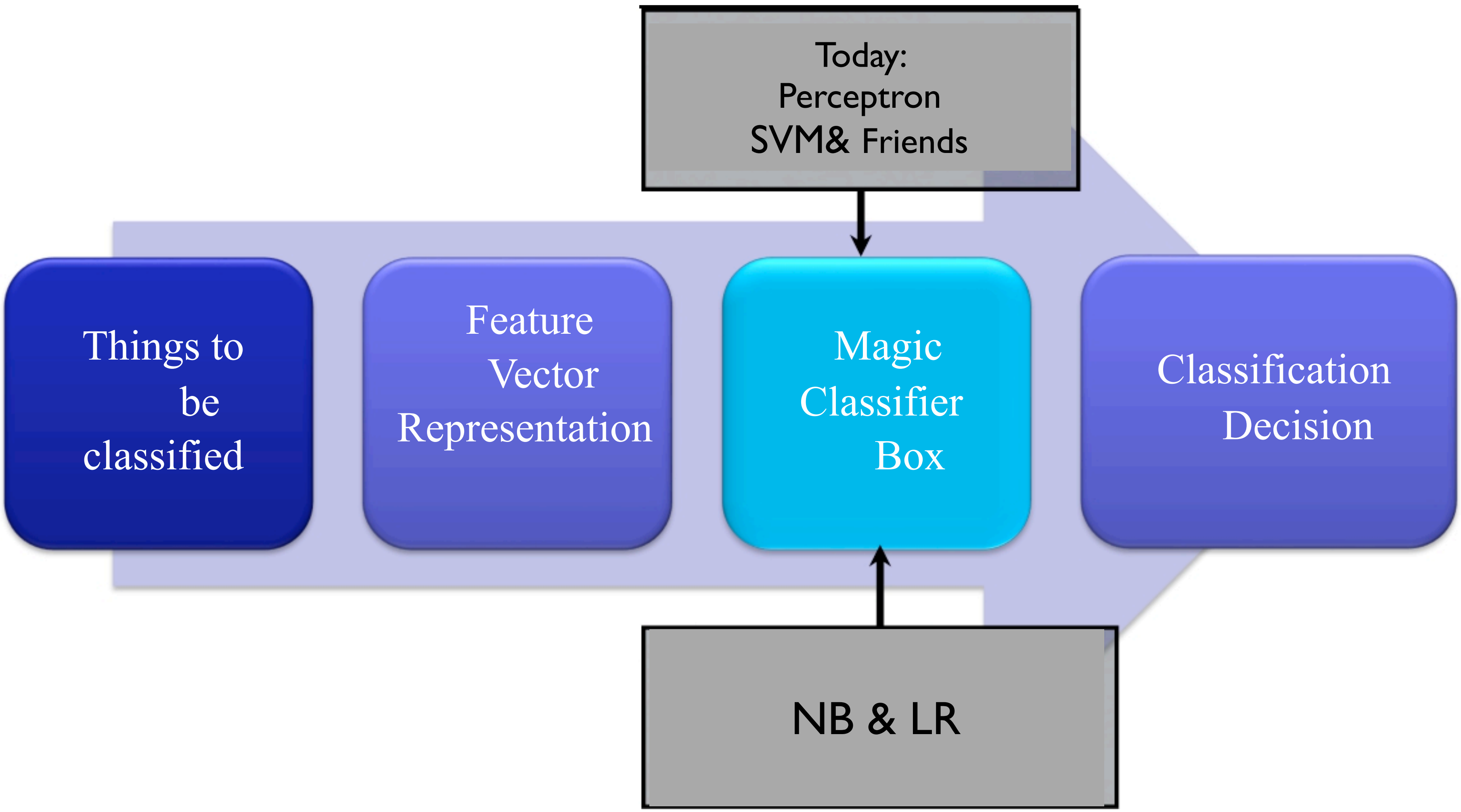
↓

Mathematical description of
how to minimize/maximize
the cost function

↓

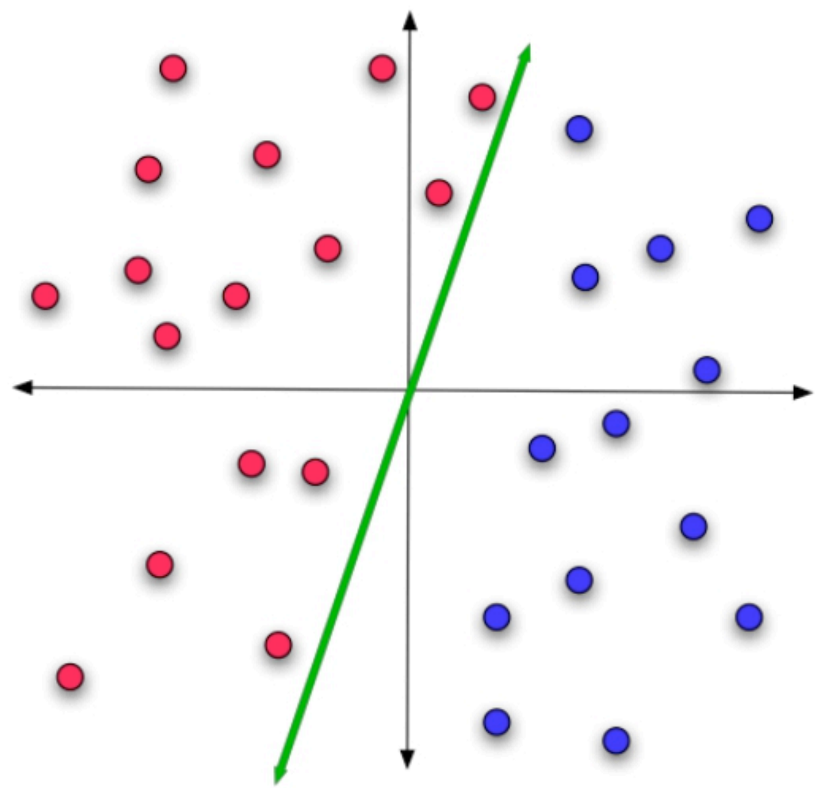
Implementation

Universal Machine Learning Diagram



Linear Classification: Informal…

Find a (line, plane, hyperplane) that divides the red points from the blue points….



Hyperplane

A **hyperplane** can be defined by $c = \vec{w} \cdot \vec{x}$

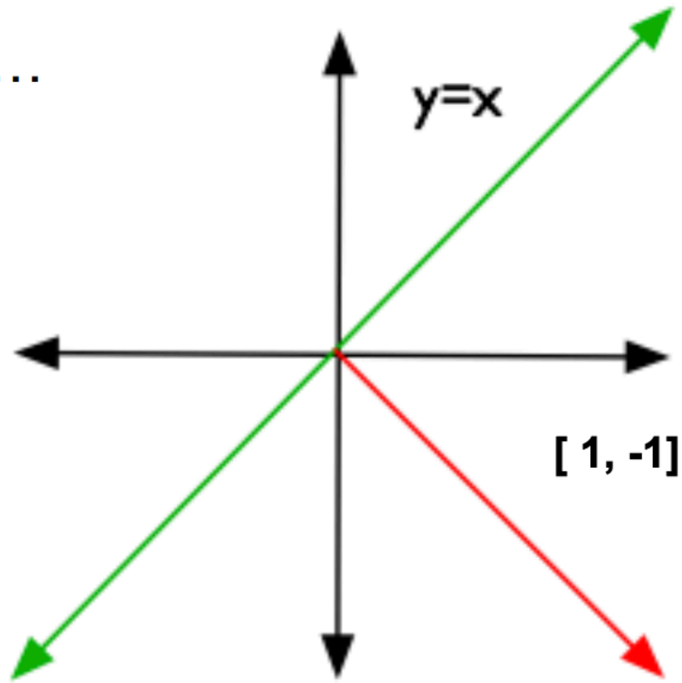
Or more simply (renormalizing) by $0 = \vec{w} \cdot \vec{x}$

Consider a two-dimension example…

$$0 = [1, -1] \begin{bmatrix} x \\ y \end{bmatrix}$$

$$0 = x - y$$

$$y = x$$



Linear Classification: Slightly more formal

Input encoded as feature vector $\vec{x}$

Model encoded as $\vec{w}$

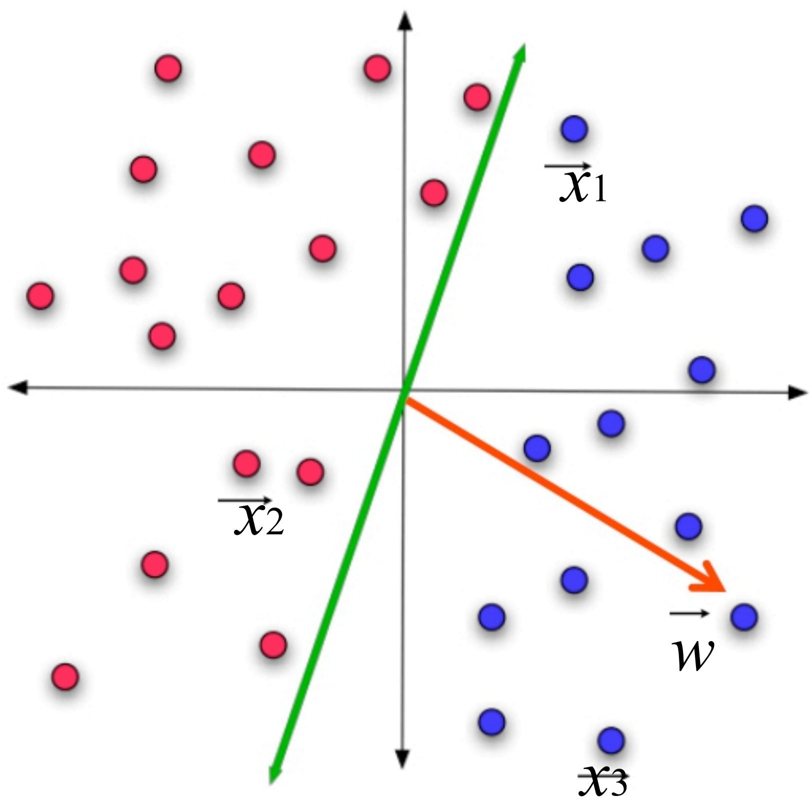
Just return $y = \vec{w} \cdot \vec{x}$!

sign(y) tell us the class:

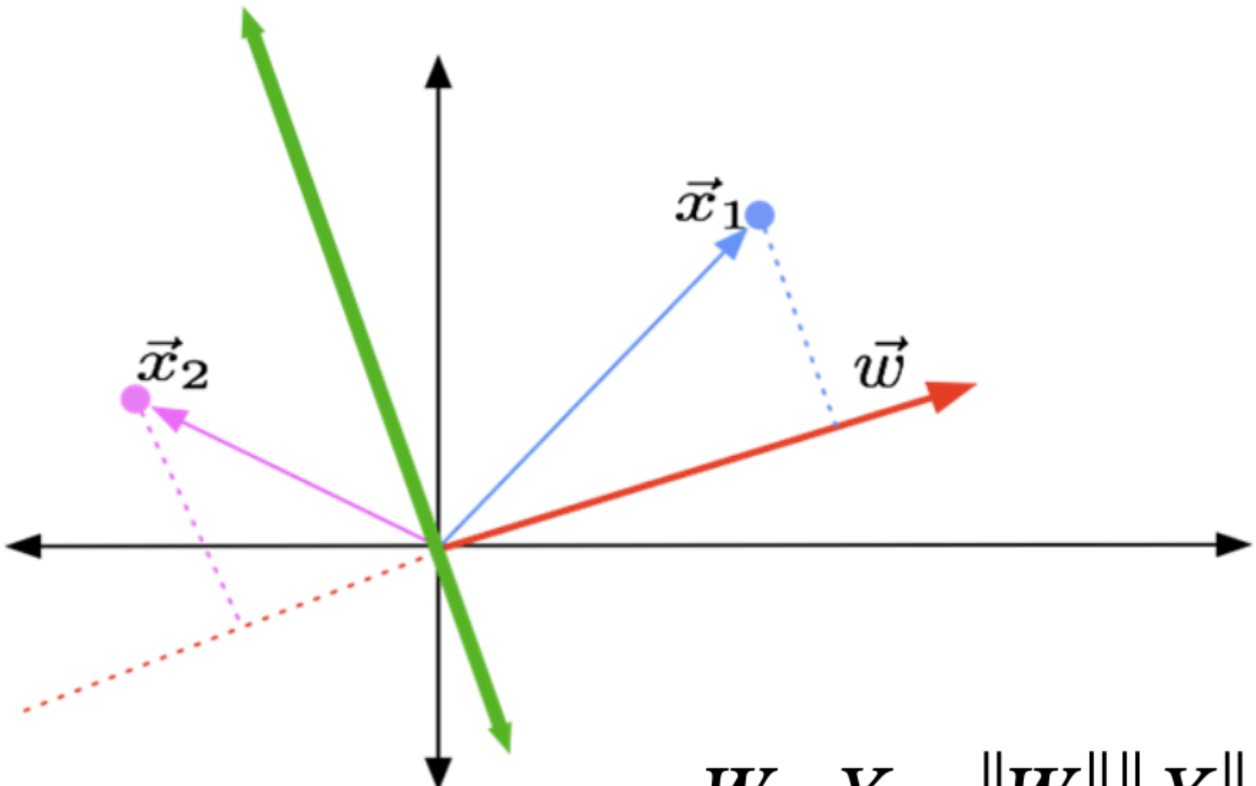
+ - blue

- - red

(All vectors normalized to length 1, for simplicity)



Computing the sign…

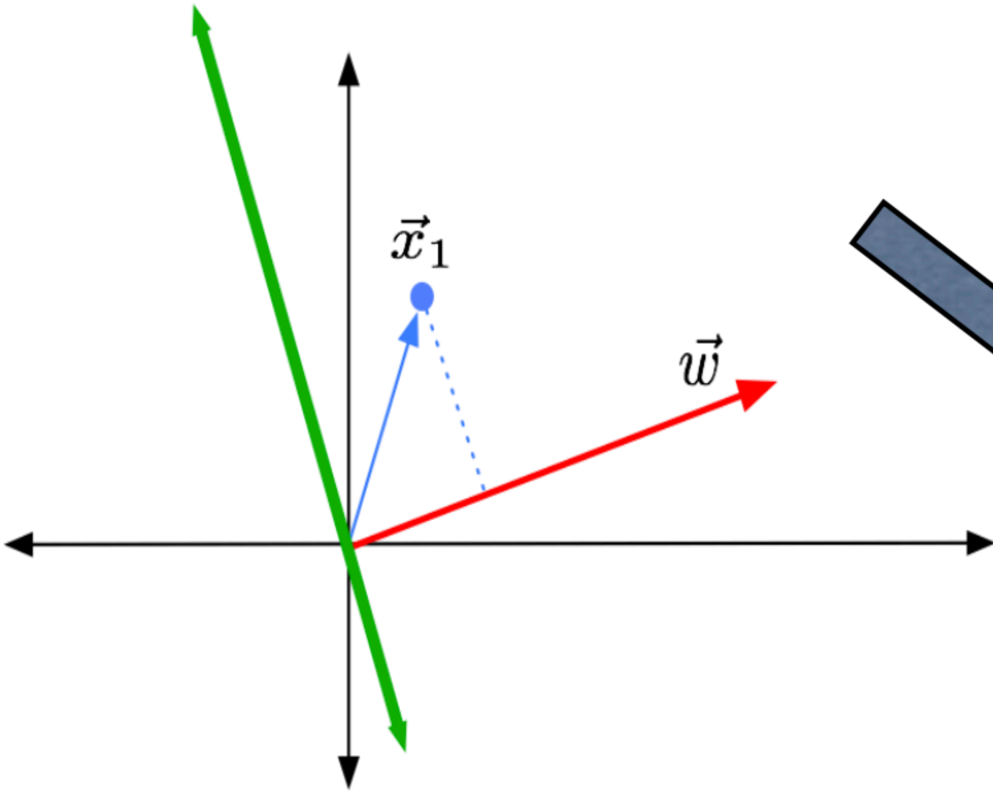


One definition of dot product: $W \cdot X = \|W\|\|X\| \cos\theta$

So $sign(W \cdot X) = sign(\cos\theta)$

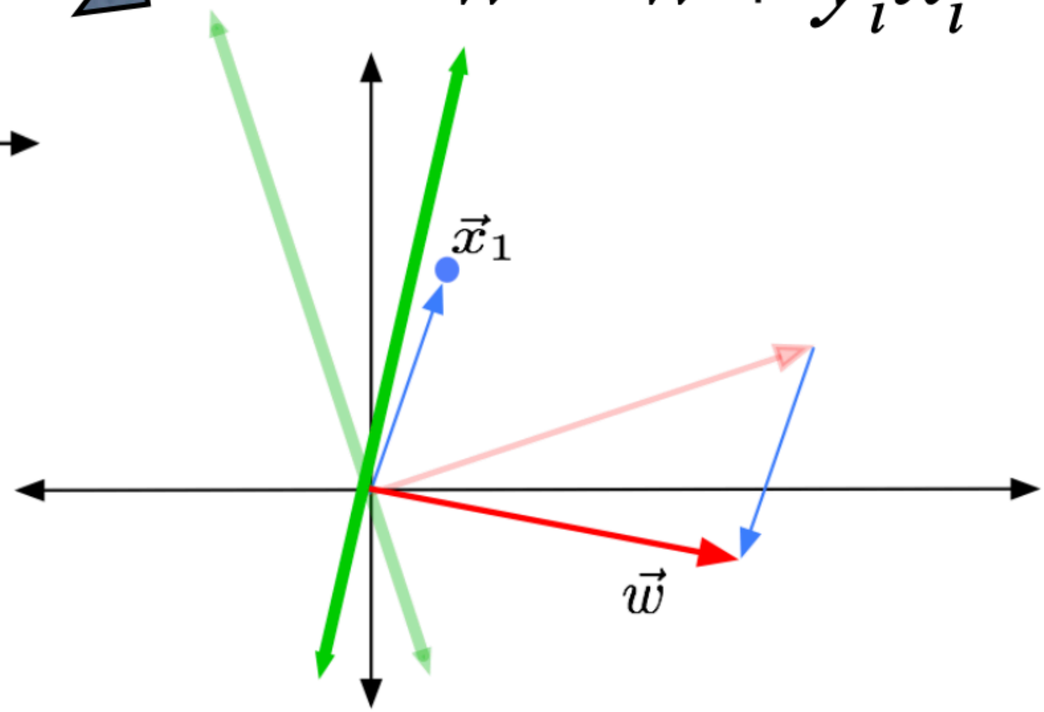
Let $y = sign(\cos\theta)$

Perceptron Update Example



If $\vec{x}_1$ is supposed to be on the other side….

$$\vec{w} = \vec{w} + y_i \vec{x}_i$$



Perceptron Learning Algorithm

Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$
Output: A classifying hyperplane $\vec{w}$
Randomly initialize $\vec{w}$;
while *model* $\vec{w}$ *makes errors on the training data* **do**
    for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**
        Let $\hat{y} = sign(\vec{w} \cdot \vec{x}_i)$;
        if $\hat{y} \neq y_i$ **then**
            $\vec{w} = \vec{w} + y_i \vec{x}_i$;
        end
    end
end

Perceptron Learning Algorithm

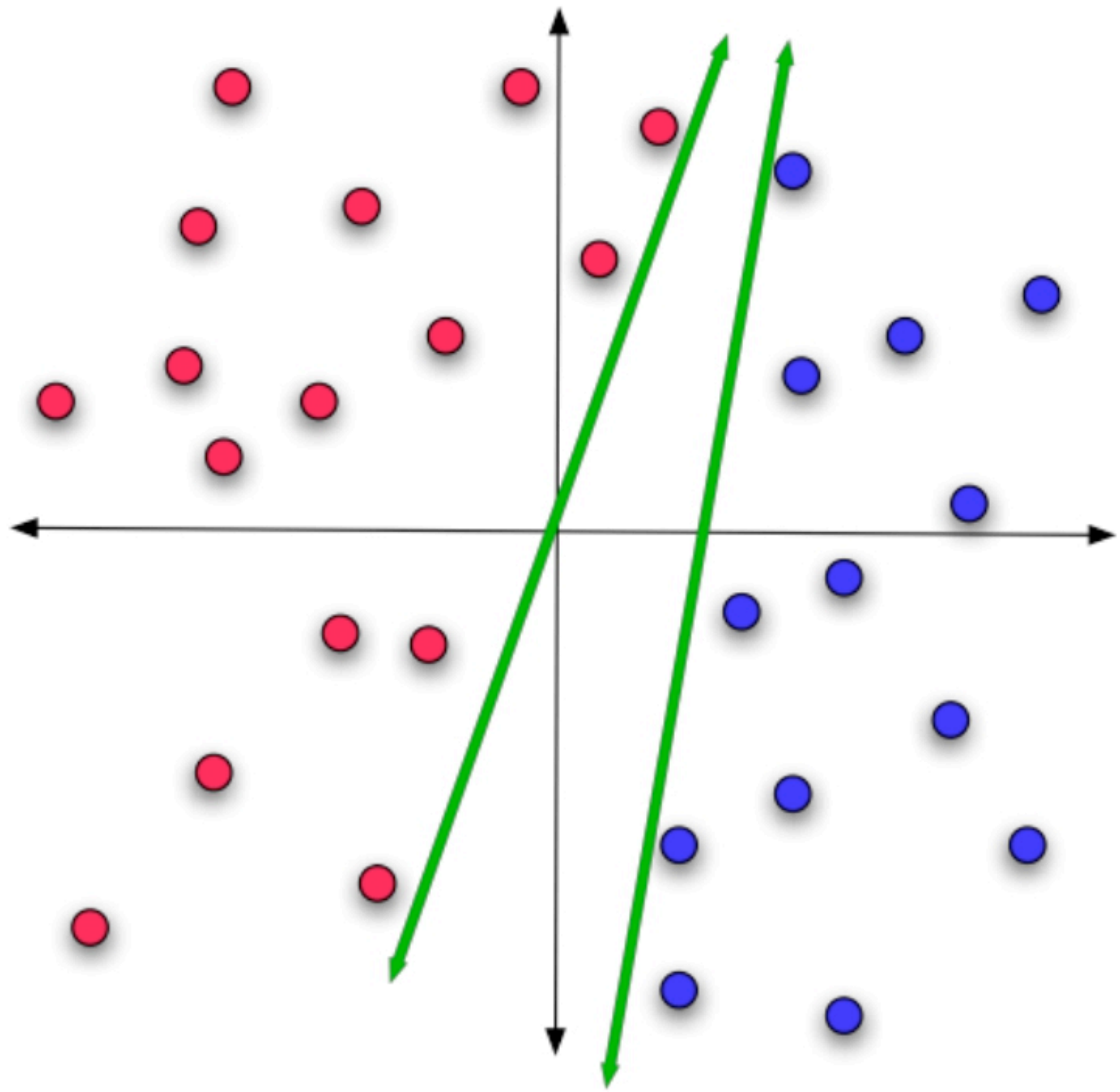
Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$
Output: A classifying hyperplane $\vec{w}$
Randomly initialize $\vec{w}$;
while *model* $\vec{w}$ *makes errors on the training data* **do**
  for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**
    Let $\hat{y} = sign(\vec{w} \cdot \vec{x}_i)$;
    if $\hat{y} \neq y_i$ **then**
      $\vec{w} = \vec{w} + y_i \vec{x}_i$;
    end
  end
end

Converges if the training set is ***linearly separable***

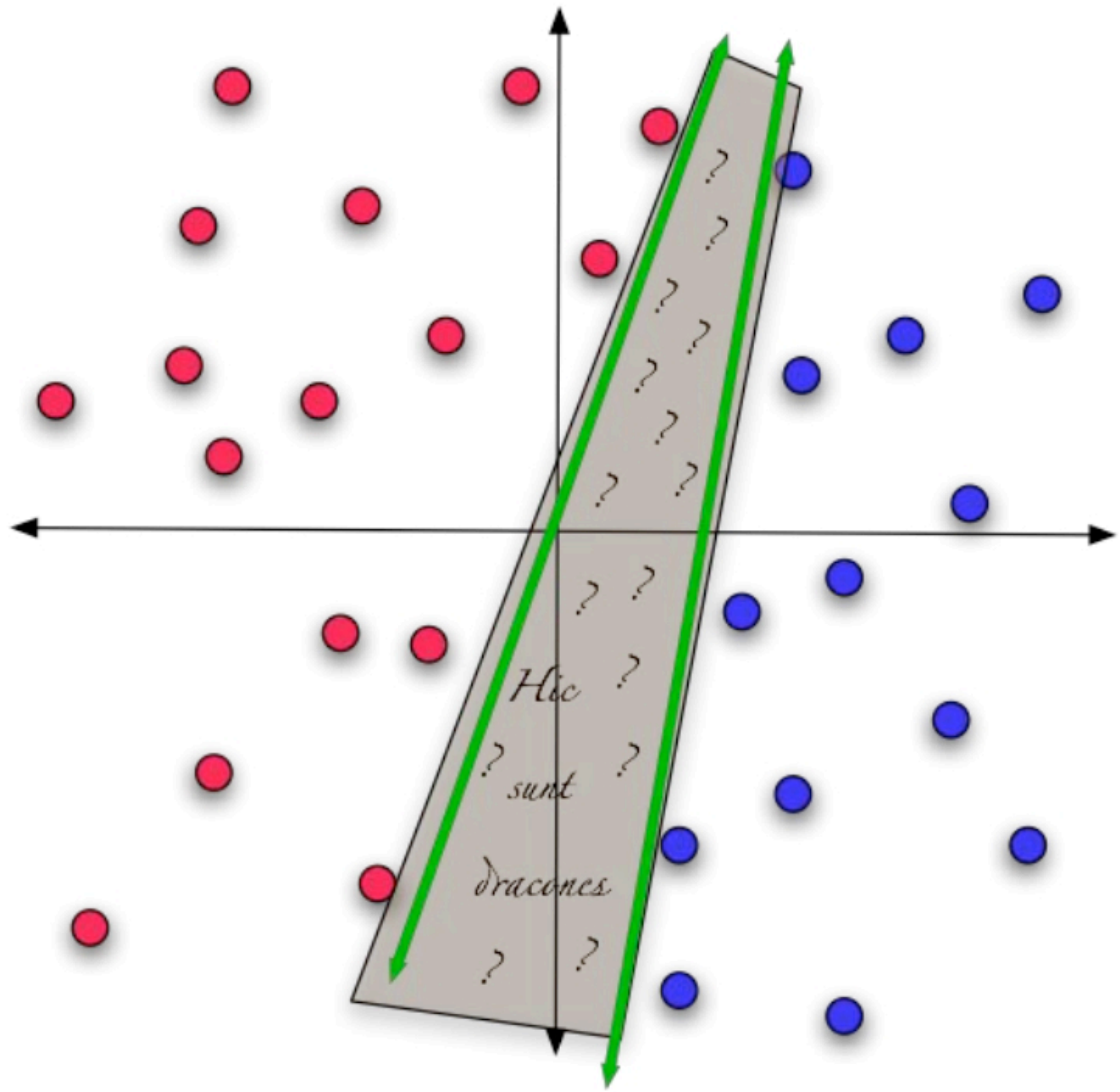
May not converge if the training set is ***not*** *linearly separable*

Support vector machines

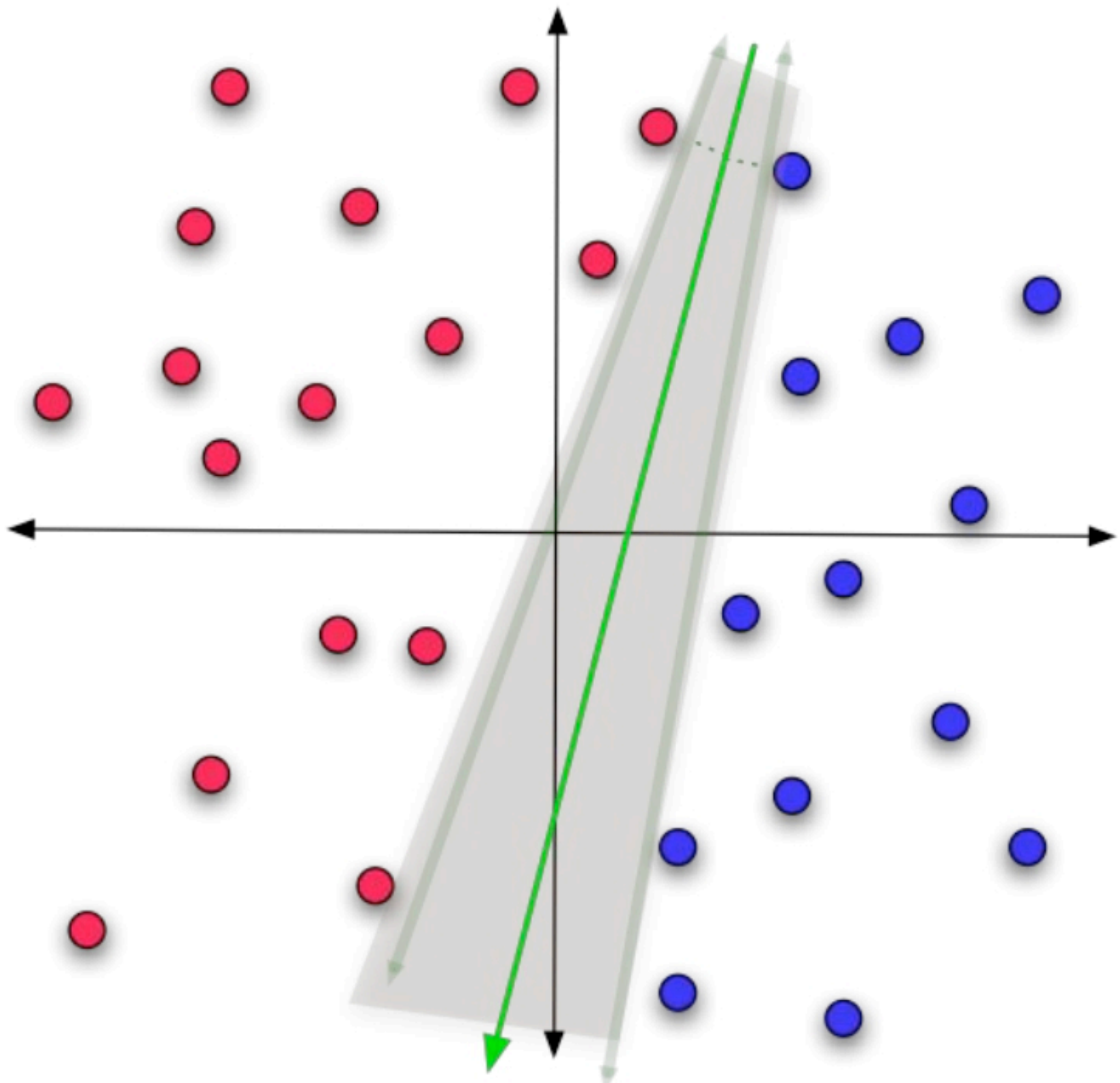
What's wrong with these hyperplanes?



They're unjustifiably biased!

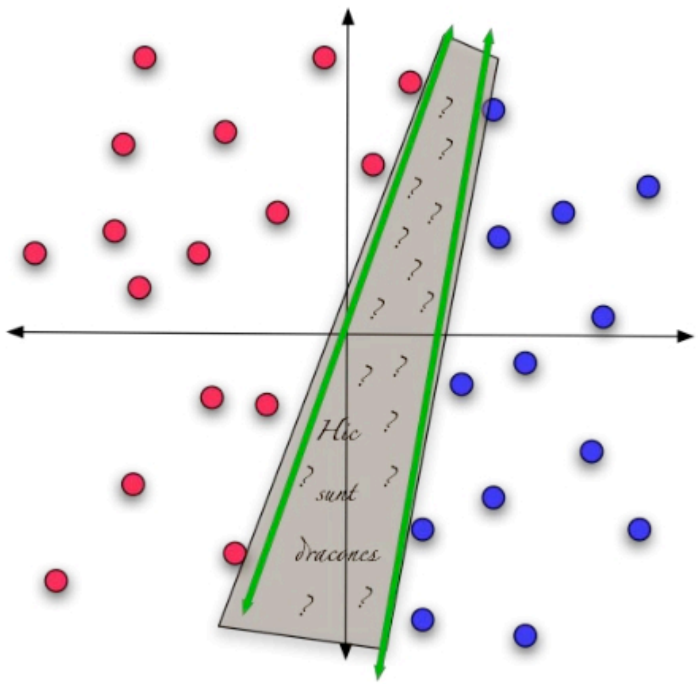


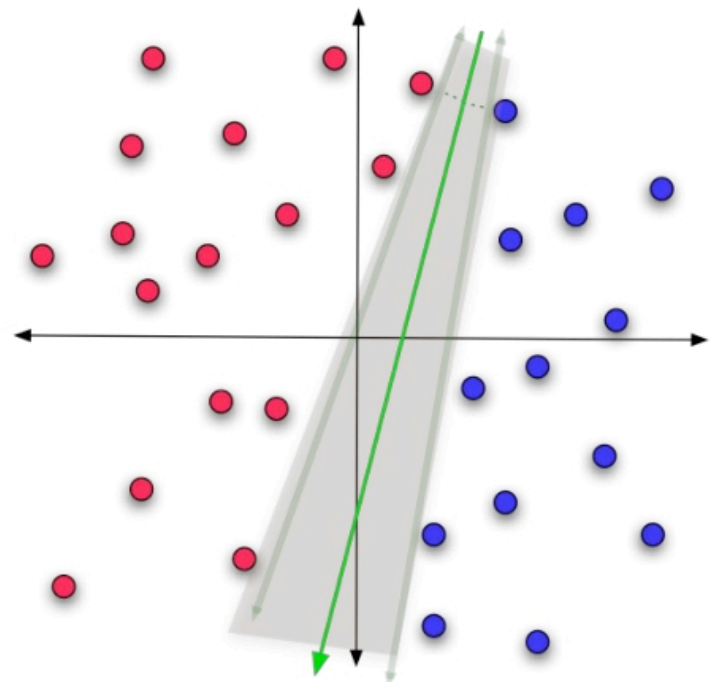
A less biased choice



Margin

- the distance to closest point in the training data
- We tend to get better generalization to **unseen data** if we choose the separating hyperplane which *maximizes the margin*

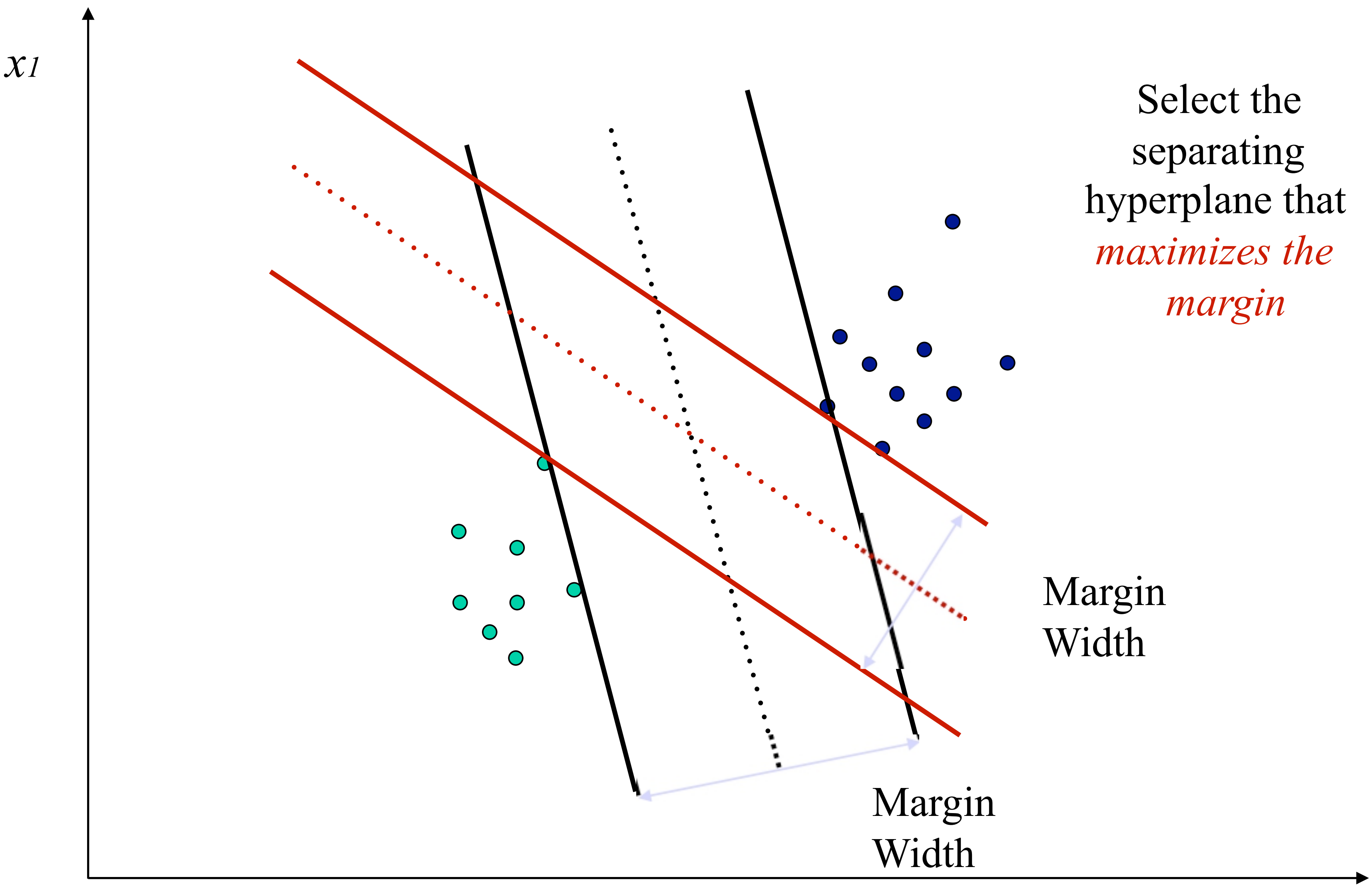




Support Vector Machines

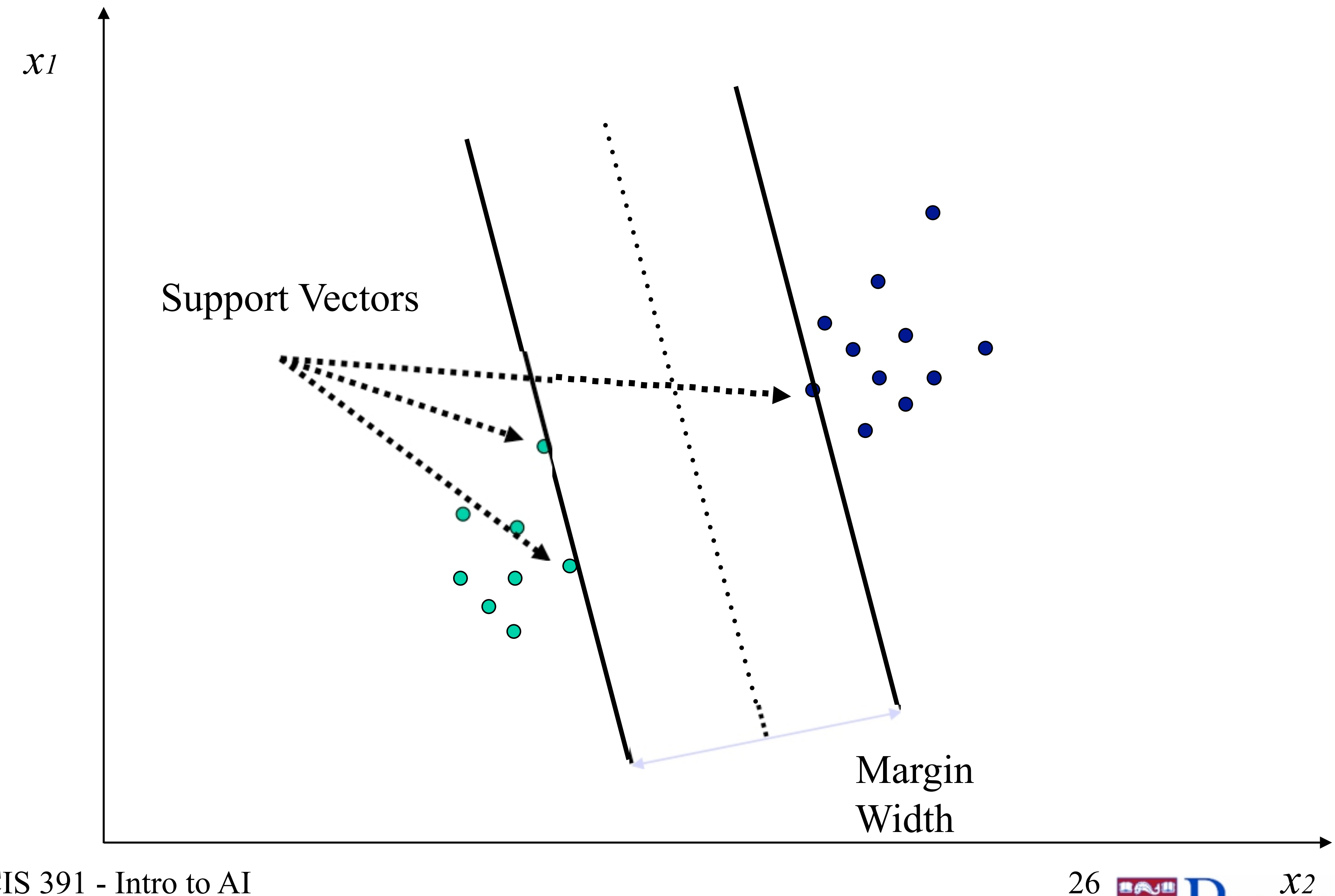
- A learning method which *explicitly calculates the maximum margin hyperplane* by solving a gigantic quadratic programming minimization problem.
- Among the very highest -performing traditional machine learning techniques.
- But it's relatively slow and quite complicated.

Maximizing the Margin



Select the separating hyperplane that *maximizes the margin*

Support Vectors

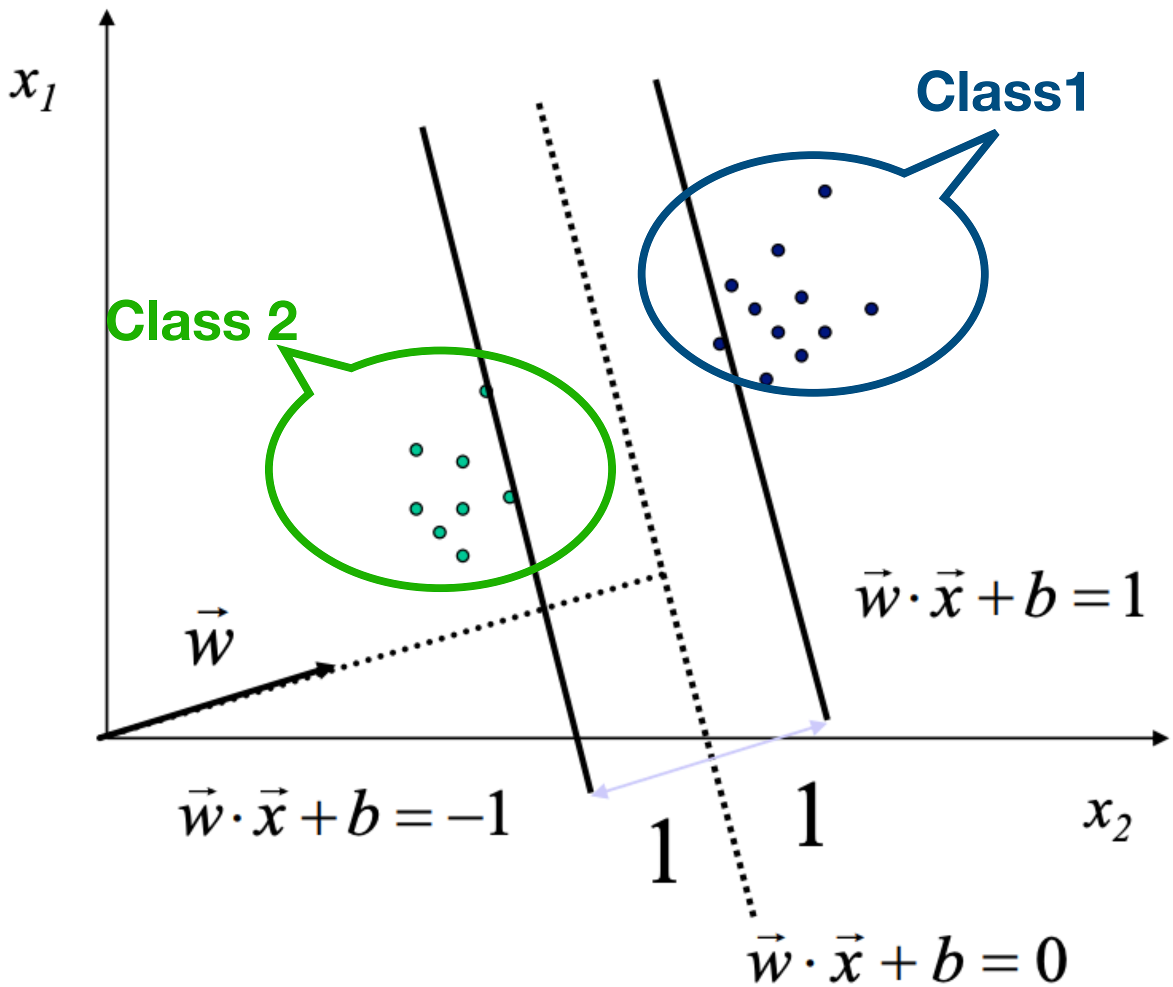


Support Vector Machines

- **A learning method which *explicitly calculates the maximum margin hyperplane*.**

\

Setting Up the Optimization Problem



The maximum margin can be characterized as a solution to an optimization problem:

$$\max.\ \frac{2}{\|w\|}$$

$$s.t.\ (w\cdot x+b)\geq 1,\ \forall x \text{ of class 1}$$

$$(w\cdot x+b)\leq -1,\ \forall x \text{ of class 2}$$

Define the margin (what ever it turns out to be) to be *one unit of width.*

Setting Up the Optimization Problem

- If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite

$$(w \cdot x_i + b) \geq 1, \;\; \forall x_i \text{ with } y_i = 1$$
$$(w \cdot x_i + b) \leq -1, \;\; \forall x_i \text{ with } y_i = -1$$

- as

$$y_i (w \cdot x_i + b) \geq 1, \;\; \forall x_i$$

- So the problem becomes:

$$\text{max. } \frac{2}{\|w\|} \quad s.t. \; y_i (w \cdot x_i + b) \geq 1, \;\; \forall x_i$$

or

$$\text{min. } \frac{1}{2}\|w\|^2 \quad s.t. \; y_i (w \cdot x_i + b) \geq 1, \;\; \forall x_i$$

Linear, (Hard-Margin) SVM Formulation

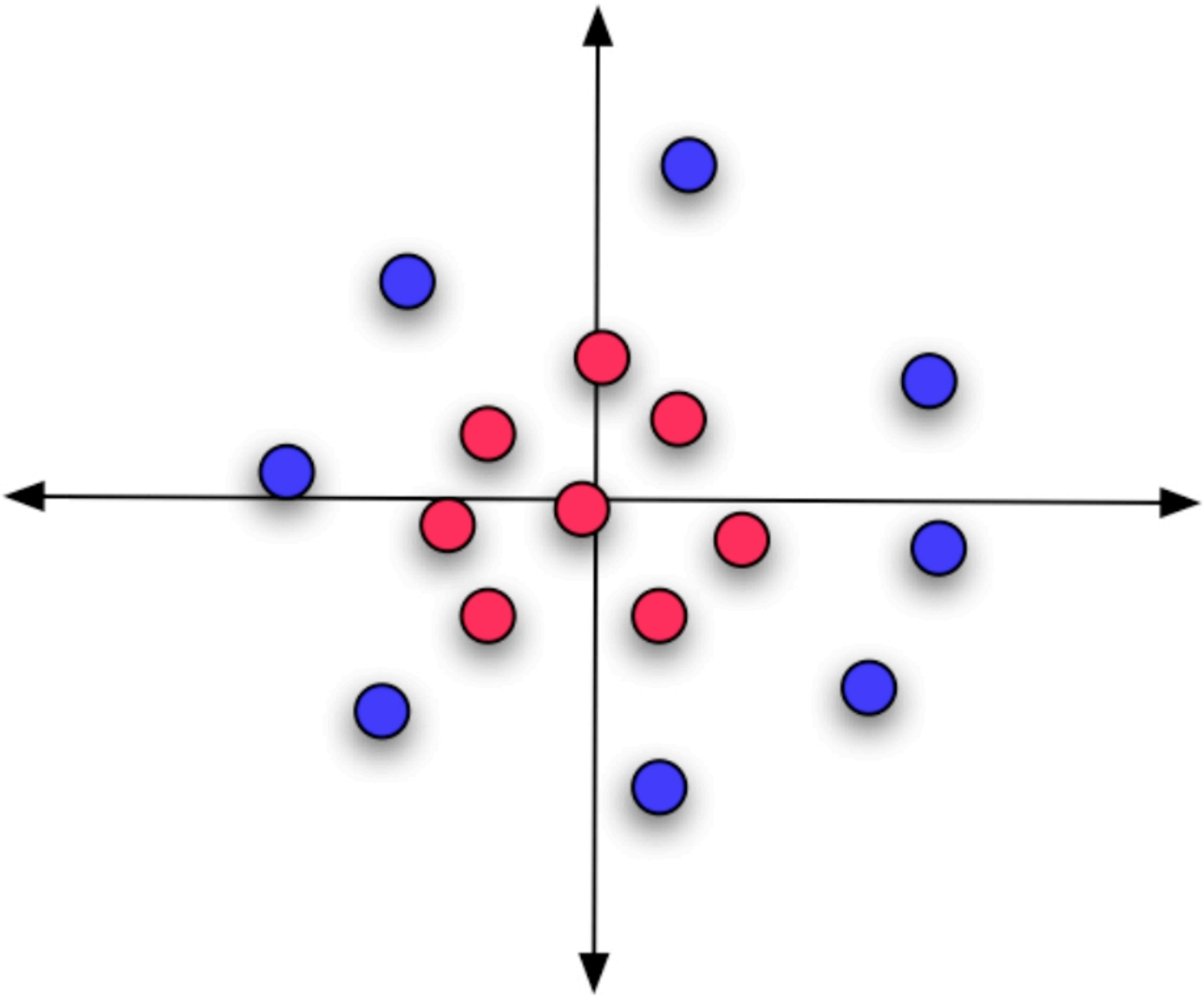
- **Find *w,b* that solves**

$$\min.\ \frac{1}{2}\|w\|^2$$

$$s.t.\ y_i(w \cdot x_i + b) \geq 1,\ \forall x_i$$

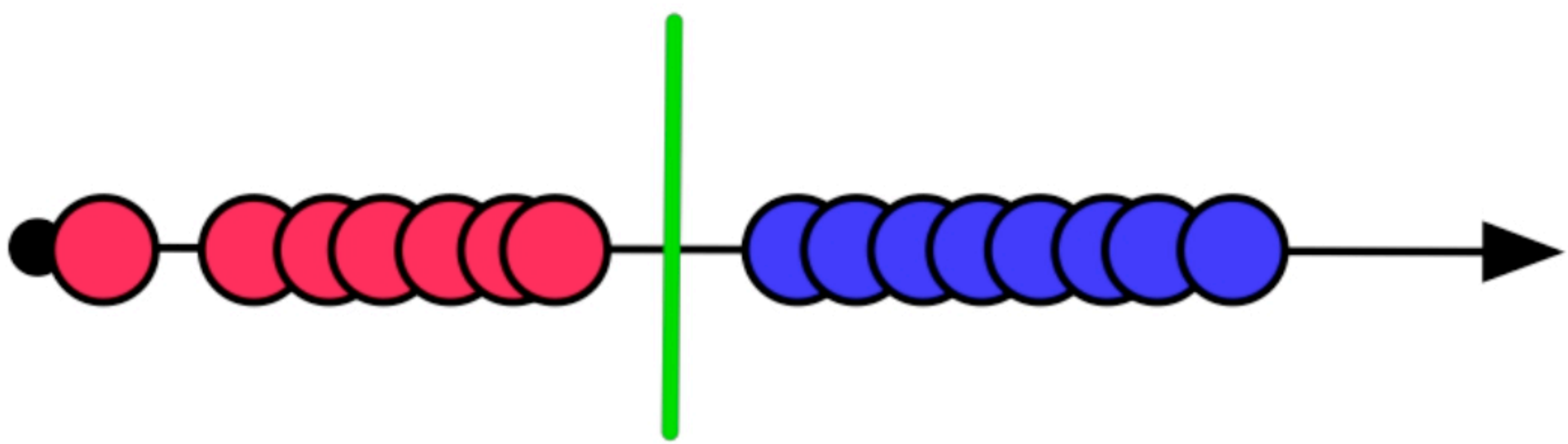
- **Problem is *convex*, so there is a unique global minimum value (when feasible)**
- **There is also a unique minimizer, i.e. weight and *b* value that provides the minimum**
- **Quadratic Programming**
  - very efficient computationally with procedures that take advantage of the special structure

What if it isn’t separable?



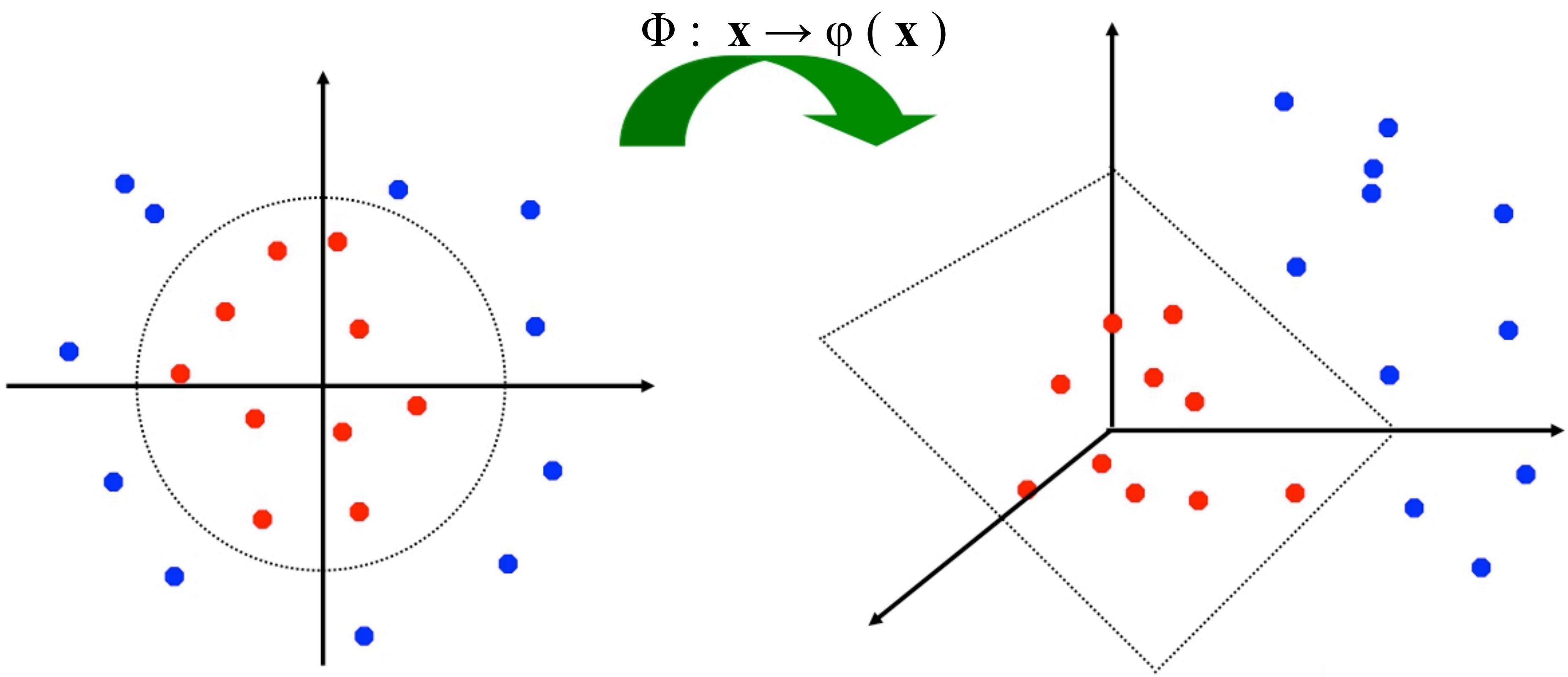
Project it to someplace where it is!

$$\phi(\langle x, y \rangle) = x^2 + y^2$$



Non - linear SVMs: Feature spaces

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



Kernel Trick

- **If our data isn't linearly separable, we can define a projection $\Phi(x_i)$ to map it into a much higher dimensional feature space where it is.**

- **For SVM where everything can be expressed as the dot products of instances this can be done efficiently using the `kernel trick':**
  - A kernel K is a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$
  - Then, we never need to explicitly map the data into the high-dimensional space to solve the optimization problem – magic!!

SVMs vs. other ML methods

Examples from the NIST database of handwritten digits

- 60K labeled digits 20x20 pixels 8bit greyscale values

- Learning methods
  - 3 -nearest neighbors
  - Hidden layer neural net
  - Specialized neural net (LeNet)
  - Boosted neural net
  - SVM
  - SVM with kernels on pairs of nearby pixels + specialized transforms
  - Shape matching (vision technique)
- Human error: on similar US Post Office database 2.5%.

Performance on the NIST digit set (2003)

	3 -NN	Hidden Layer NN	LeNet	Boosted LeNet	SVM	Kernel SVM	Shape Match
Error %	2.4	1.6	0.9	0.7	1.1	0.56	0.63
Run time (millisec /digit)	1000	10	30	50	2000	200	
Memory (MB)	12	.49	.012	.21	11		
Training time (days)	0	7	14	30	10		

In 2010) (.35% error) by a neural network