



# Support Vector Machines & Kernels

# Outline

- Prediction

  - Why might predictions be wrong?

- Support vector machines

  - Doing really well with linear models

- Kernels

  - Making the non-linear linear

# Why Might Predictions be Wrong?

- True non-determinism
  - Flip a biased coin
  - $p(\text{heads}) = \theta$
  - Estimate  $\theta$
  - If  $\theta > 0.5$  predict 'heads', else 'tails'

Lots of ML research on problems like this:

- Learn a model
- Do the best you can in expectation

# Why Might Predictions be Wrong?

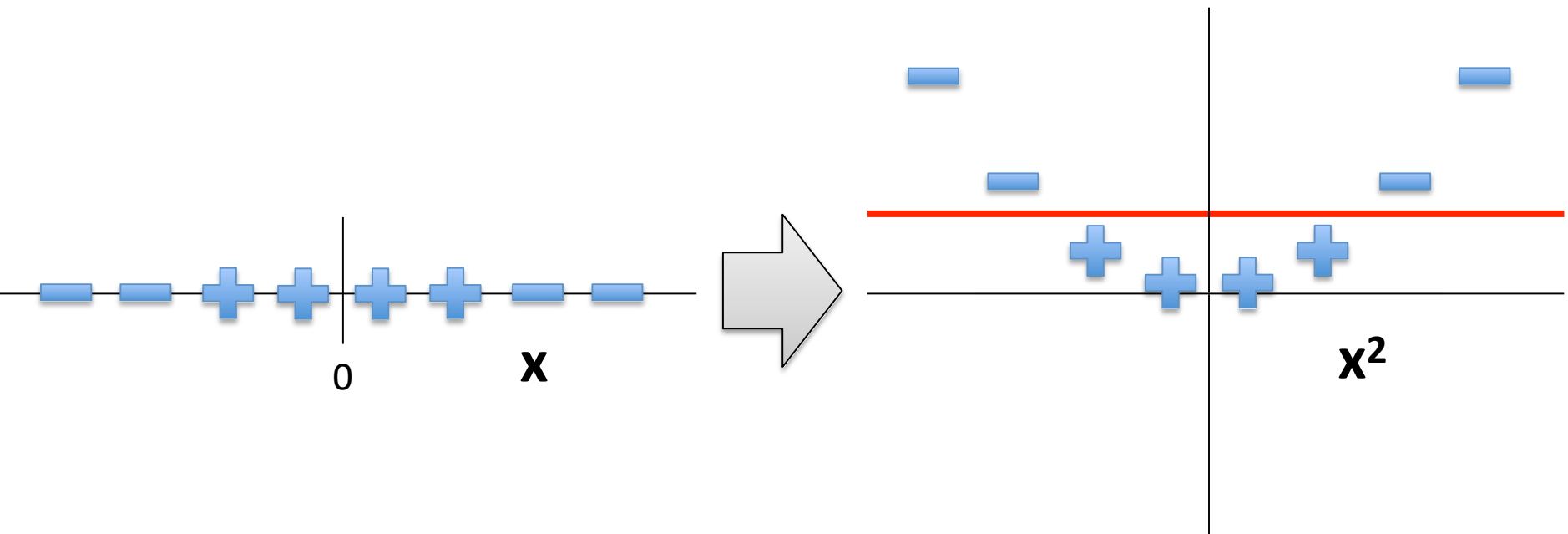
- Partial observability
  - Something needed to predict  $y$  is missing from observation  $\mathbf{x}$
  - ~~M~~ parity problem
    - $\mathbf{x}$  contains  $M$  bits (hard PO)
    - $\mathbf{x}$  contains  $N$  bits but learner ignores some of them (soft PO)
- Noise in the observation  $\mathbf{x}$ 
  - Measurement error
  - Instrument limitations

# Why Might Predictions be Wrong?

- True non-determinism
- Partial observability
  - hard, soW
- Representational bias
- Algorithmic bias
- Bounded resources

# Representational Bias

- Having the right features ( $\mathbf{x}$ ) is crucial



# Support Vector Machines

Doing *Really* Well with Linear  
Decision Surfaces

# Strengths of SVMs

- Good generalization
  - in theory
  - in practice
- Works well with few training instances
- Find globally best model
- Efficient algorithms
- Amenable to the kernel trick



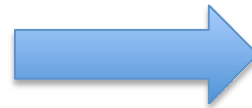
# Minor Notation Change

To better match notation used in SVMs  
...and to make matrix formulas simpler

We will drop using superscripts for the  $i^{\text{th}}$  instance

$i^{\text{th}}$  instance

$\mathbf{x}^{(i)}$

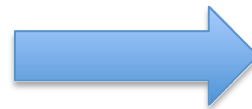


$\mathbf{x}_i$

**Bold** denotes  
vector

$i^{\text{th}}$  instance label

$y^{(i)}$



$y_i$

**Non-bold**  
denotes scalar

$j^{\text{th}}$  feature of  $i^{\text{th}}$  instance

$x_j^{(i)}$



$x_{ij}$

# Linear Separators

- Training instances

$$\mathbf{x} \in \mathbb{R}^{d+1}, x_0 = 1$$

$$y \in \{-1, 1\}$$

- Model parameters

$$\mathbf{w} \in \mathbb{R}^{d+1}$$

- Hyperplane

$$\mathbf{w}^T \mathbf{x} = h(\mathbf{w}, \mathbf{x}) = 0$$

- Decision function

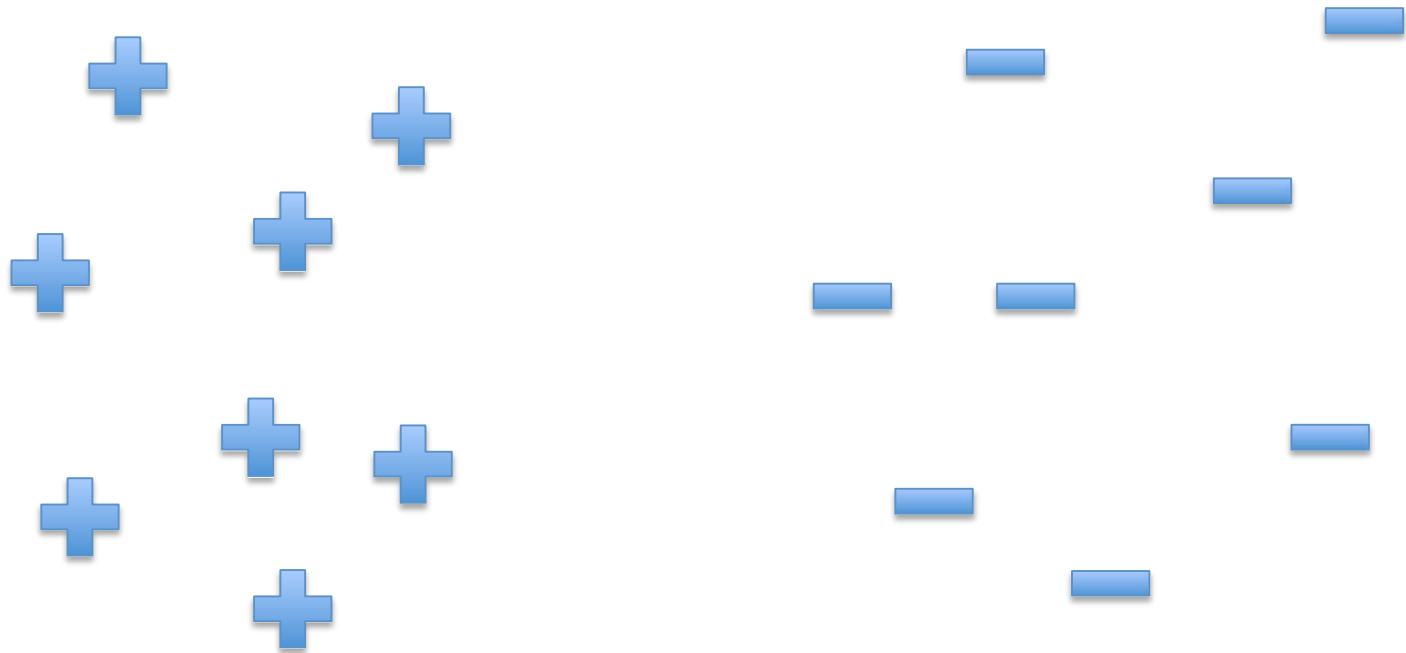
$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(h(\mathbf{w}, \mathbf{x}))$$

Recall:

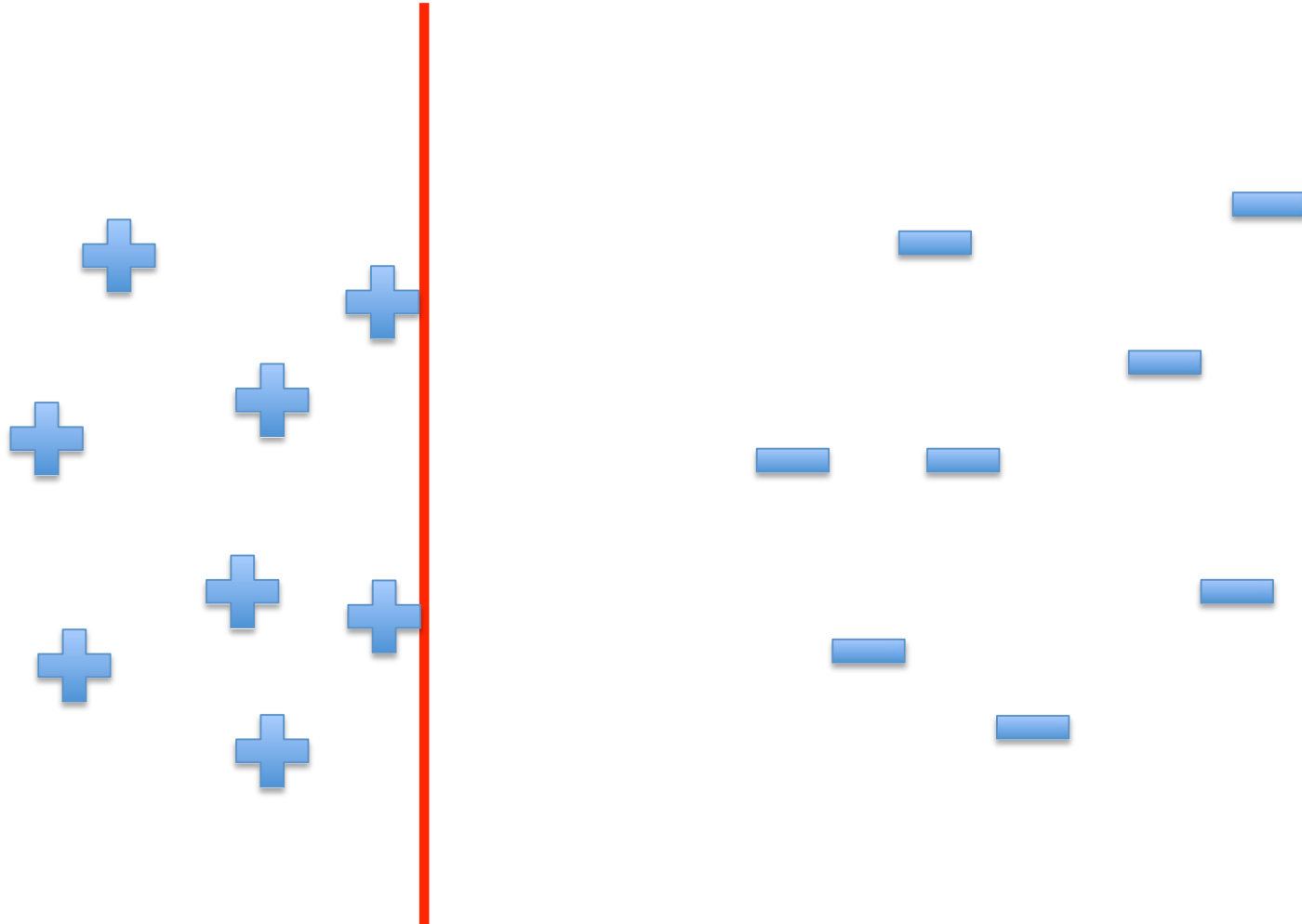
Inner (dot) product:

$$\begin{aligned} h(\mathbf{u}, \mathbf{v}) &= \mathbf{u} \cdot \mathbf{v} = \\ \mathbf{u}^T \mathbf{v} &= \sum_i u_i v_i \end{aligned}$$

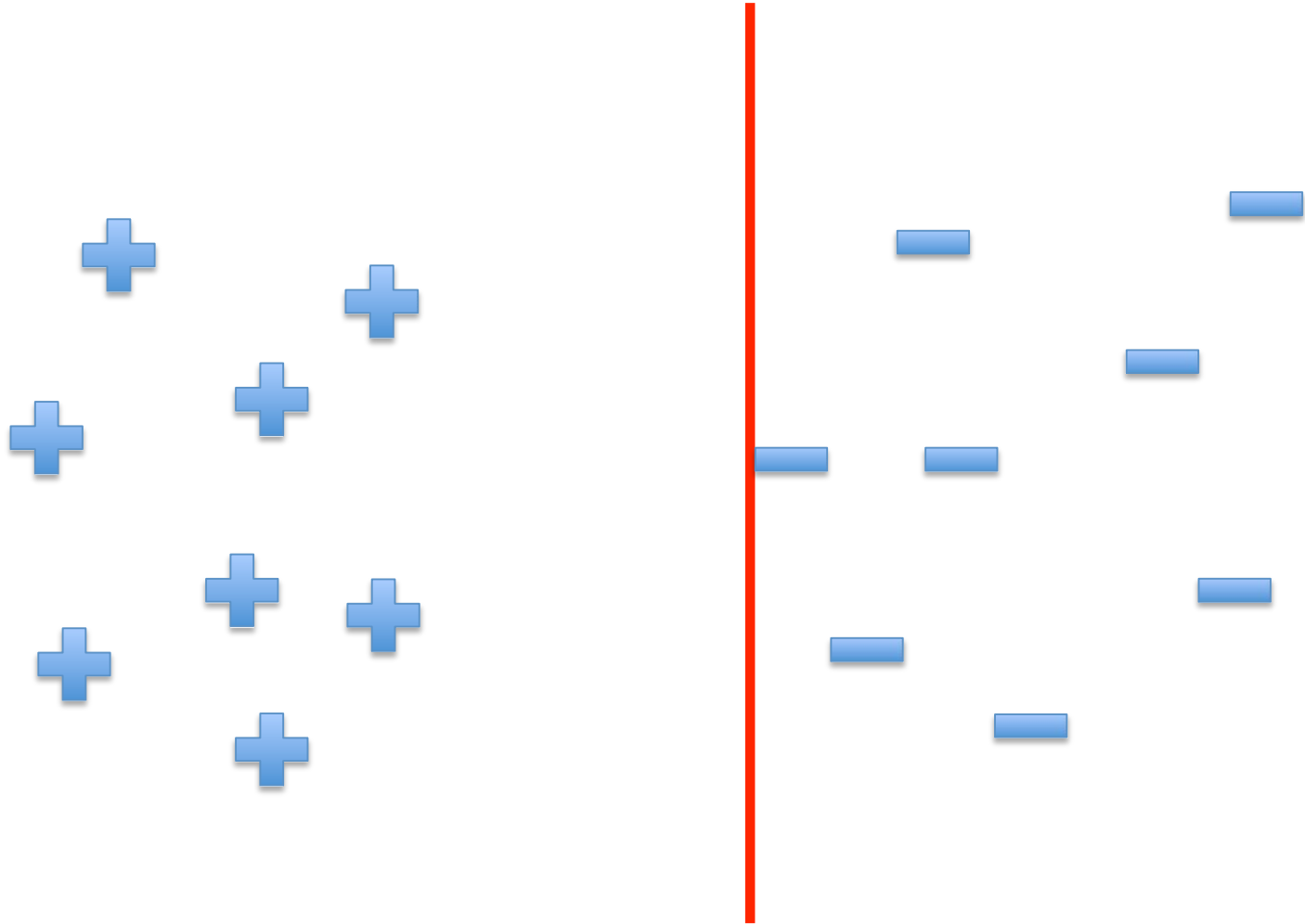
# Intuitions



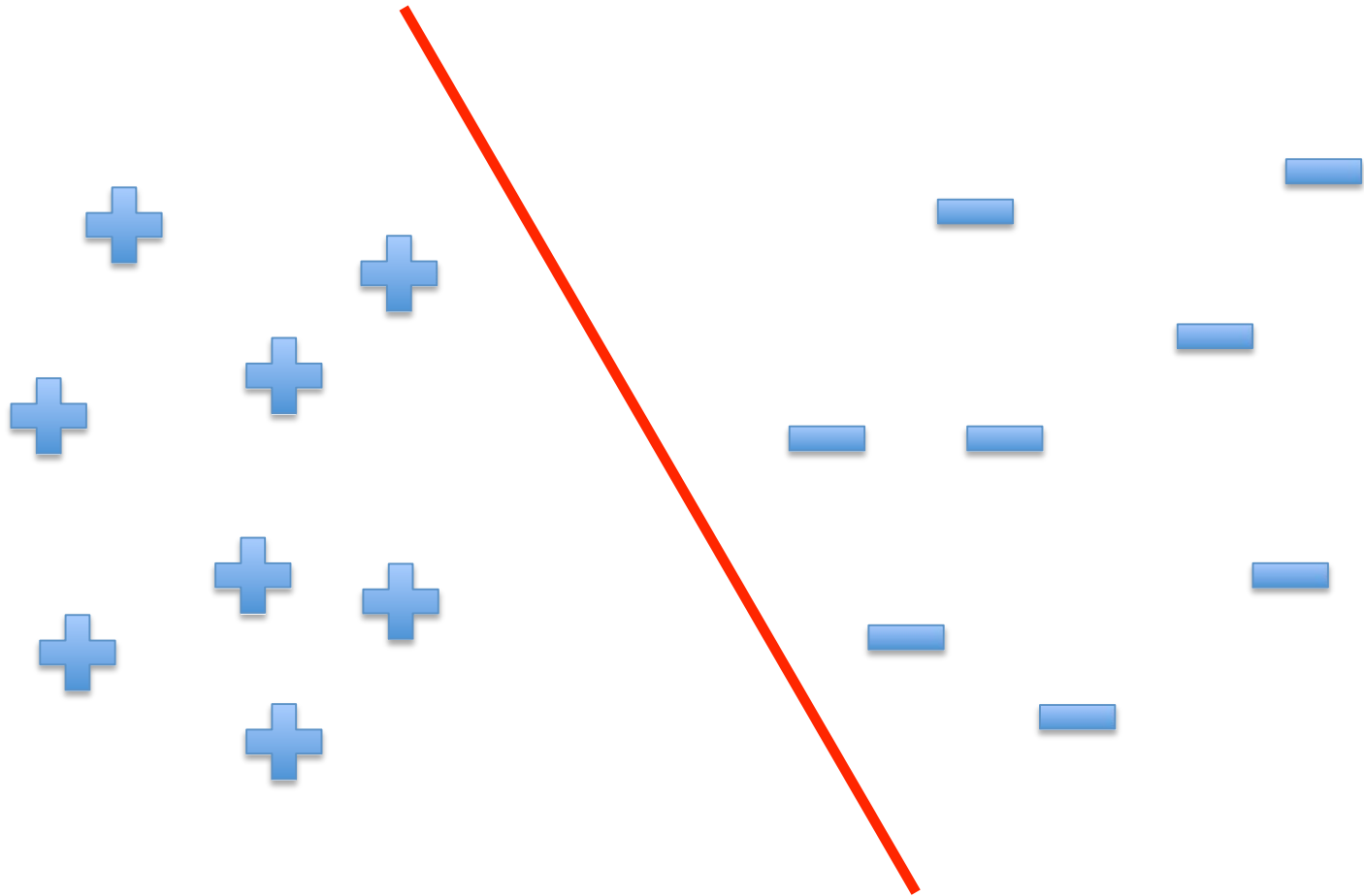
# Intuitions



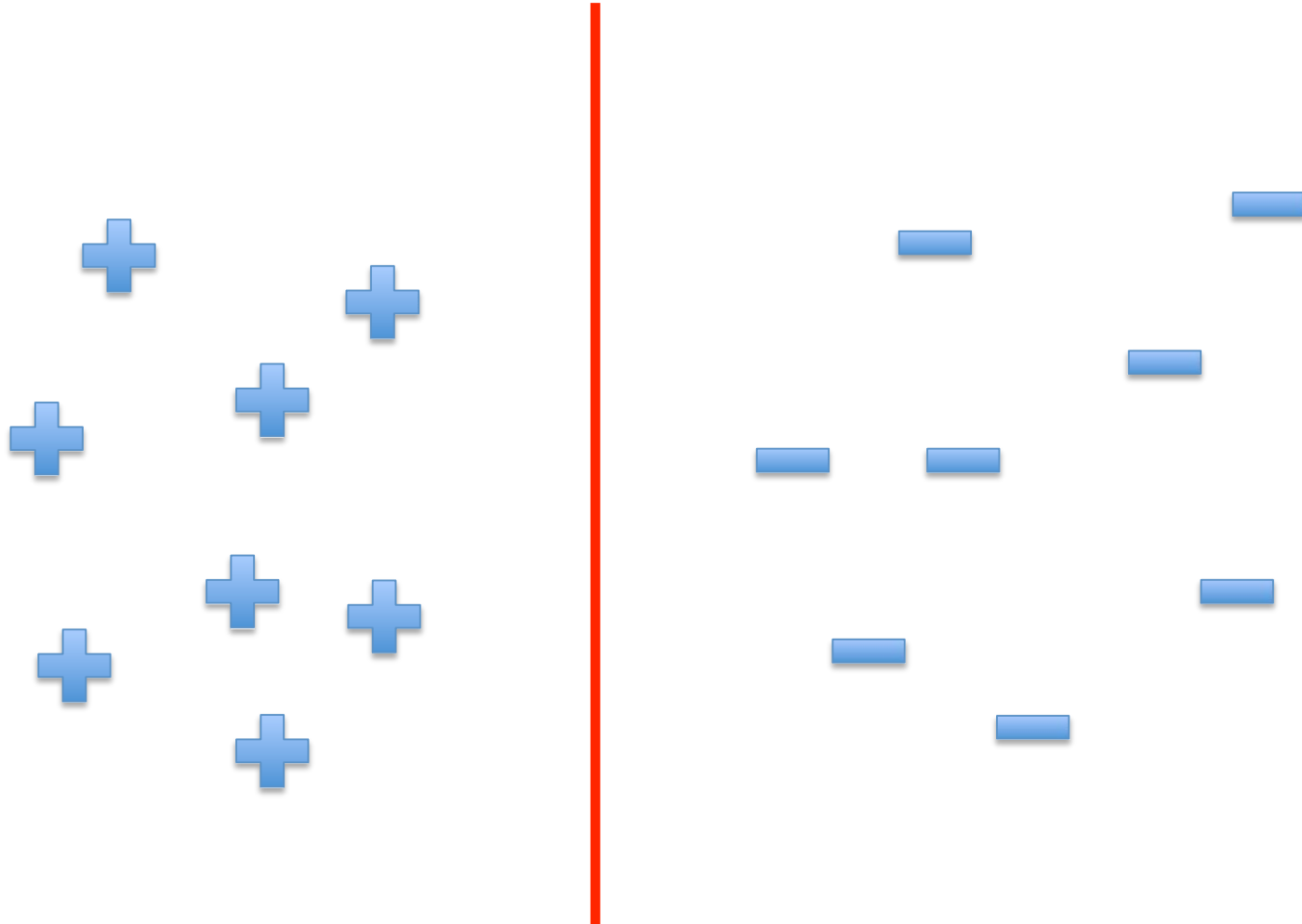
# Intuitions



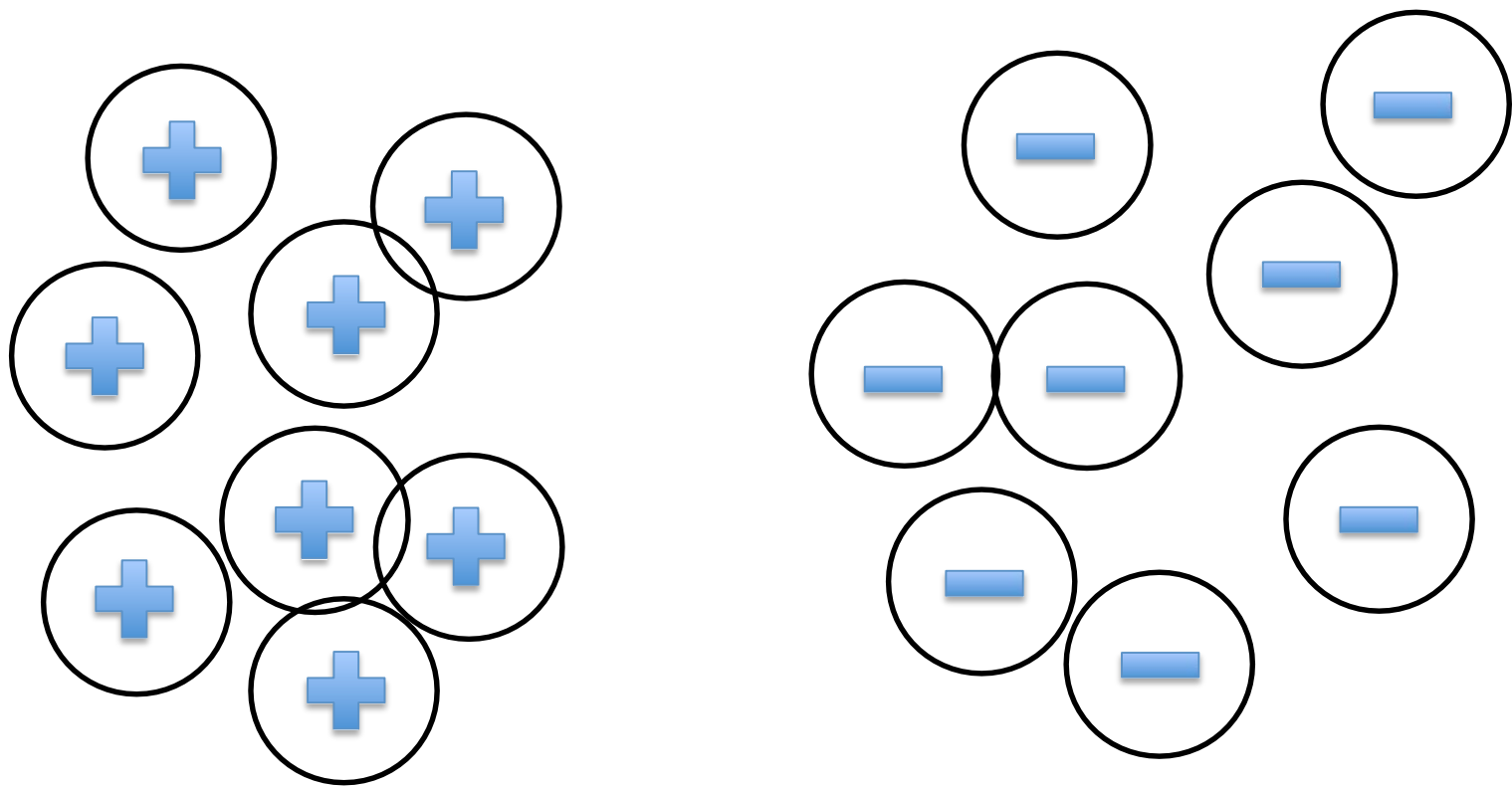
# Intuitions



# A “Good” Separator

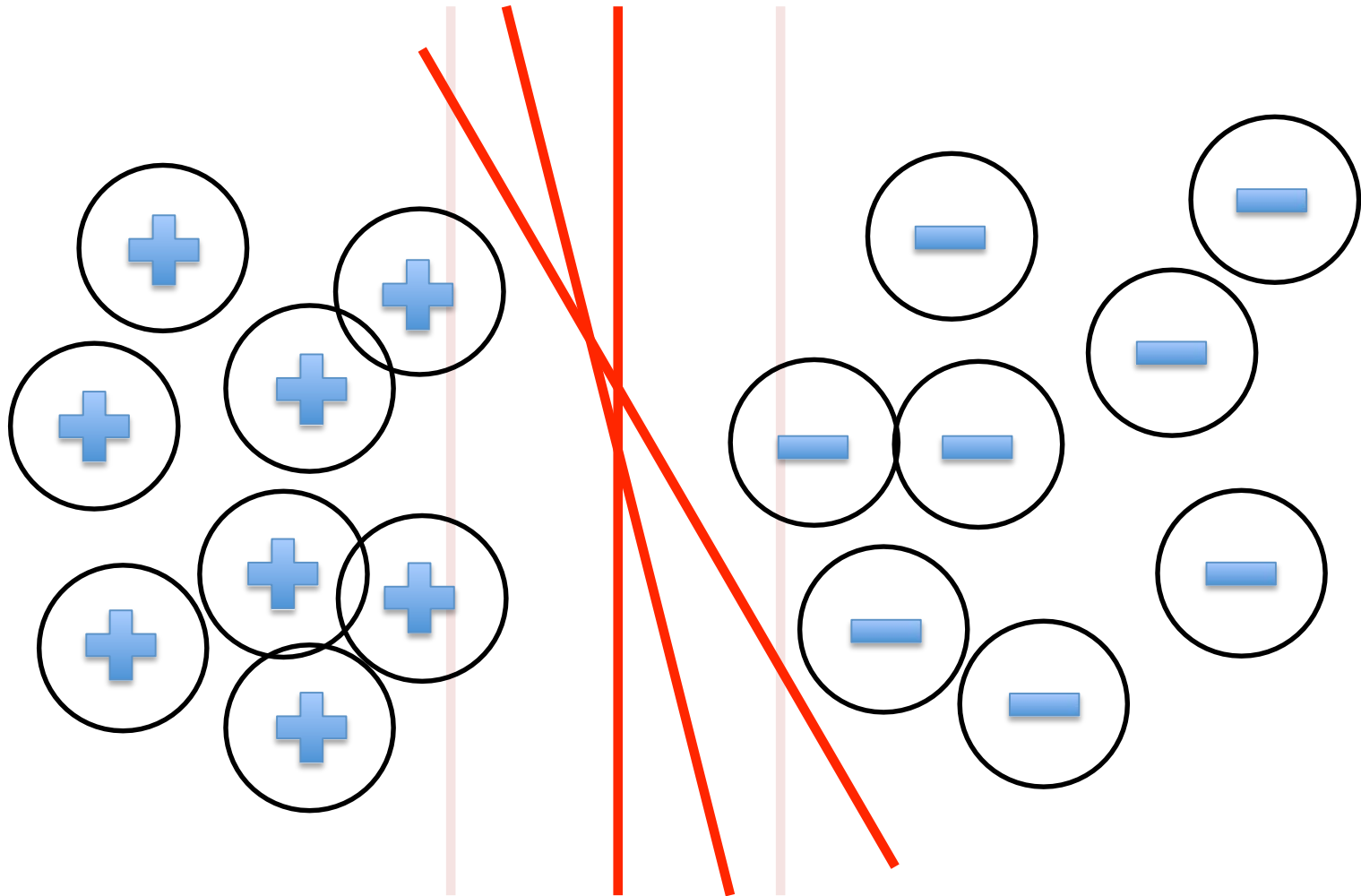


# Noise in the Observations

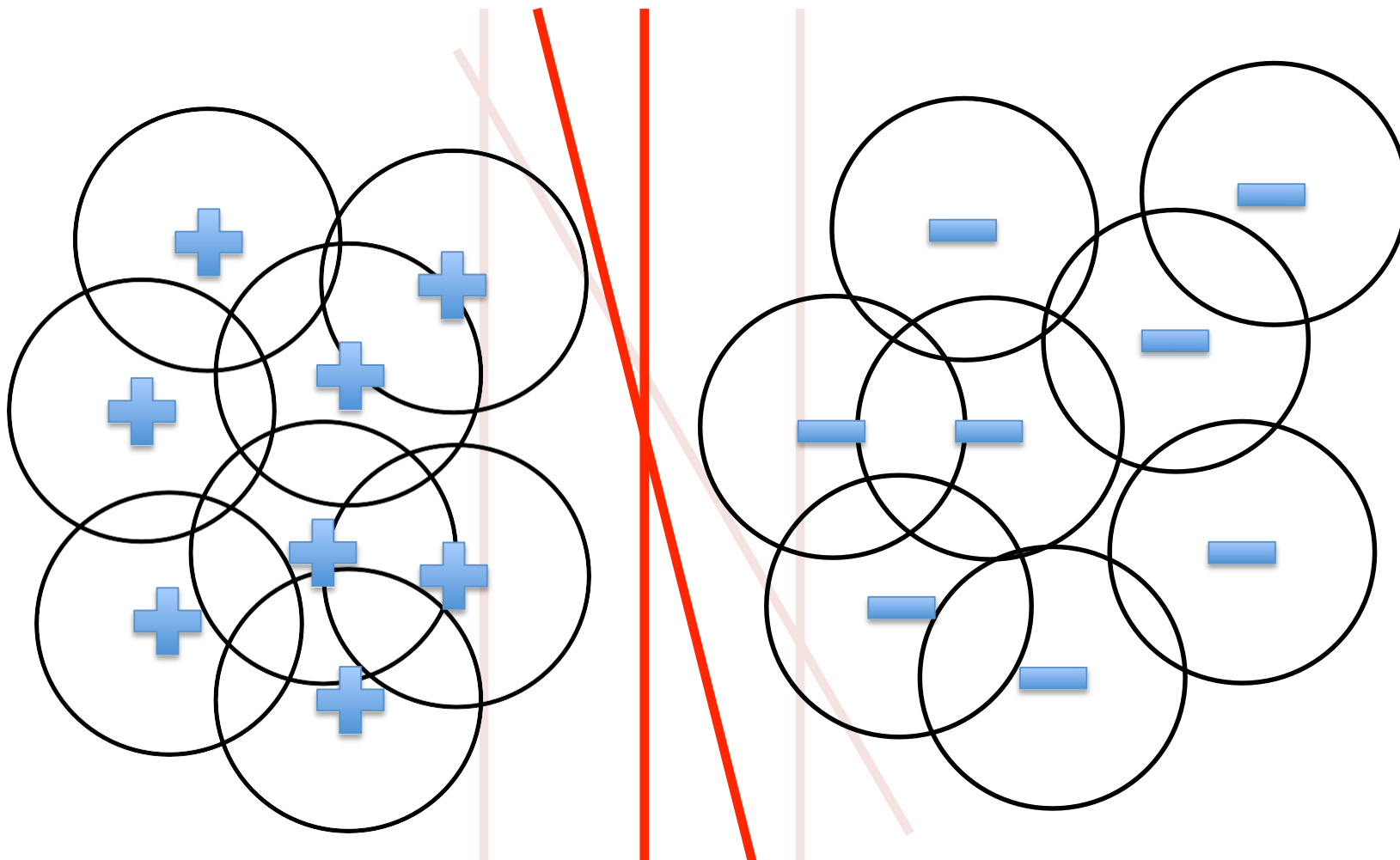




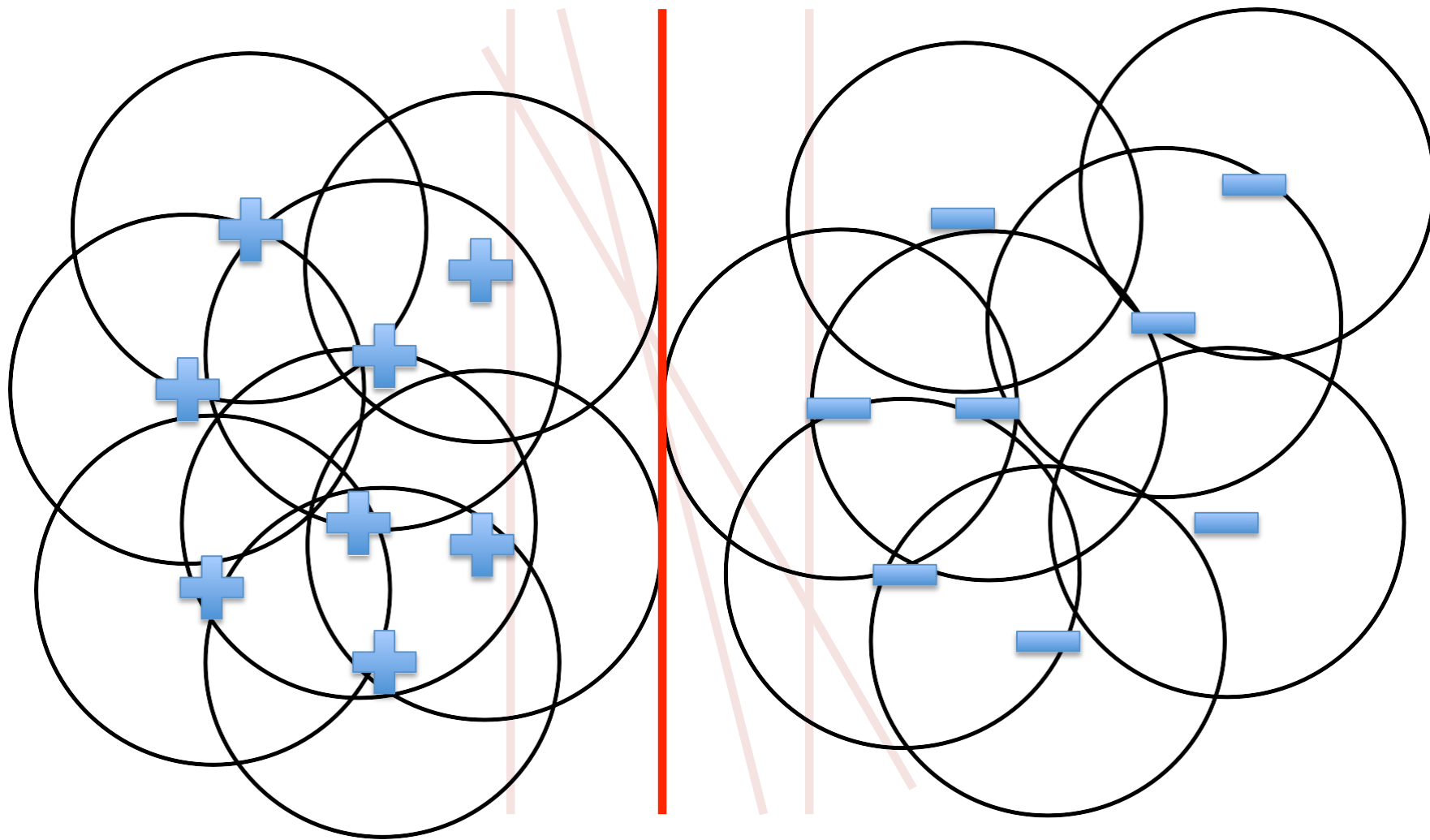
# Ruling Out Some Separators



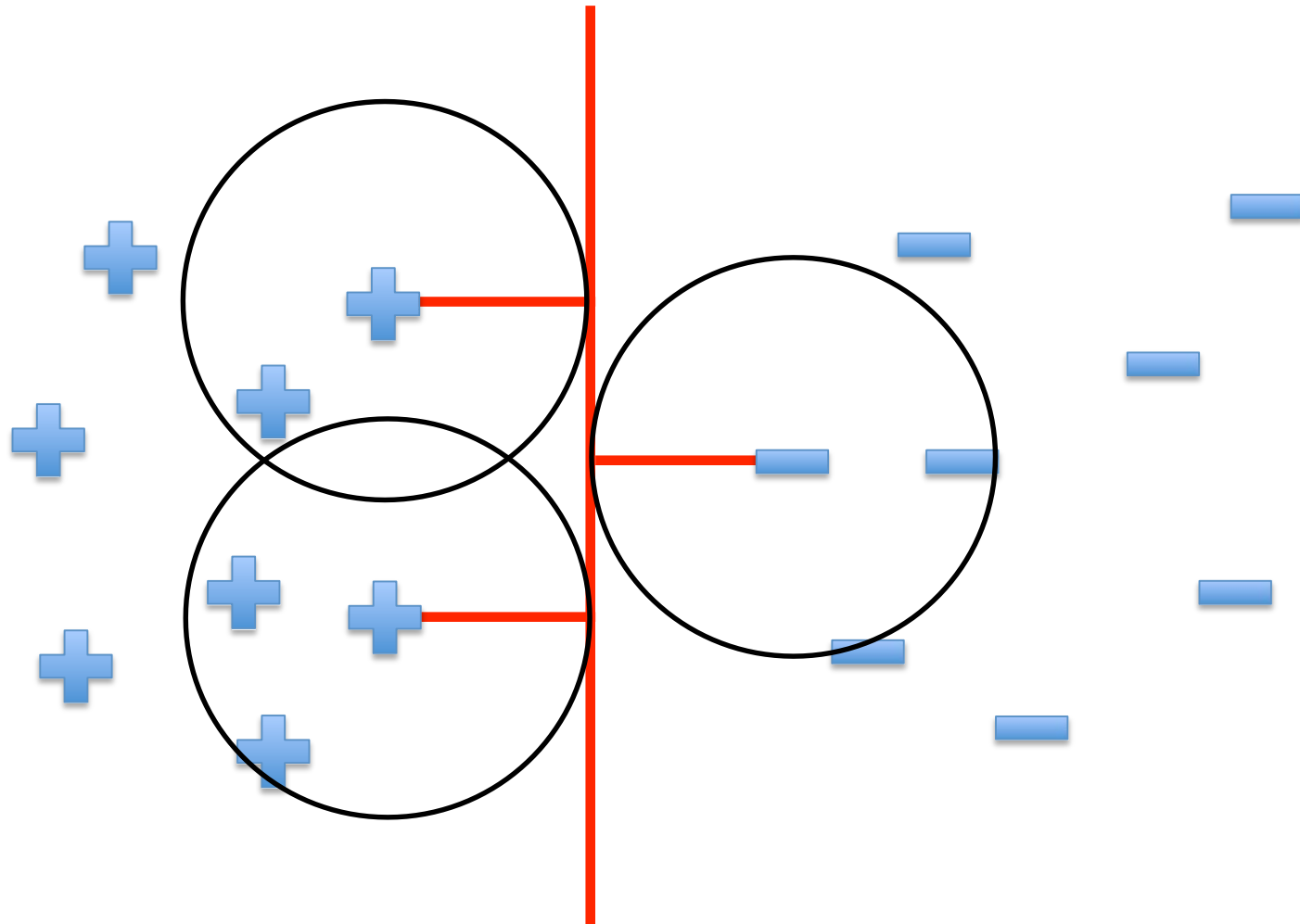
# Lots of Noise



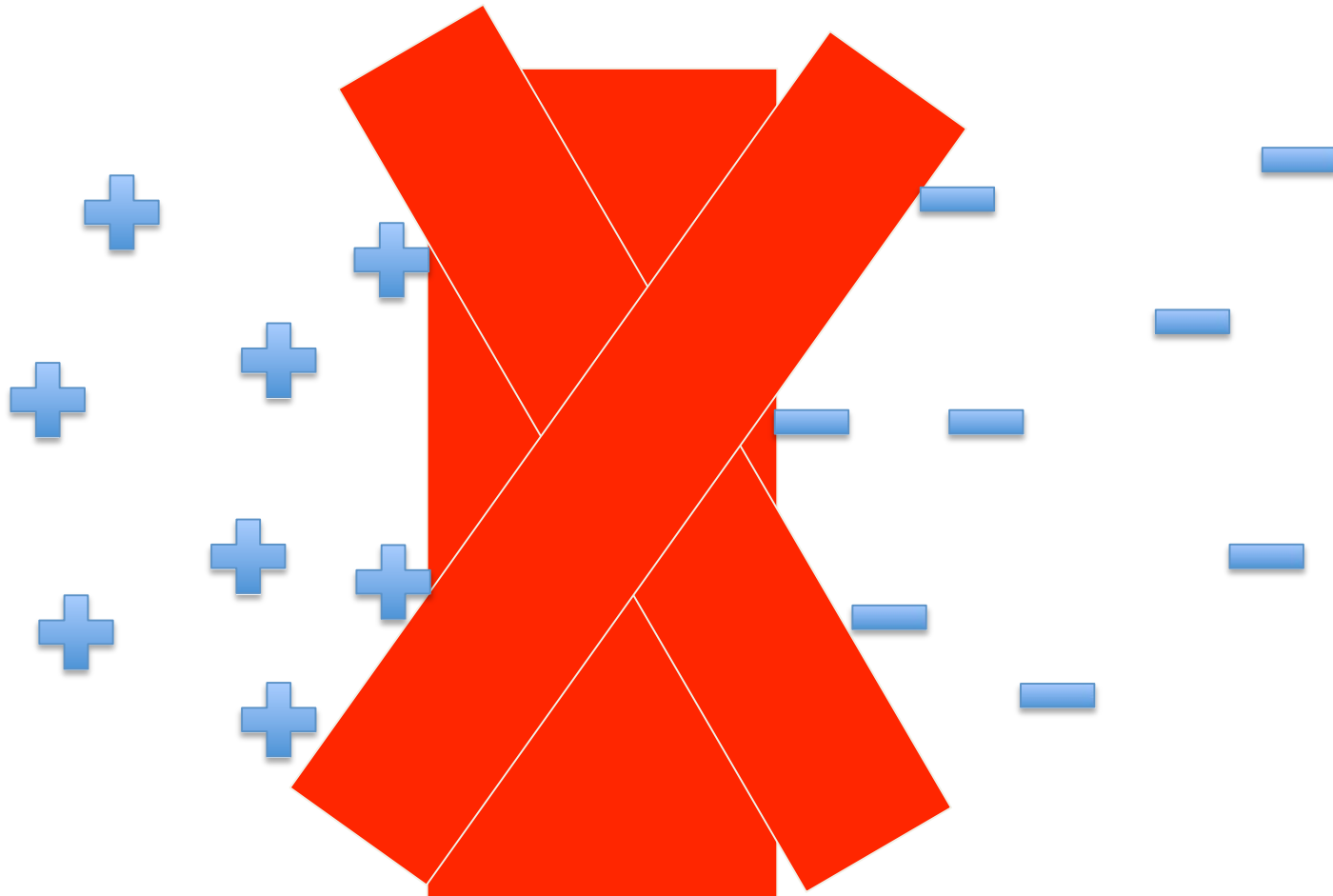
# Only One Separator Remains



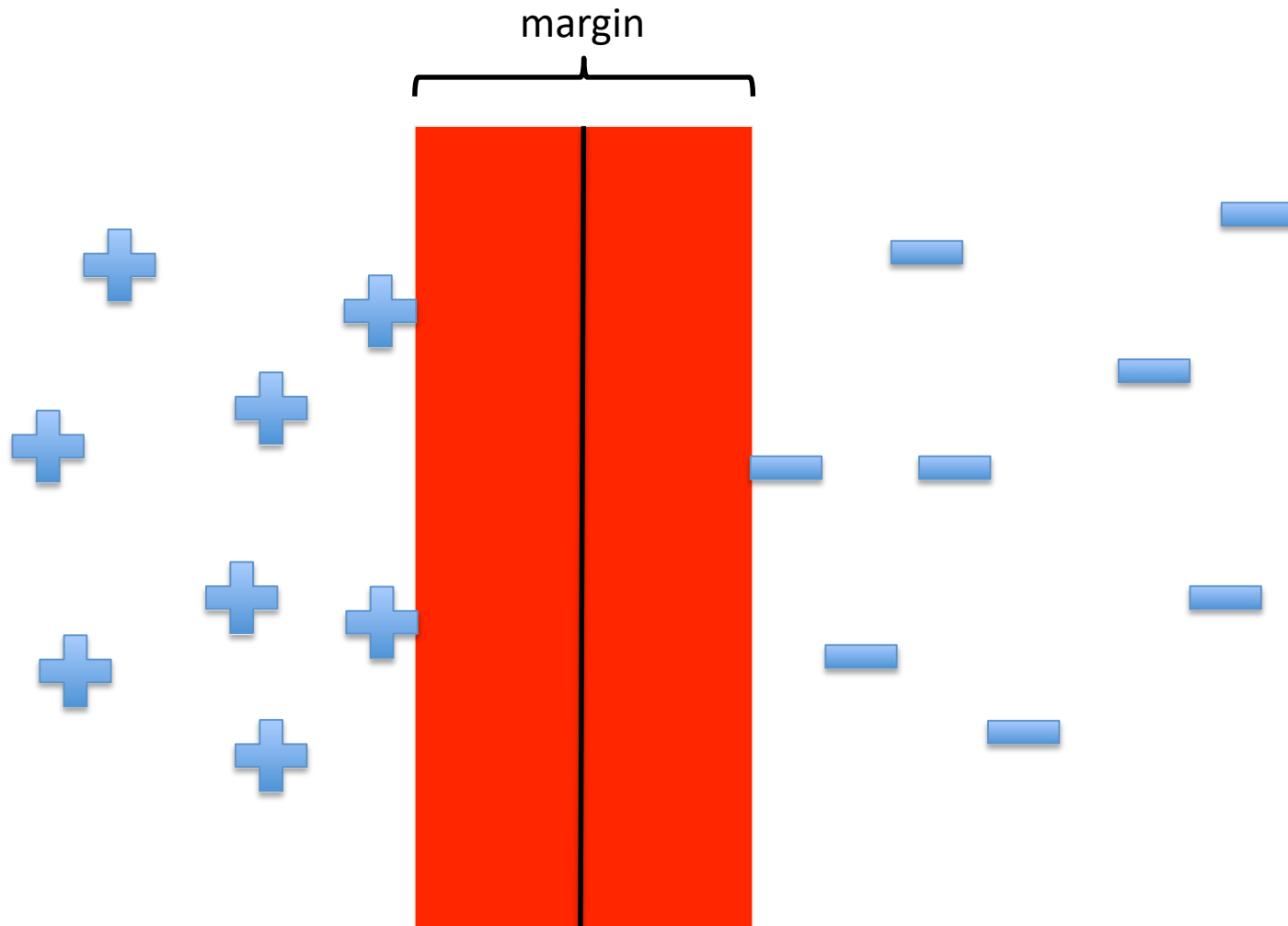
# Maximizing the Margin



# 'Fa' Separators



# ' $\alpha$ ' Separators



# Why Maximize Margin

Increasing margin reduces *capacity*

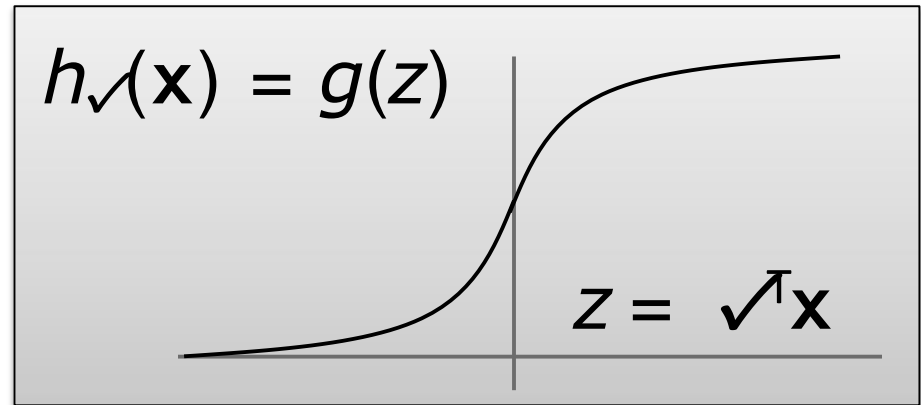
- i.e., fewer possible models

Lesson from Learning Theory:

- If the following holds:
  - $H$  is sufficiently constrained in size
  - and/or the size of the training data set  $n$  is large,then low training error is likely to be evidence of low generalization error

# Alternative View of Logistic Regression

$$h_{\checkmark}(\mathbf{x}) = \frac{1}{1 + e^{-\checkmark^T \mathbf{x}}}$$



If  $y = 1$ , we want  $h_{\checkmark}(\mathbf{x}) \hat{=} 1$ ,  $\checkmark^T \mathbf{x} \gg 0$

If  $y = 0$ , we want  $h_{\checkmark}(\mathbf{x}) \hat{=} 0$ ,  $\checkmark^T \mathbf{x} \ll 0$

$$J(\checkmark) = - \sum_{i=1}^N [ \underbrace{y_i \log h_{\checkmark}(\mathbf{x}_i)}_{\text{cost}_1(\checkmark^T \mathbf{x}_i)} + \underbrace{(1 - y_i) \log (1 - h_{\checkmark}(\mathbf{x}_i))}_{\text{cost}_0(\checkmark^T \mathbf{x}_i)} ]$$

$\min_{\checkmark} J(\checkmark)$

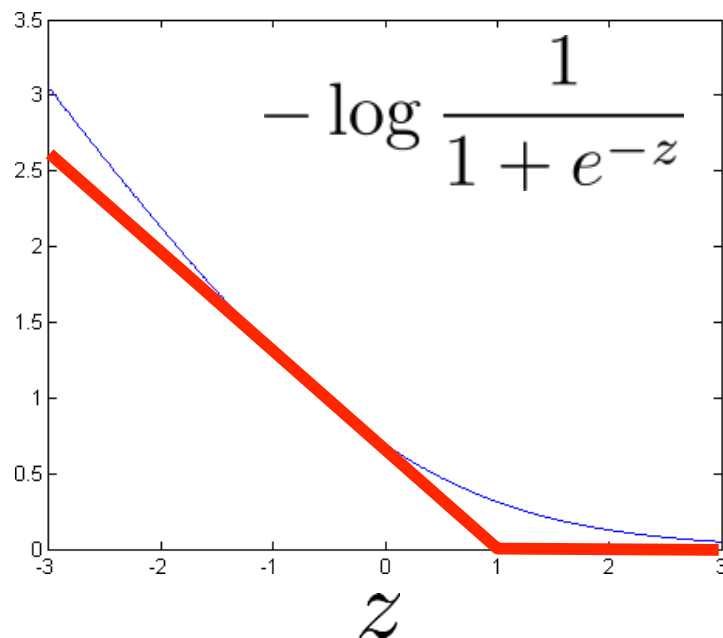


# Alternate View of Logistic Regression

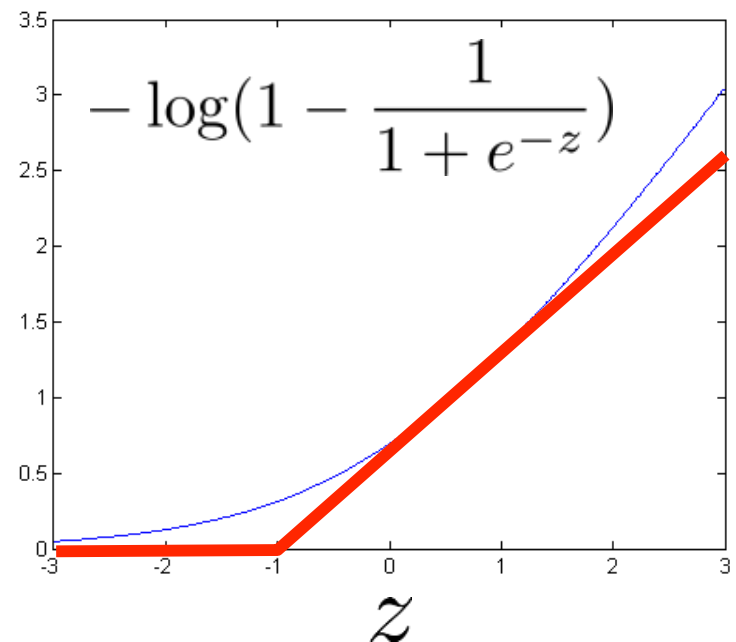
Cost of example:  $-y_i \log h_{\sqrt{}}(\mathbf{x}_i) - (1 - y_i) \log (1 - h_{\sqrt{}}(\mathbf{x}_i))$

$$h_{\sqrt{}}(\mathbf{x}) = \frac{1}{1 + e^{-\sqrt{}}^T \mathbf{x}} \quad z = \sqrt{}}^T \mathbf{x}$$

If  $y = 1$  (want  $\sqrt{}}^T \mathbf{x} \geq 0$ ):



If  $y = 0$  (want  $\sqrt{}}^T \mathbf{x} \leq 0$ ):



# Logistic Regression to SVMs

Logistic Regression:

$$\min_{\checkmark} \frac{1}{n} \sum_{i=1}^n [y_i \log h_{\checkmark}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\checkmark}(\mathbf{x}_i))] + \frac{1}{2} \sum_{j=1}^d \checkmark_j^2$$

Support Vector Machines:

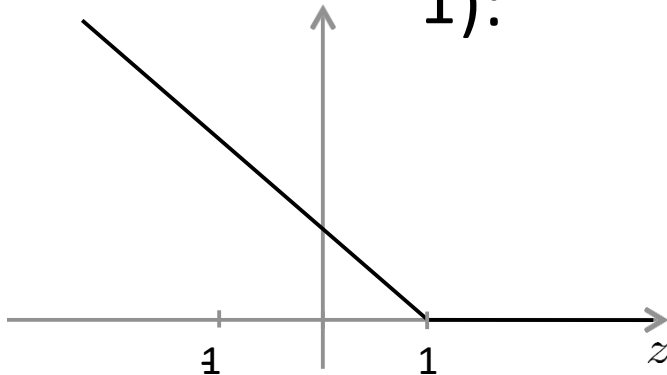
$$\min_{\checkmark} C \frac{1}{n} \sum_{i=1}^n [y_i \text{cost}_1(\checkmark^\top \mathbf{x}_i) + (1 - y_i) \text{cost}_0(\checkmark^\top \mathbf{x}_i)] + \frac{1}{2} \sum_{j=1}^d \checkmark_j^2$$

You can think of  $C$  as similar to  $\frac{1}{n}$

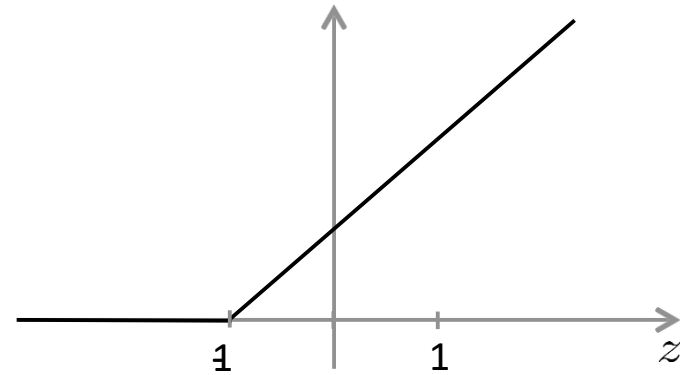
# Support Vector Machine

$$\min_{\mathbf{w}} C \sum_{i=1}^n [y_i \text{cost}_1(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \text{cost}_0(\mathbf{w}^T \mathbf{x}_i)] + \frac{1}{2} \sum_{j=1}^d \mathbf{x}_j^2$$

If  $y = 1$  (want  $\mathbf{w}^T \mathbf{x} \leq -1$ ):



If  $y = 0$  (want  $\mathbf{w}^T \mathbf{x} \leq -1$ ):



$$\text{hinge}(h(\mathbf{x})) = \max(0, 1 - y \cdot h(\mathbf{x}))$$

# Support Vector Machine

$$\min_{\sqrt{\cdot}} C \sum_{i=1}^n [y_i \text{cost}_1(\sqrt{\cdot} \mathbf{x}_i) + (1 - y_i) \text{cost}_0(\sqrt{\cdot} \mathbf{x}_i)] + \frac{1}{2} \sum_{j=1}^d X^d \sqrt{j}^2$$

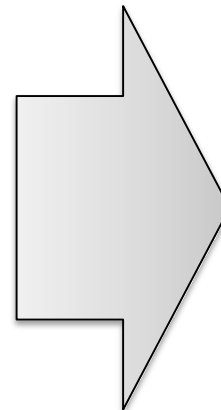
$y = 1 / 0$

with  $C = 1$

$y = +1 / -1$

$$\min_{\sqrt{\cdot}} \frac{1}{2} \sum_{j=1}^d X^d \sqrt{j}^2$$

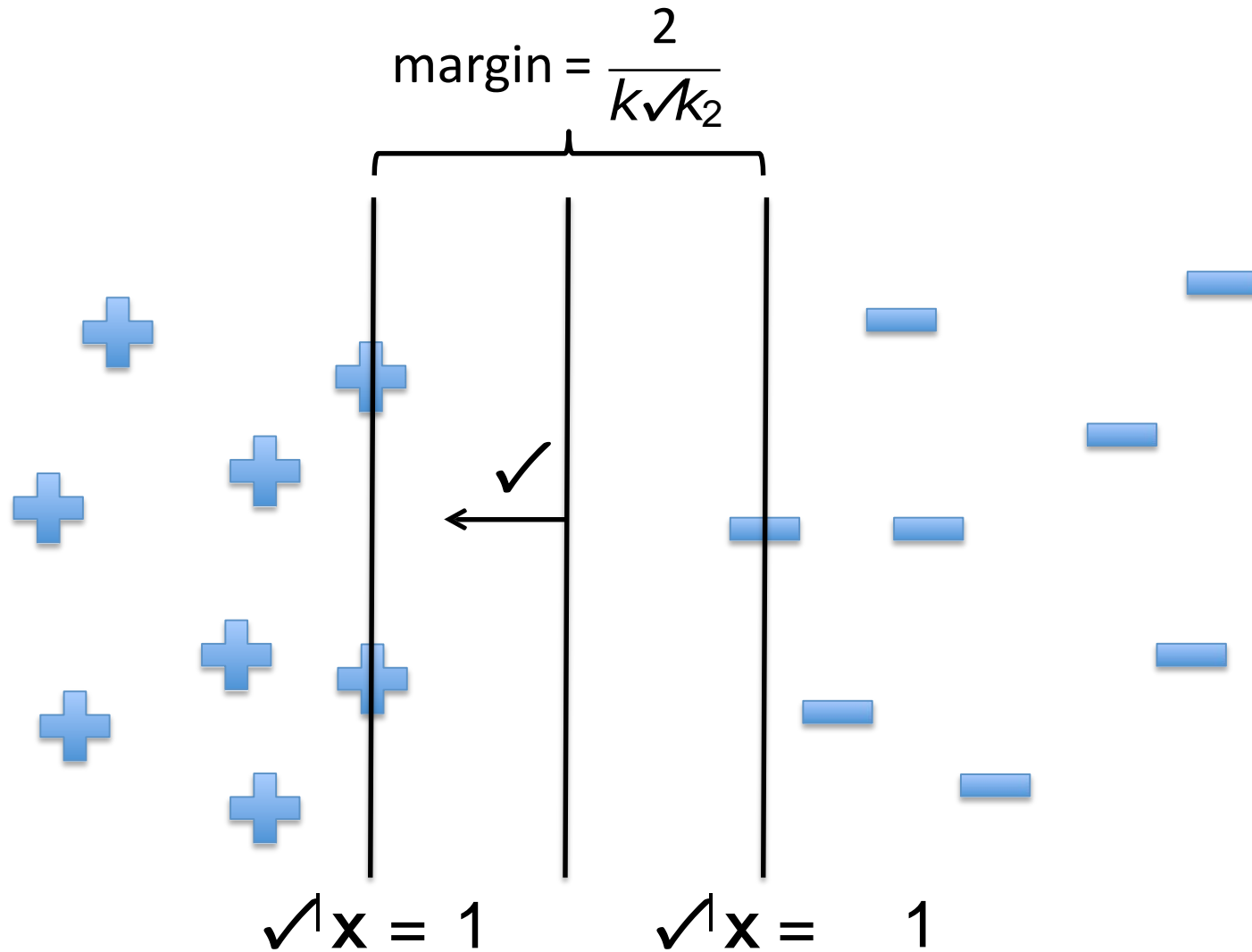
s.t.  $\sqrt{\cdot} \mathbf{x}_i \leq 1$  if  $y_i = 1$   
 $\sqrt{\cdot} \mathbf{x}_i \leq -1$  if  $y_i = -1$



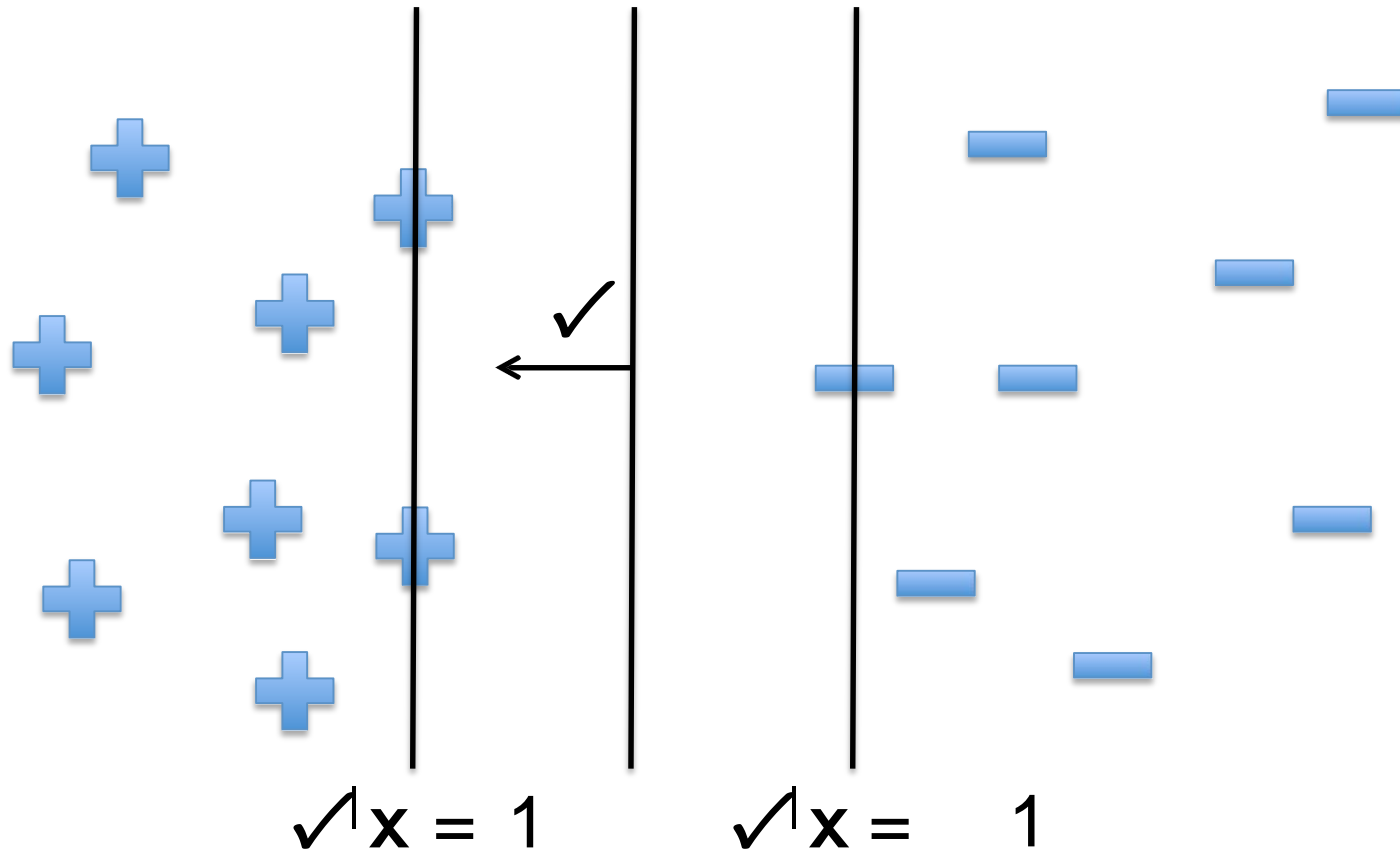
$$\min_{\sqrt{\cdot}} \frac{1}{2} \sum_{j=1}^d X^d \sqrt{j}^2$$

s.t.  $y_i(\sqrt{\cdot} \mathbf{x}_i) \leq 1$

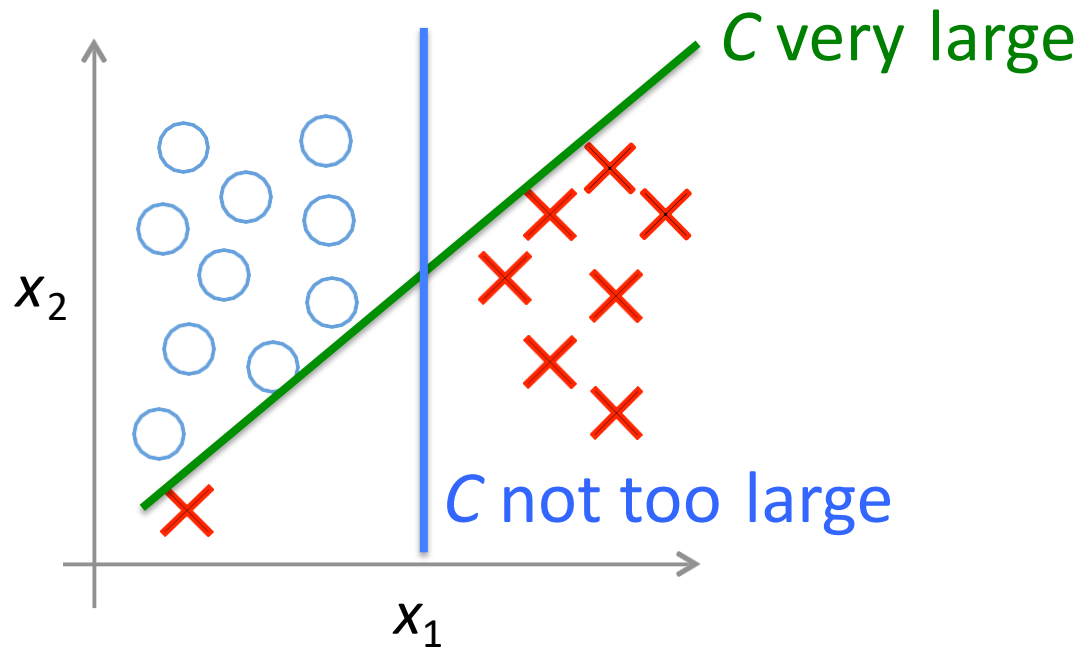
# Maximum Margin Hyperplane



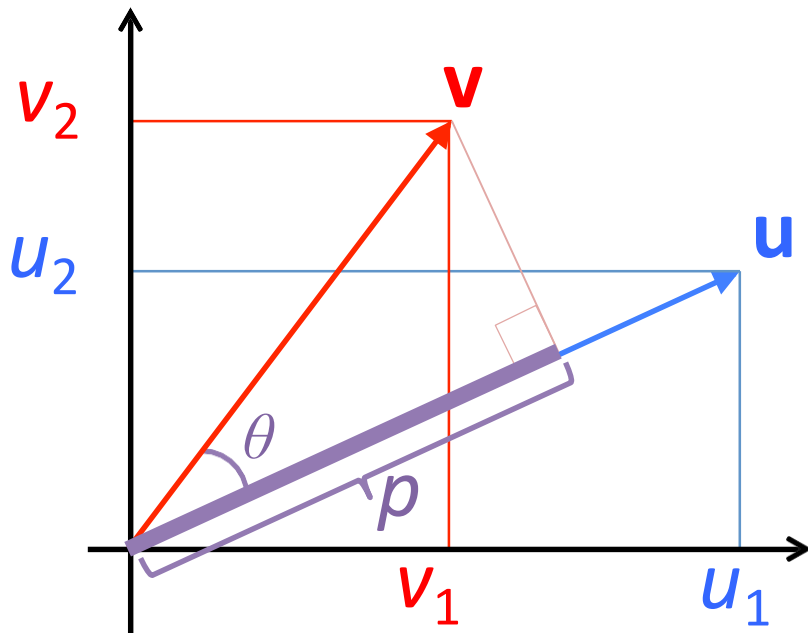
# Support Vectors



# Large Margin Classifier in Presence of Outliers



# Vector Inner Product



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

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

$$u^T v = v^T u$$

$$= u_1 v_1 + u_2 v_2$$

$$= \|u\|_2 \|v\|_2 \cos \theta$$

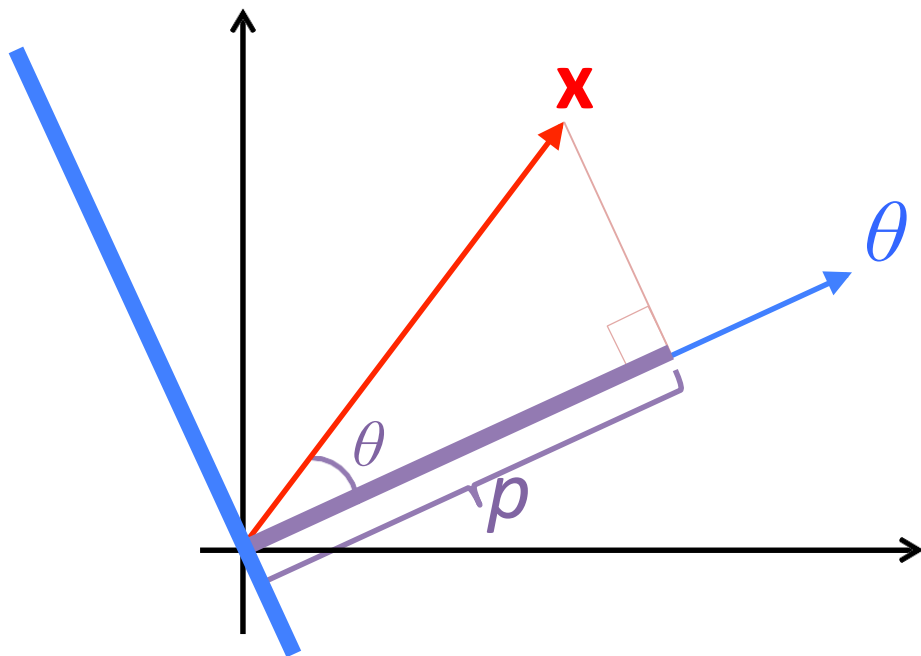
$$= p \|u\|_2 \quad \text{where } p = \|v\|_2 \cos \theta$$



# Understanding the Hyperplane

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{j=1}^d x_j^2 \\ \text{s.t.} \quad & \mathbf{x}_i \leq 1 \quad \text{if } y_i = 1 \\ & \mathbf{x}_i \leq -1 \quad \text{if } y_i = -1 \end{aligned}$$

Assume  $\theta_0 = 0$  so that the hyperplane is centered at the origin, and that  $d = 2$



$$\begin{aligned} \|\mathbf{x}\| &= \|\mathbf{x}\|_2 \cos \theta \\ &= p \|\mathbf{x}\|_2 \end{aligned}$$

# Maximizing the Margin

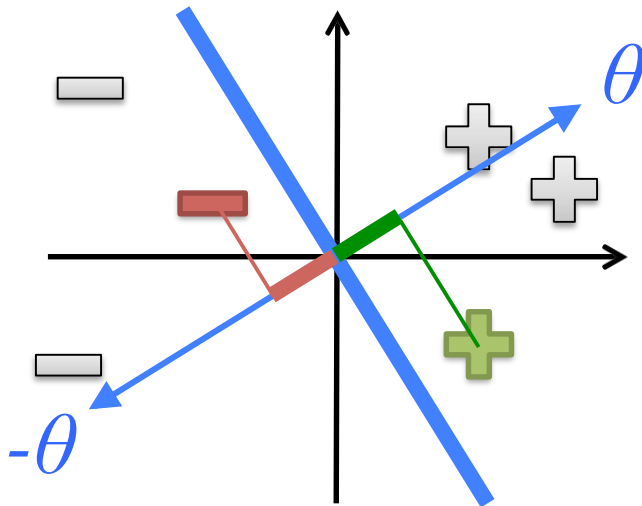
$$\min \frac{1}{2} \sum_{j=1}^d \sum_i \sqrt{x_j^2}$$

$$\text{s.t. } \sqrt{\|\mathbf{x}_i\|} \leq 1 \quad \text{if } y_i = 1$$

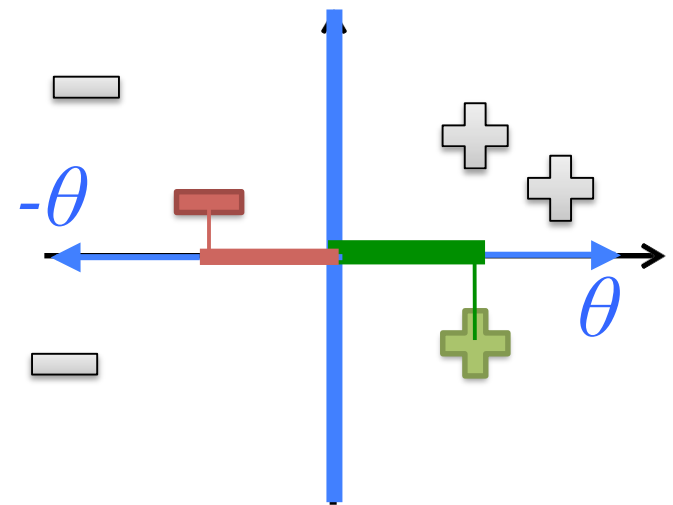
$$\sqrt{\|\mathbf{x}_i\|} \leq -1 \quad \text{if } y_i = -1$$

Assume  $\theta_0 = 0$  so that the hyperplane is centered at the origin, and that  $d=2$

Let  $p_i$  be the projection of  $\mathbf{x}_i$  onto the vector  $\theta$



Since  $p$  is small, therefore  $\sqrt{k_2}$  must be large to have  $p\sqrt{k_2} \leq 1$  (or  $\leq 1$ )

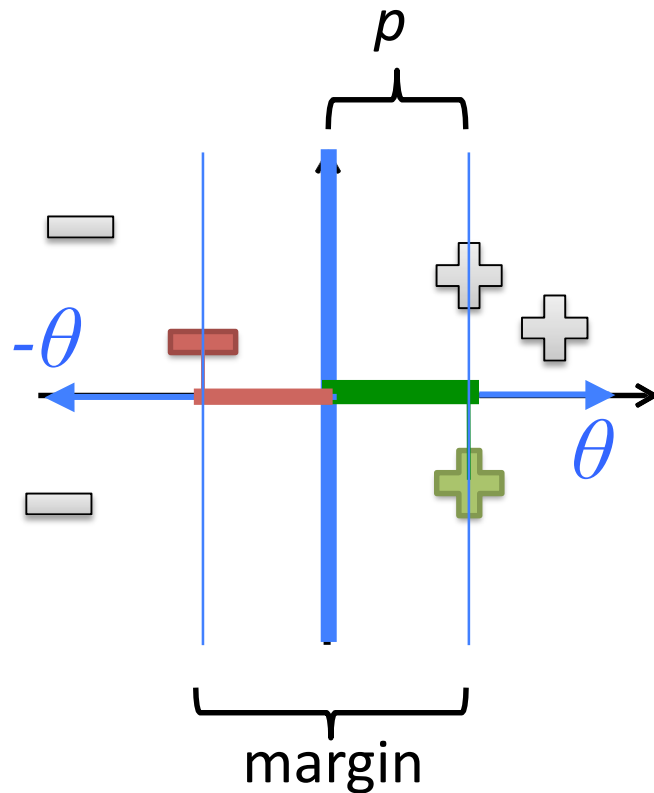


Since  $p$  is larger,  $\sqrt{k_2}$  can be smaller in order to have  $p\sqrt{k_2} \leq 1$  (or  $\leq 1$ )

# Size of the Margin

For the support vectors, we have  $p k \sqrt{k_2} = \pm 1$

- $p$  is the length of the projection of the SVs onto  $\theta$



Therefore,

$$p = \frac{1}{k \sqrt{k_2}}$$

$$\text{margin} = 2p = \frac{2}{k \sqrt{k_2}}$$

# The SVM Dual Problem

The primal SVM problem was given as

$$\min_{\gamma} \frac{1}{2} \sum_{j=1}^d \gamma_j^2$$

$$\text{s.t. } \gamma_i (\sum_{j=1}^d \gamma_j x_{ij}) \leq 1 - \delta_i$$

Can solve it more efficiently by taking the Lagrangian dual

- Duality is a common idea in optimization
- It transforms a difficult optimization problem into a simpler one
- Key idea: introduce slack variables  $\alpha_i$  for each constraint
  - $\alpha_i$  indicates how important a particular constraint is to the solution

# The SVM Dual Problem

- The Lagrangian is given by

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \sum_{j=1}^d \mathbf{w}_j^2 - \sum_{i=1}^n \alpha_i (y_i \mathbf{w}^T \mathbf{x}_i - 1)$$

s.t.  $\alpha_i \leq 0 \quad \forall i$

- We must minimize over  $\mathbf{w}$  and maximize over  $\boldsymbol{\alpha}$
- At optimal solution, partials w.r.t  $\mathbf{w}$ 's are 0

Solve by a bunch of algebra and calculus ...  
and we obtain ...

# SVM Dual Representation

$$\begin{aligned} \text{Maximize } J(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j h(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t. } \alpha_i &\leq 0 \quad \forall i \\ \sum_i \alpha_i y_i &= 0 \end{aligned}$$

The decision function is given by

$$h(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^n \alpha_i y_i h(\mathbf{x}, \mathbf{x}_i) + b \right)$$

where  $b = \frac{1}{|SV|} \sum_{i \in SV} \alpha_i y_i - \frac{1}{|SV|} \sum_{j \in SV} \alpha_j y_j h(\mathbf{x}_i, \mathbf{x}_j)$

# Understanding the Dual

$$\begin{aligned} \text{Maximize } J(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j h(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t. } &\alpha_i \leq 0 \quad \forall i \\ &\alpha_i y_i = 0 \end{aligned}$$

Balances between the weight of constraints for different classes

Constraint weights ( $\alpha_i$ 's) cannot be negative

# Understanding the Dual

$$\text{Maximize } J(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j h(x_i, x_j)$$

s.t.  $\alpha_i \geq 0 \quad \forall i$

Points with different labels increase the sum

Points with same label decrease the sum

Measures the similarity between points

Intuitively, we should be more careful around points near the margin



# Understanding the Dual

$$\begin{aligned}
 \text{Maximize } J(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j h(\mathbf{x}_i, \mathbf{x}_j) \\
 \text{s.t. } \alpha_i &\leq 0 \quad \forall i \\
 \sum_i \alpha_i y_i &= 0
 \end{aligned}$$

In the solution, either:

- $\alpha_i > 0$  and the constraint is tight ( $y_i(\sqrt{\lambda} \mathbf{x}_i) = 1$ )  
 ➤ point is a support vector
- $\alpha_i = 0$   
 ➤ point is not a support vector

# Employing the Solution

- Given the optimal solution  $\alpha^*$ , optimal weights are

$$\mathbf{w} = \sum_{i \in \text{SVs}} \alpha_i^* \mathbf{x}_i$$

- In this formulation, have *not* added  $x_0 = 1$
- Therefore, we can solve one of the SV constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 1$$

to obtain  $\theta_0$

- Or, more commonly, take the average solution over all support vectors

# What if Data Are Not Linearly Separable?

- Cannot find  $\theta$  that satisfies  $y_i(\mathbf{v}^T \mathbf{x}_i) \leq 1 - \delta_i$

- Introduce slack variables  $\xi_i$

$$y_i(\mathbf{v}^T \mathbf{x}_i) \leq 1 - \xi_i \quad \delta_i$$

- New problem:

$$\min_{\mathbf{v}} \frac{1}{2} \sum_{j=1}^d v_j^2 + C \sum_i \xi_i$$

$$\text{s.t. } y_i(\mathbf{v}^T \mathbf{x}_i) \leq 1 - \xi_i \quad \delta_i$$

# Strengths of SVMs

- Good generalization in theory
- Good generalization in practice
- Work well with few training instances
- Find globally best model
- Efficient algorithms
- Amenable to the kernel trick ...

# What if Surface is Non-Linear?

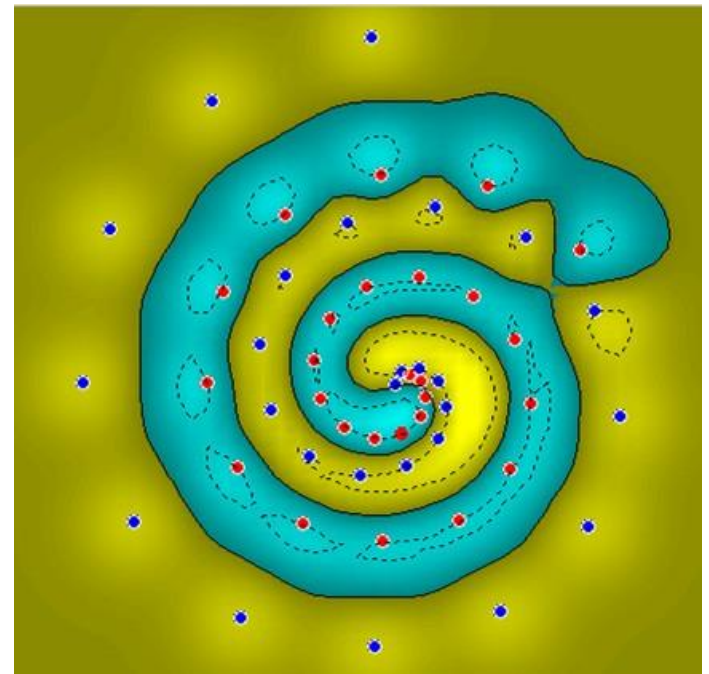
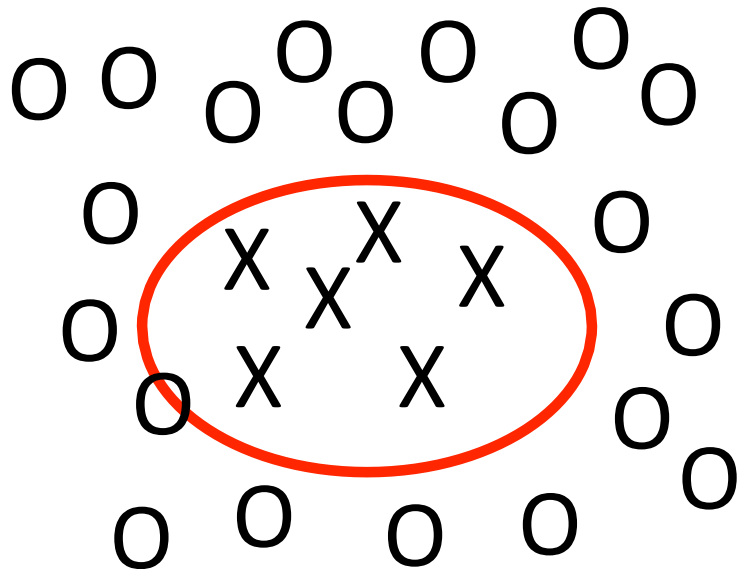
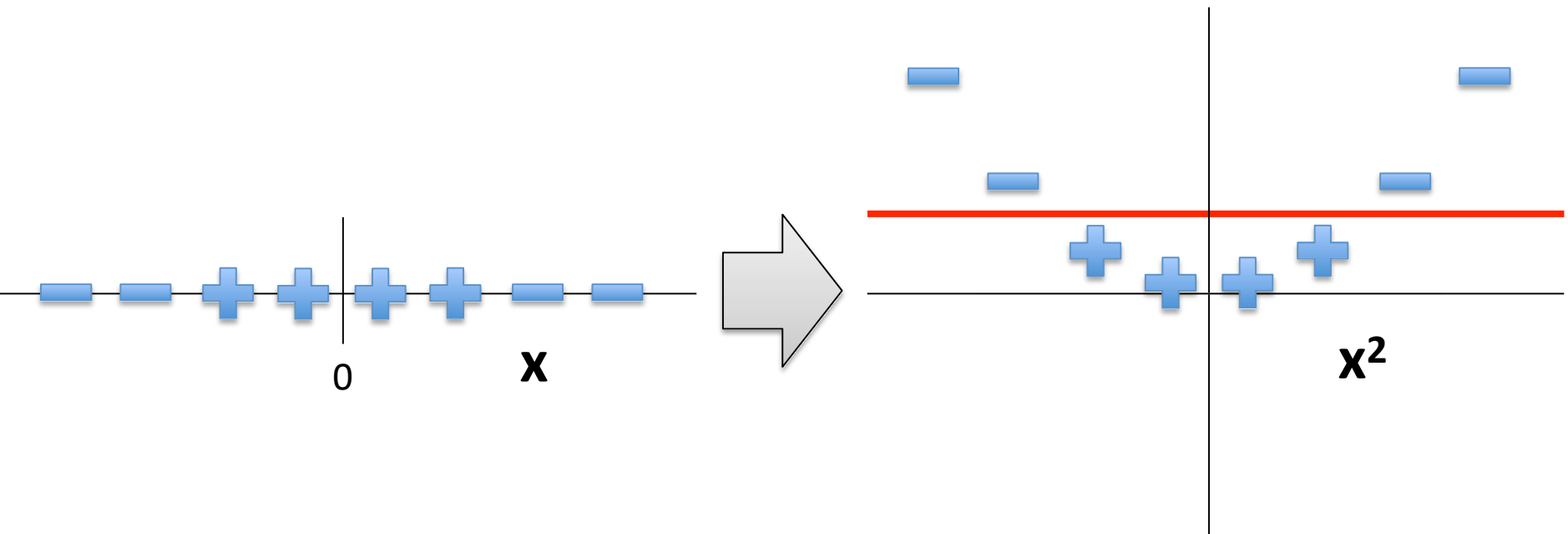


Image from <http://www.atrandomresearch.com/iclass/>

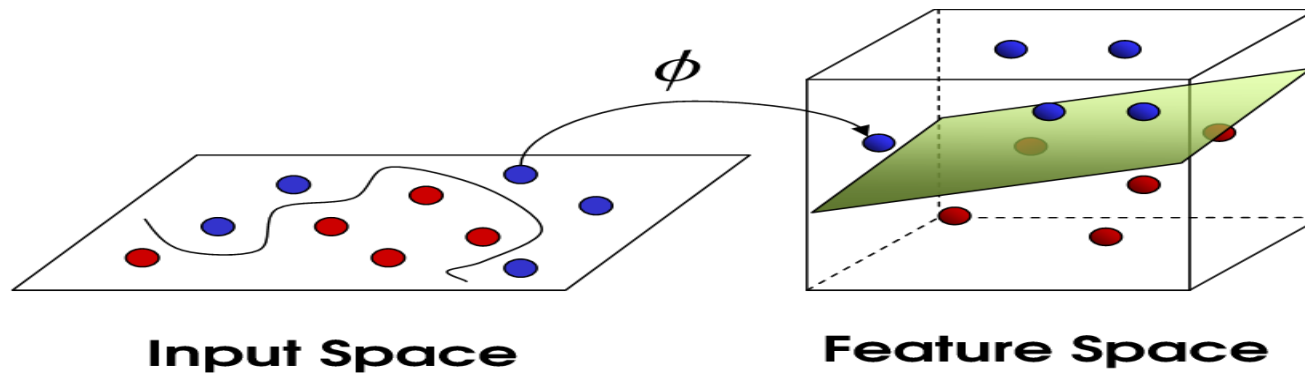
# Kernel Methods

Making the ~~Non-Linear~~ Linear

# When Linear Separators Fail



# Mapping into a New Feature Space



$$\Phi: X \rightarrow \hat{X} = \Phi(\mathbf{x})$$

- For example, with  $\mathbf{x}_i \in \mathbb{R}^2$

$$\Phi([x_{i1}, x_{i2}]) = [x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2]$$

- Rather than run SVM on  $\mathbf{x}_i$ , run it on  $\Phi(\mathbf{x}_i)$ 
  - Find non-linear separator in input space
- What if  $\Phi(\mathbf{x}_i)$  is really big?
- Use kernels to compute it implicitly!



# Kernels

- Find kernel  $K$  such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

- Computing  $K(\mathbf{x}_i, \mathbf{x}_j)$  should be efficient, much more so than computing  $\Phi(\mathbf{x}_i)$  and  $\Phi(\mathbf{x}_j)$
- Use  $K(\mathbf{x}_i, \mathbf{x}_j)$  in SVM algorithm rather than  $\mathbf{x}_i, \mathbf{x}_j$
- Remarkably, this is possible!

# The Polynomial Kernel

Let  $\mathbf{x}_i = [X_{i1}, X_{i2}]$  and  $\mathbf{x}_j = [X_{j1}, X_{j2}]$

Consider the following function:

$$\begin{aligned}
 K(\mathbf{x}_i, \mathbf{x}_j) &= h\mathbf{x}_i, \mathbf{x}_j i^2 \\
 &= (X_{i1}X_{j1} + X_{i2}X_{j2})^2 \\
 &= (X_{i1}^2 X_{j1}^2 + X_{i2}^2 X_{j2}^2 + 2X_{i1}X_{i2}X_{j1}X_{j2}) \\
 &= h\emptyset(\mathbf{x}_i), \emptyset(\mathbf{x}_j) i
 \end{aligned}$$

where

$$\begin{aligned}
 \emptyset(\mathbf{x}_i) &= [X_{i1}^2, X_{i2}^2, \sqrt{2}X_{i1}X_{i2}] \\
 \emptyset(\mathbf{x}_j) &= [X_{j1}^2, X_{j2}^2, \sqrt{2}X_{j1}X_{j2}]
 \end{aligned}$$

# The Polynomial Kernel

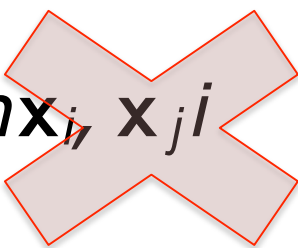
- Given by  $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ 
  - $\Phi(\mathbf{x})$  contains all monomials of degree  $d$
- Useful in visual pattern recognition
  - Example:
    - 16x16 pixel image
    - $10^{10}$  monomials of degree 5
    - Never explicitly compute  $\Phi(\mathbf{x})$  !
- Variation:  $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$ 
  - Adds all lower-order monomials (degrees 1,...,d)!


# The Kernel Trick

“Given an algorithm which is formulated in terms of a positive definite kernel  $K_1$ , one can construct an alternative algorithm by replacing  $K_1$  with another positive definite kernel  $K_2$ ”

➤ SVMs can use the kernel trick

# Incorporating Kernels into SVM

$$J(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$


$$J(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$


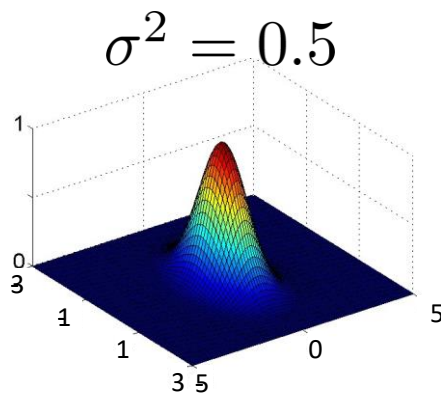
$$\begin{aligned} \text{s.t. } & \alpha_i \geq 0 \quad \forall i \\ & \sum_i \alpha_i y_i = 0 \end{aligned}$$

# The Gaussian Kernel

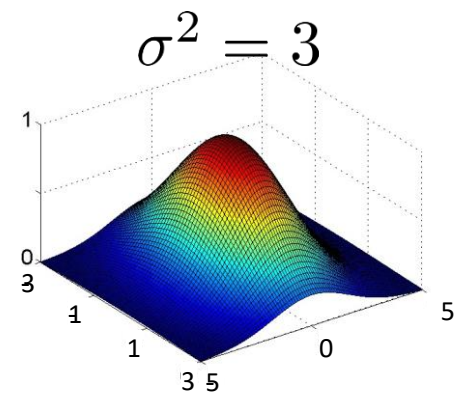
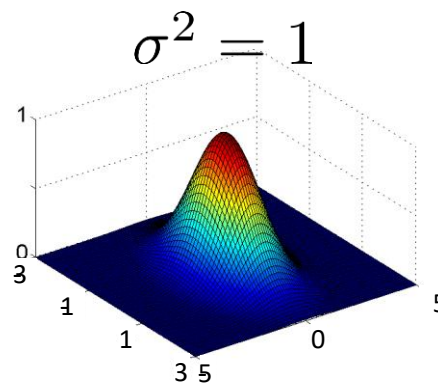
- Also called Radial Basis Function (RBF) kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2} \right)$$

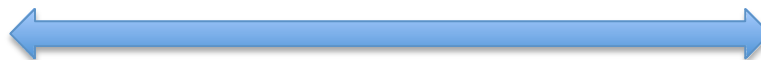
- Has value 1 when  $\mathbf{x}_i = \mathbf{x}_j$
- Value falls off to 0 with increasing distance
- Note: Need to do feature scaling before using Gaussian Kernel



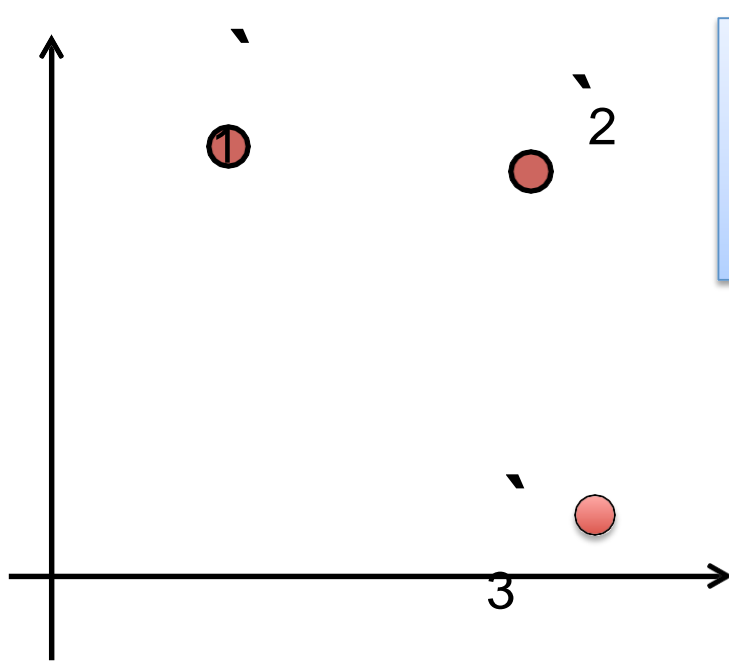
lower bias,  
higher variance



higher bias,  
lower variance



# Gaussian Kernel Example



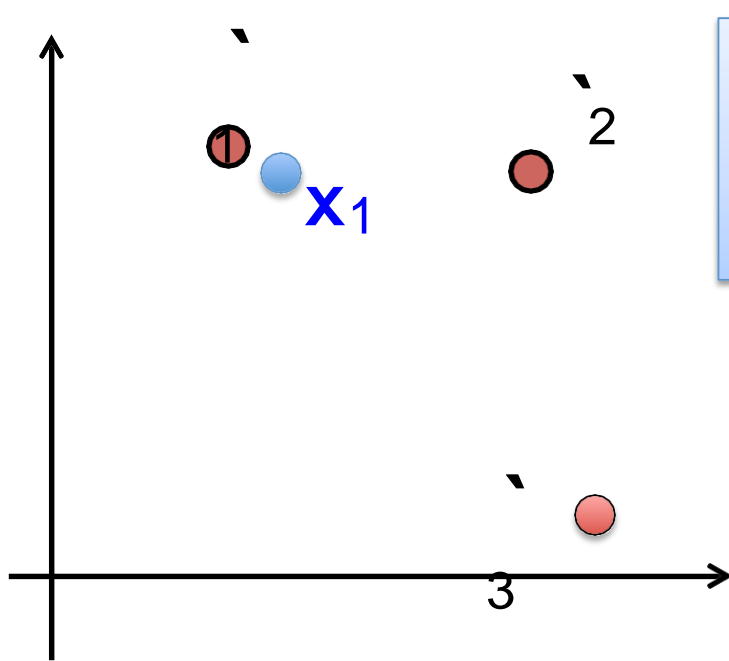
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$$

Imagine we've learned that:

$$\mathbf{w} = [-0.5, 1, 1, 0]$$

Predict +1 if  $\mathbf{w}_0 + \mathbf{w}_1 K(\mathbf{x}, \mathbf{x}_1) + \mathbf{w}_2 K(\mathbf{x}, \mathbf{x}_2) + \mathbf{w}_3 K(\mathbf{x}, \mathbf{x}_3) \leq 0$

# Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$$

Imagine we've learned that:

$$\mathbf{w} = [-0.5, 1, 1, 0]$$

Predict +1 if  $\mathbf{w}_0 + \mathbf{w}_1 K(\mathbf{x}, \mathbf{x}_1) + \mathbf{w}_2 K(\mathbf{x}, \mathbf{x}_2) + \mathbf{w}_3 K(\mathbf{x}, \mathbf{x}_3) \leq 0$

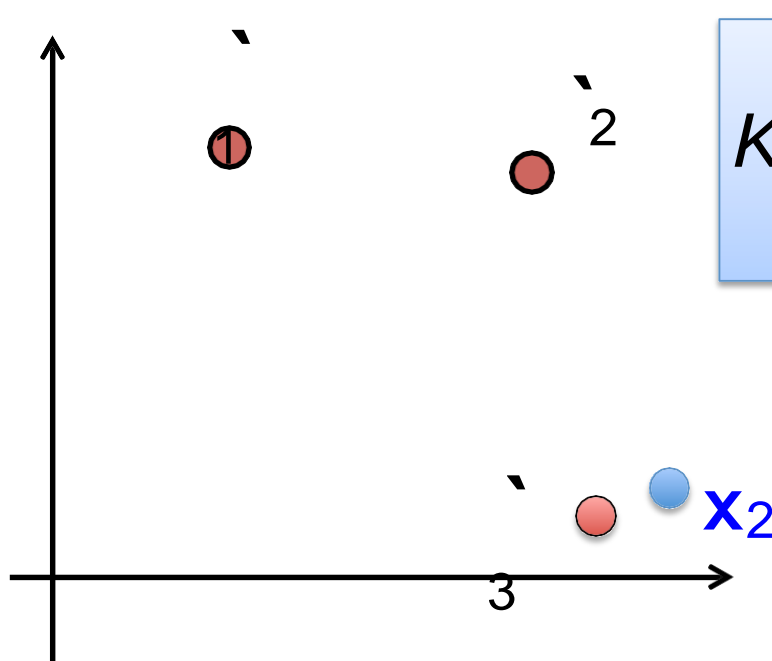
- For  $\mathbf{x}_1$ , we have  $K(\mathbf{x}_1, \mathbf{x}_1) \hat{=} 1$ , other similarities  $\approx 0$

$$\mathbf{w}_0 + \mathbf{w}_1(1) + \mathbf{w}_2(0) + \mathbf{w}_3(0)$$

$$\equiv -0.5 \leq 0 \text{ (so predict +1)}$$



# Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$$

Imagine we've learned that:

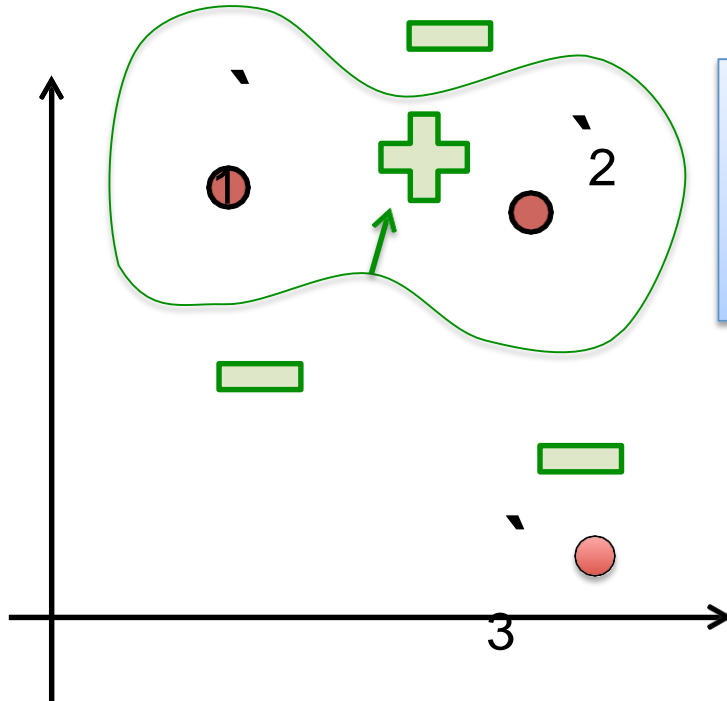
$$\mathbf{w} = [-0.5, 1, 1, 0]$$

Predict +1 if  $\mathbf{w}_0 + \mathbf{w}_1 K(\mathbf{x}, \mathbf{x}_1) + \mathbf{w}_2 K(\mathbf{x}, \mathbf{x}_2) + \mathbf{w}_3 K(\mathbf{x}, \mathbf{x}_3) \leq 0$

- For  $\mathbf{x}_2$ , we have  $K(\mathbf{x}_2, \mathbf{x}_3) \approx 1$ , other similarities  $\approx 0$

$$\begin{aligned} \mathbf{w}_0 + \mathbf{w}_1(0.5) + \mathbf{w}_2(0) + \mathbf{w}_3(1) &= 0(1) + 1(0.5) + 1(0) + 0(1) \\ &= 0.5 < 0, \text{ so predict 1} \end{aligned}$$

# Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$$

Imagine we've learned that:

$$\mathbf{w} = [-0.5, 1, 1, 0]$$

Predict +1 if  $\mathbf{w}_0 + \mathbf{w}_1 K(\mathbf{x}, \mathbf{x}_1) + \mathbf{w}_2 K(\mathbf{x}, \mathbf{x}_2) + \mathbf{w}_3 K(\mathbf{x}, \mathbf{x}_3) \leq 0$

Rough sketch of decision surface

# Other Kernels

- Sigmoid Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i^T \mathbf{x}_j + c)$$

- Neural networks use sigmoid as activation function
- SVM with a sigmoid kernel is equivalent to 2-layer perceptron

- Cosine Similarity Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$$

- Popular choice for measuring similarity of text documents
- $L_2$  norm projects vectors onto the unit sphere; their dot product is the cosine of the angle between the vectors

# Other Kernels

- Chi-squared Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\mu \sum_k \frac{(x_{ik} - x_{jk})^2}{x_{ik} + x_{jk}} \right)$$

- Widely used in computer vision applications
  - Chi-squared measures distance between probability distributions
  - Data is assumed to be non-negative, often with  $L_1$  norm of 1
- String kernels
  - Tree kernels
  - Graph kernels

# An Aside: The Math Behind Kernels

What does it *mean* to be a kernel?

- $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$  for some  $\Phi$

What does it *take* to be a kernel?

- The Gram matrix  $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ 
  - Symmetric matrix
  - Positive semi-definite matrix:

$$\mathbf{z}^T \mathbf{G} \mathbf{z} \geq 0 \text{ for every non-zero vector } \mathbf{z} \in \mathbb{R}^n$$

Establishing “kernel-hood” from first principles is non-trivial

# A Few Good Kernels...

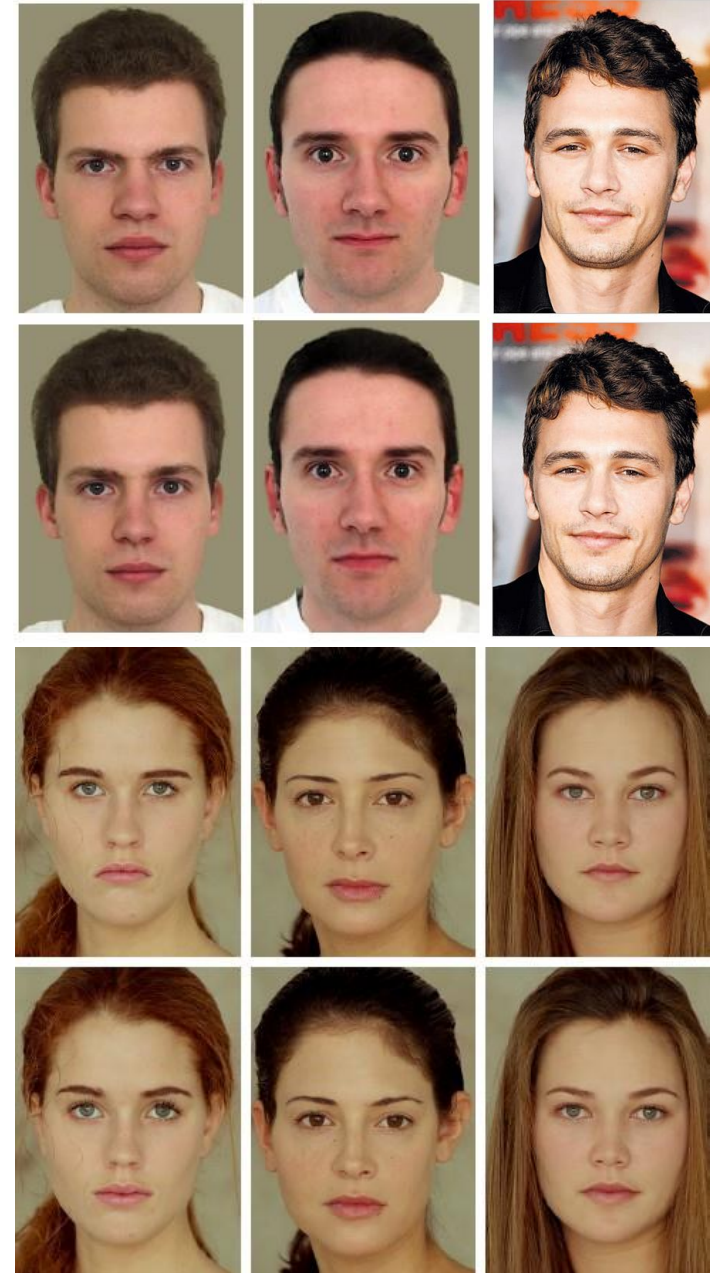
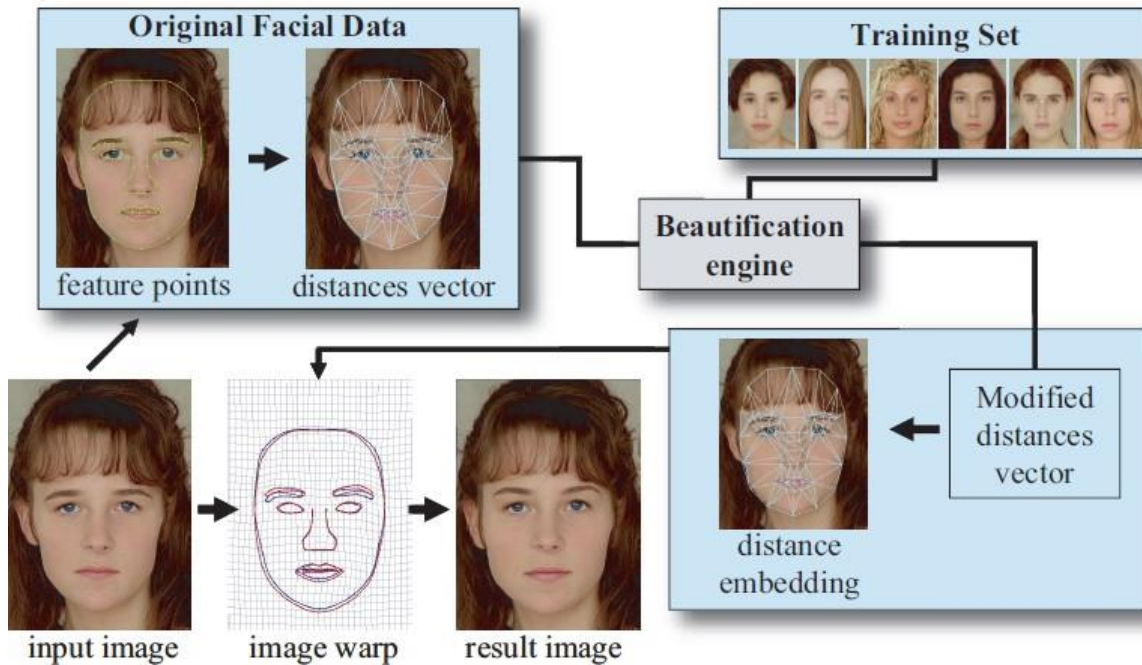
- Linear Kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = h\mathbf{x}_i, \mathbf{x}_j$
- Polynomial kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = (h\mathbf{x}_i, \mathbf{x}_j + c)^d$   
 –  $c \geq 0$  trades off influence of lower order terms
- Gaussian kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$
- Sigmoid kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh \left( \gamma \mathbf{x}_i^T \mathbf{x}_j + c \right)$

Many more...

- Cosine similarity kernel
- Chi-squared kernel
- String/tree/graph/wavelet/etc kernels

# Application: Automatic Photo Retouching

(Leyvand et al., 2008)



# Practical Advice for Applying SVMs

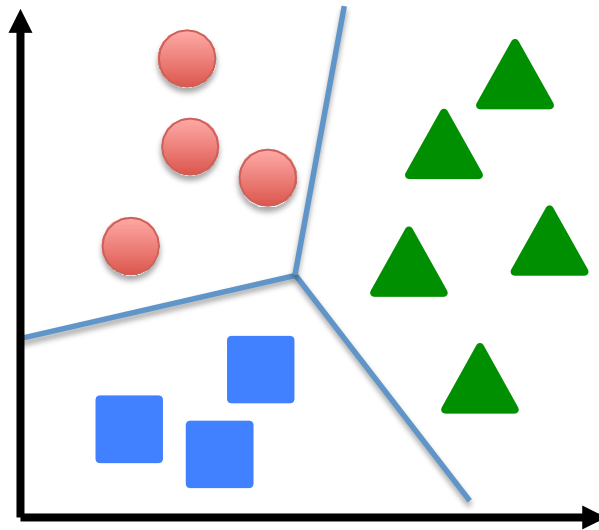
- Use SVM soWware package to solve for parameters
  - e.g., SVMlight, libsvm, cvx (fast!), etc.
- Need to specify:
  - Choice of parameter  $C$
  - Choice of kernel function
    - Associated kernel parameters

e.g.,  $K(\mathbf{x}_i, \mathbf{x}_j) = (\sqrt{h\mathbf{x}_i^T \mathbf{x}_j} + c)^d$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2a^2} \right)$$



# Multi-Class Classification with SVMs



$$y \in \{1, \dots, K\}$$

- Many SVM packages already have multi-class classification built in
- Otherwise, use ~~one-vs-one~~ ~~one-vs-rest~~
  - Train  $K$  SVMs, each picks out one class from rest, yielding  $\checkmark^{(1)}, \dots, \checkmark^{(K)}$
  - Predict class  $i$  with largest  $(\checkmark^{(i)}) \mid \mathbf{x}$

# SVMs vs Logistic Regression

## (Advice from Andrew Ng)

$n$  = #training examples       $d$  = #features

If  $d$  is large (relative to  $n$ )    (e.g.,  $d > n$  with  $d = 10,000$ ,  $n = 10-1,000$ )

- Use logistic regression or SVM with a linear kernel

If  $d$  is small (up to 1,000),  $n$  is intermediate (up to 10,000)

- Use SVM with Gaussian kernel

If  $d$  is small (up to 1,000),  $n$  is large (50,000+)

- Create/add more features, then use logistic regression or SVM without a kernel

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

# Other SVM Variations

- nu SVM
  - nu parameter controls:
    - Fraction of support vectors (lower bound) and misclassification rate (upper bound)
    - E.g.,  $\nu = 0.05$  guarantees that  $\geq 5\%$  of training points are SVs and training error rate is  $\leq 5\%$
  - Harder to optimize than ~~CSVM~~ and not as scalable
- SVMs for regression
- ~~One-class~~ SVMs
- SVMs for clustering
- ...

# Conclusion

- SVMs find optimal linear separator
- The kernel trick makes SVMs learn ~~non-linear~~ decision surfaces
- Strength of SVMs:
  - Good theoretical and empirical performance
  - Supports many types of kernels
- Disadvantages of SVMs:
  - “Slow” to train/predict for huge data sets (but relatively fast!)
  - Need to choose the kernel (and tune its parameters)