

# CS578 Statistical Machine Learning Lecture 3

Jean Honorio  
Purdue University

*(based on slides by Tommi Jaakkola, MIT CSAIL)*

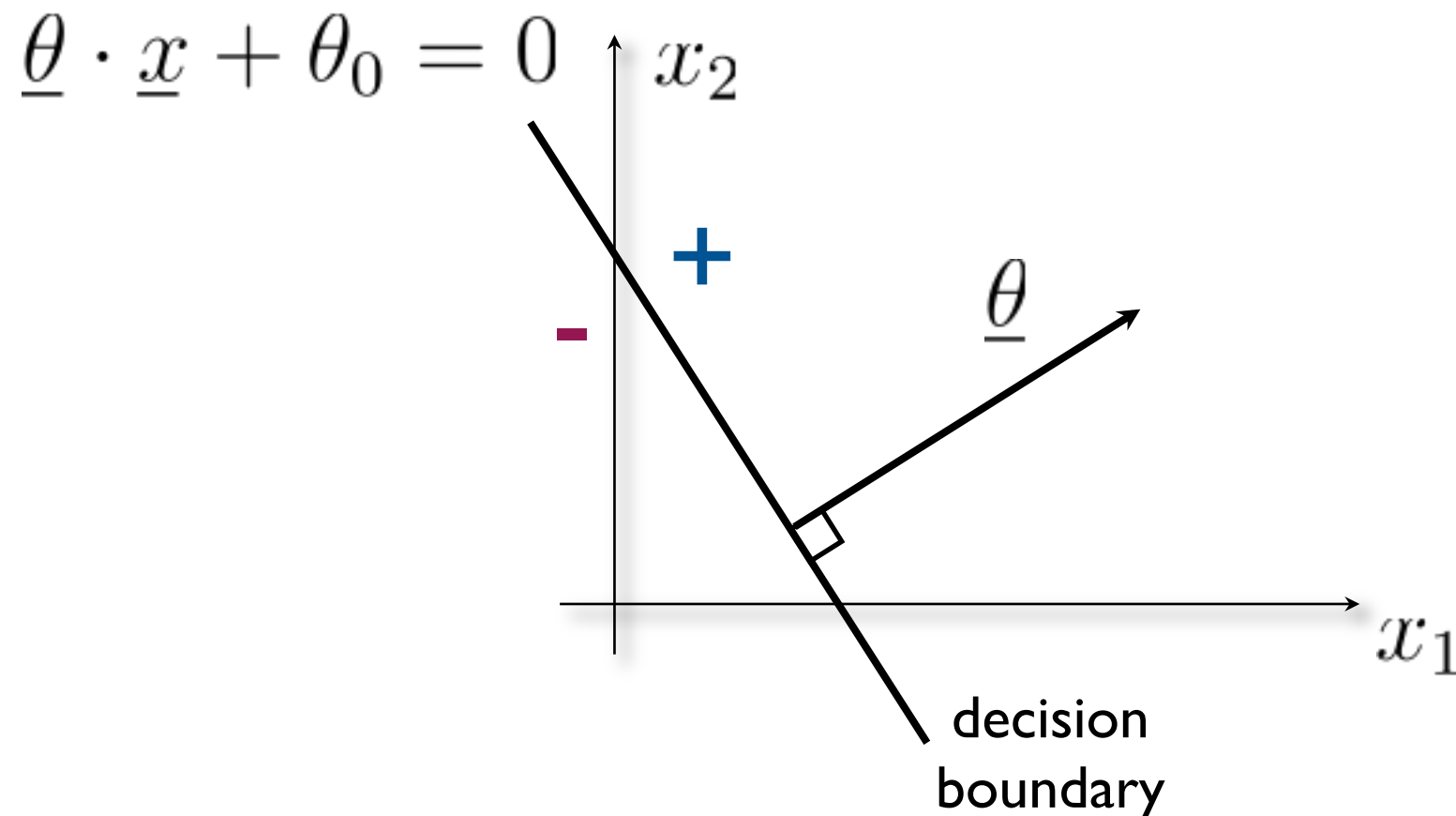
# Today's topics

- Quick review of support vector machines...
- Need for more powerful classifiers
- Feature mappings, non-linear classifiers, kernels
  - non-linear feature mappings
  - kernels, kernel perceptron
- If Homework 0 grade less than or equal to 6.5
  - You might not be able to follow the course
  - Homeworks, midterm, final are more difficult
  - In the past, students got C, D or F
  - **Advise**: prepare on linear algebra, statistics, take course next semester

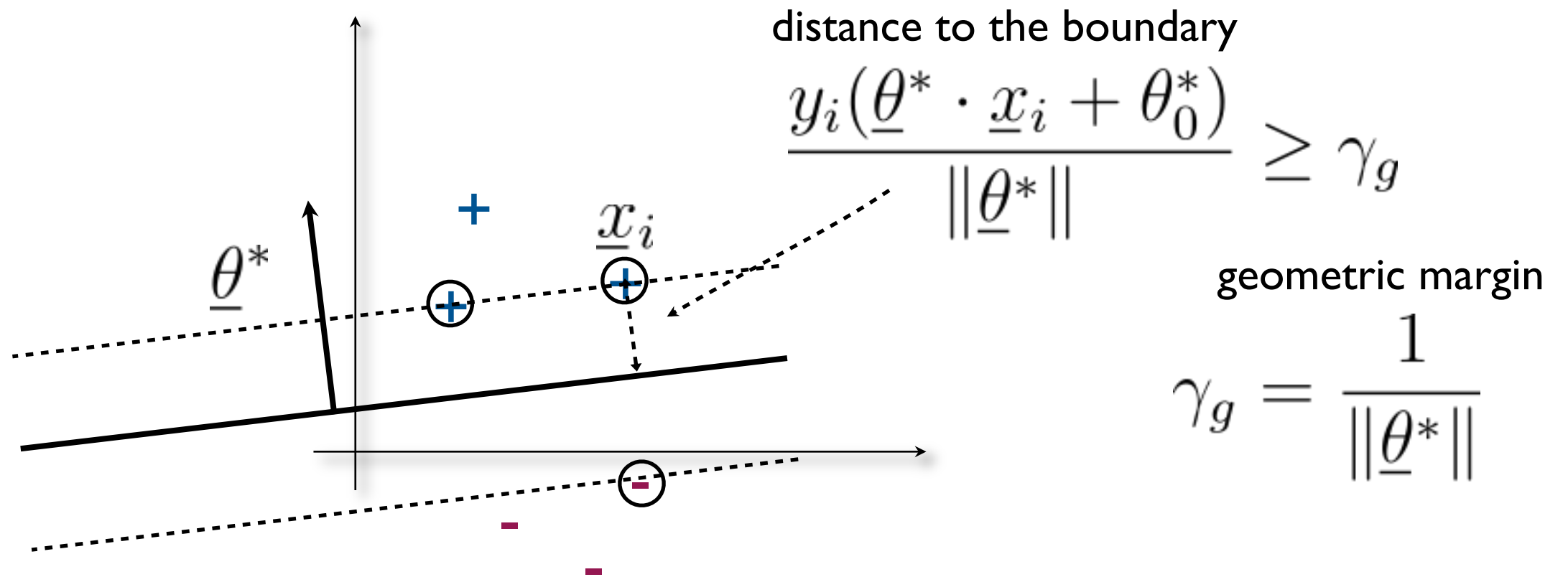
# Linear classifiers (with offset)

- A linear classifier with parameters  $(\underline{\theta}, \theta_0)$

$$\begin{aligned} f(\underline{x}; \underline{\theta}, \theta_0) &= \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0) \\ &= \begin{cases} +1, & \text{if } \underline{\theta} \cdot \underline{x} + \theta_0 > 0 \\ -1, & \text{if } \underline{\theta} \cdot \underline{x} + \theta_0 \leq 0 \end{cases} \end{aligned}$$



# Support vector machine



To find  $\underline{\theta}^*, \theta_0^*$  :

minimize  $\frac{1}{2} \|\underline{\theta}\|^2$  subject to

$$y_i(\underline{\theta} \cdot \underline{x}_i + \theta_0) \geq 1, \quad i = 1, \dots, n$$

- We get a max-margin decision boundary by solving a quadratic programming problem
- The solution is sparse (support vectors)

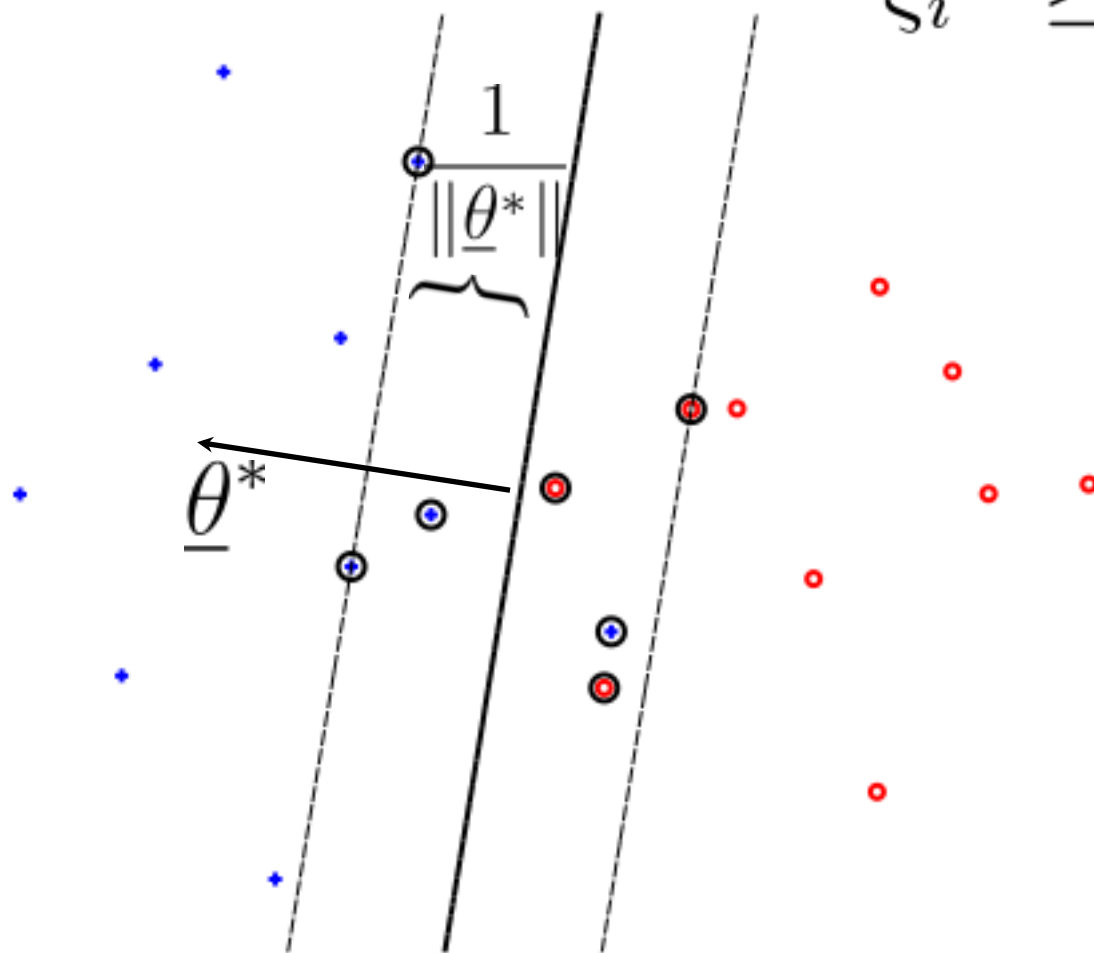
# Support vector machine

- Relaxed quadratic optimization problem

$$\text{minimize } \frac{1}{2} \|\underline{\theta}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{subject to}$$

$$y_i(\underline{\theta} \cdot \underline{x}_i + \theta_0) \geq 1 - \xi_i, \quad i = 1, \dots, n$$

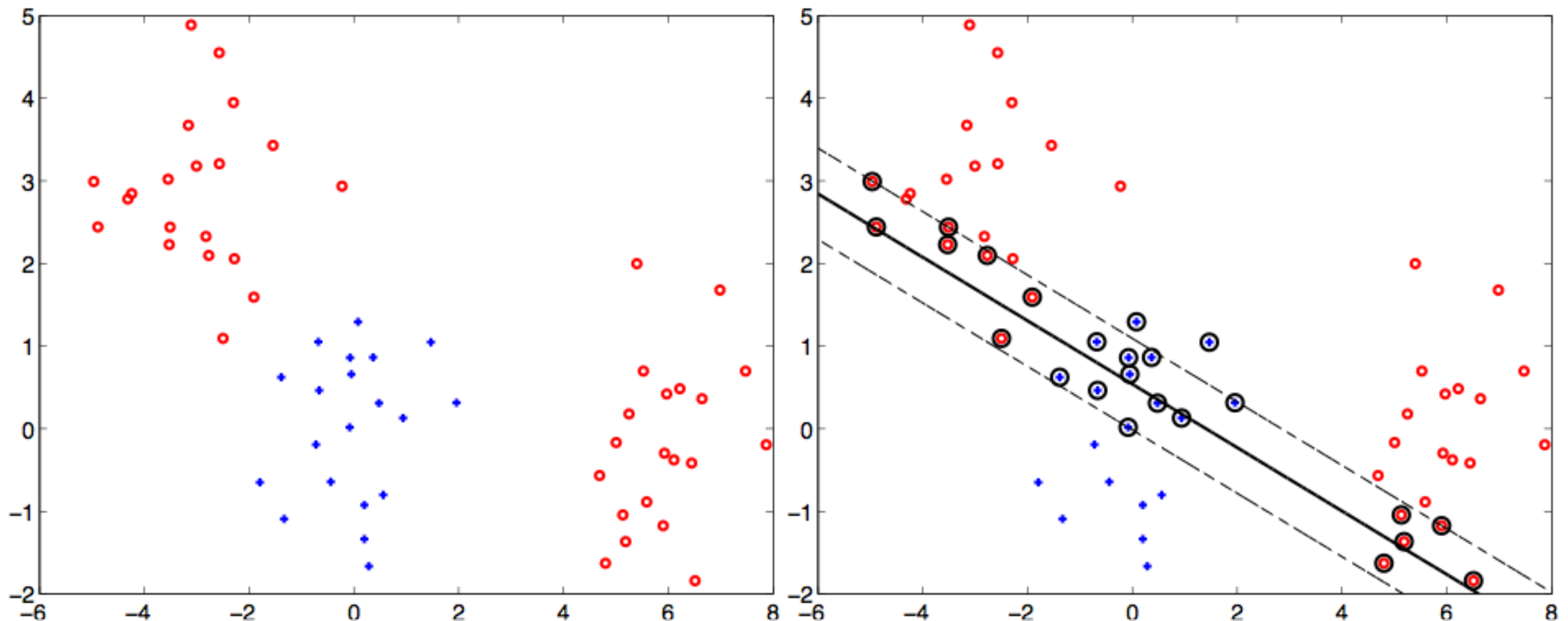
$$\xi_i \geq 0, \quad i = 1, \dots, n$$



The value of  $C$  is an additional parameter we have to set

# Beyond linear classifiers...

- Many problems are not solved well by a linear classifier even if we allow misclassified examples (SVM with slack)
- E.g., data from experiments typically involve “clusters” of different types of examples



# Non-linear feature mappings

- The easiest way to make the classifier more powerful is to add non-linear coordinates to the feature vectors
- The classifier is still linear in the parameters, not inputs

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0)$$

linear classifier

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \phi(\underline{x}) + \theta_0)$$

non-linear classifier

# Non-linear feature mappings

- The easiest way to make the classifier more powerful is to add non-linear coordinates to the feature vectors
- The classifier is still linear in the parameters, not inputs

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0)$$

linear classifier

$$\underline{\theta} \cdot \underline{x} + \theta_0 = 0$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \phi(\underline{x}) + \theta_0)$$

non-linear classifier



# Non-linear feature mappings

- The easiest way to make the classifier more powerful is to add non-linear coordinates to the feature vectors
- The classifier is still linear in the parameters, not inputs

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0)$$

linear classifier

$$\underline{\theta} \cdot \underline{x} + \theta_0 = 0$$

$$\theta_1 x_1 + \theta_2 x_2 + \theta_0 = 0$$

linear decision  
boundary

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \phi(\underline{x}) + \theta_0)$$

non-linear classifier

# Non-linear feature mappings

- The easiest way to make the classifier more powerful is to add non-linear coordinates to the feature vectors
- The classifier is still linear in the parameters, not inputs

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \underline{\phi}(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0)$$

linear classifier

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{\phi}(\underline{x}) + \theta_0)$$

non-linear classifier

$$\underline{\theta} \cdot \underline{\phi}(\underline{x}) + \theta_0 = 0$$

# Non-linear feature mappings

- The easiest way to make the classifier more powerful is to add non-linear coordinates to the feature vectors
- The classifier is still linear in the parameters, not inputs

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{x} + \theta_0)$$

linear classifier

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \phi(\underline{x}) + \theta_0)$$

non-linear classifier

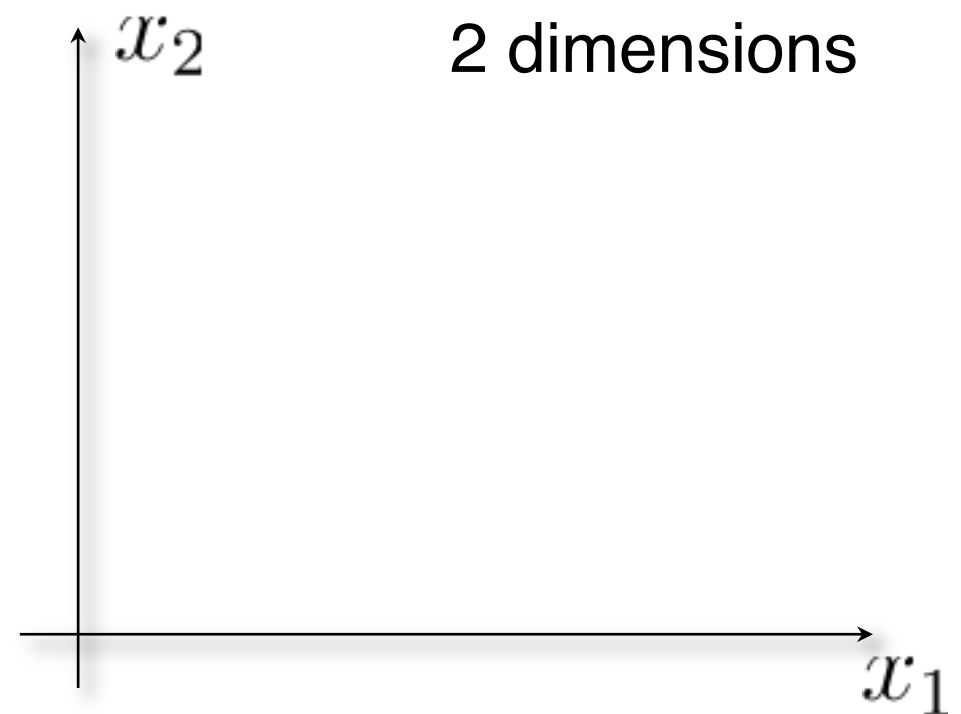
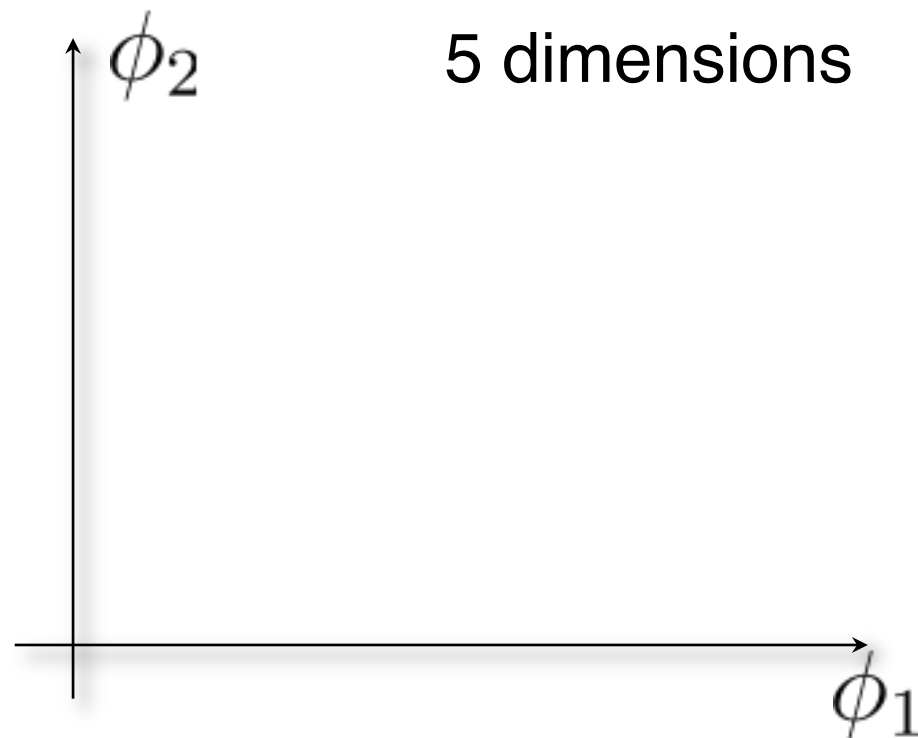
$$\underline{\theta} \cdot \phi(\underline{x}) + \theta_0 = 0$$

$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 \sqrt{2} x_1 x_2 + \theta_5 x_2^2 + \theta_0 = 0$$

non-linear decision boundary

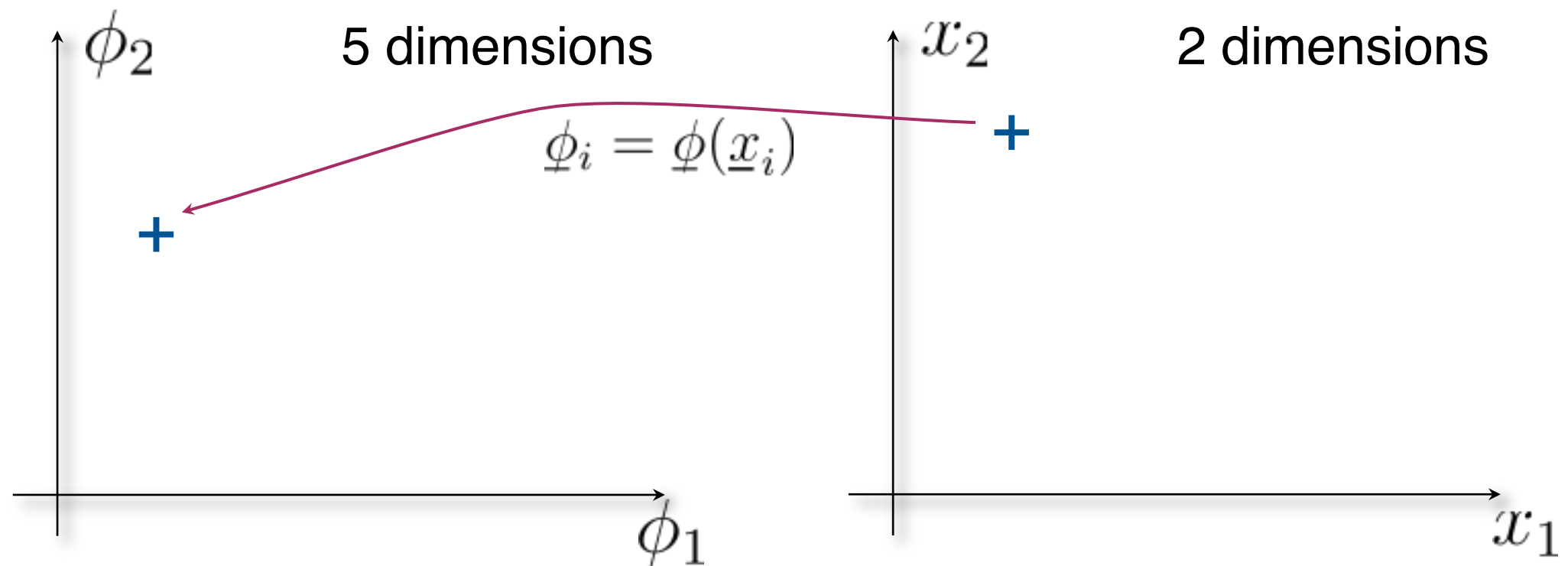
# Non-linear feature mappings

- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



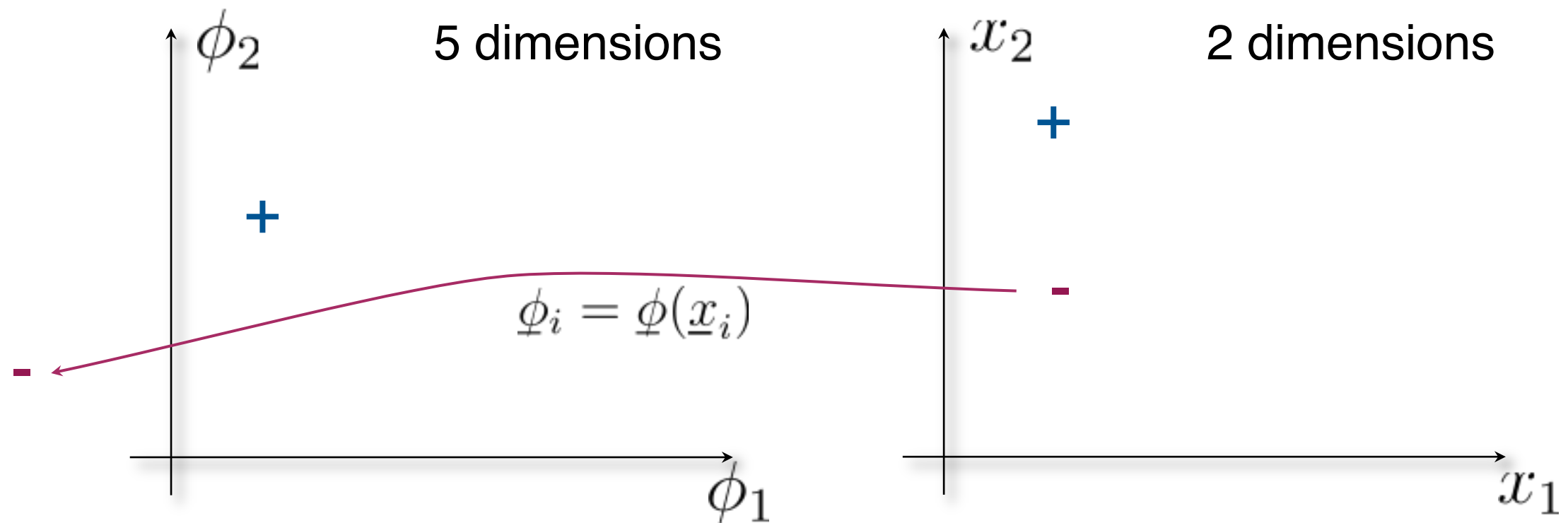
# Non-linear feature mappings

- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



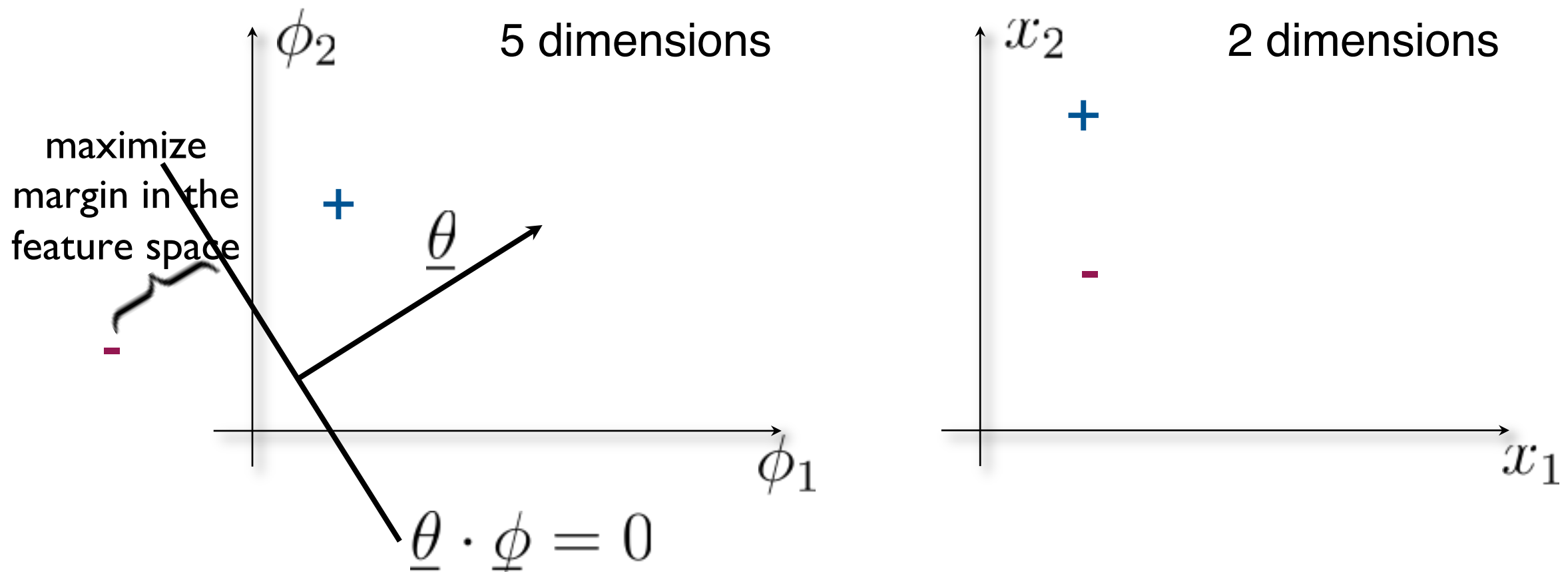
# Non-linear feature mappings

- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



# Non-linear feature mappings

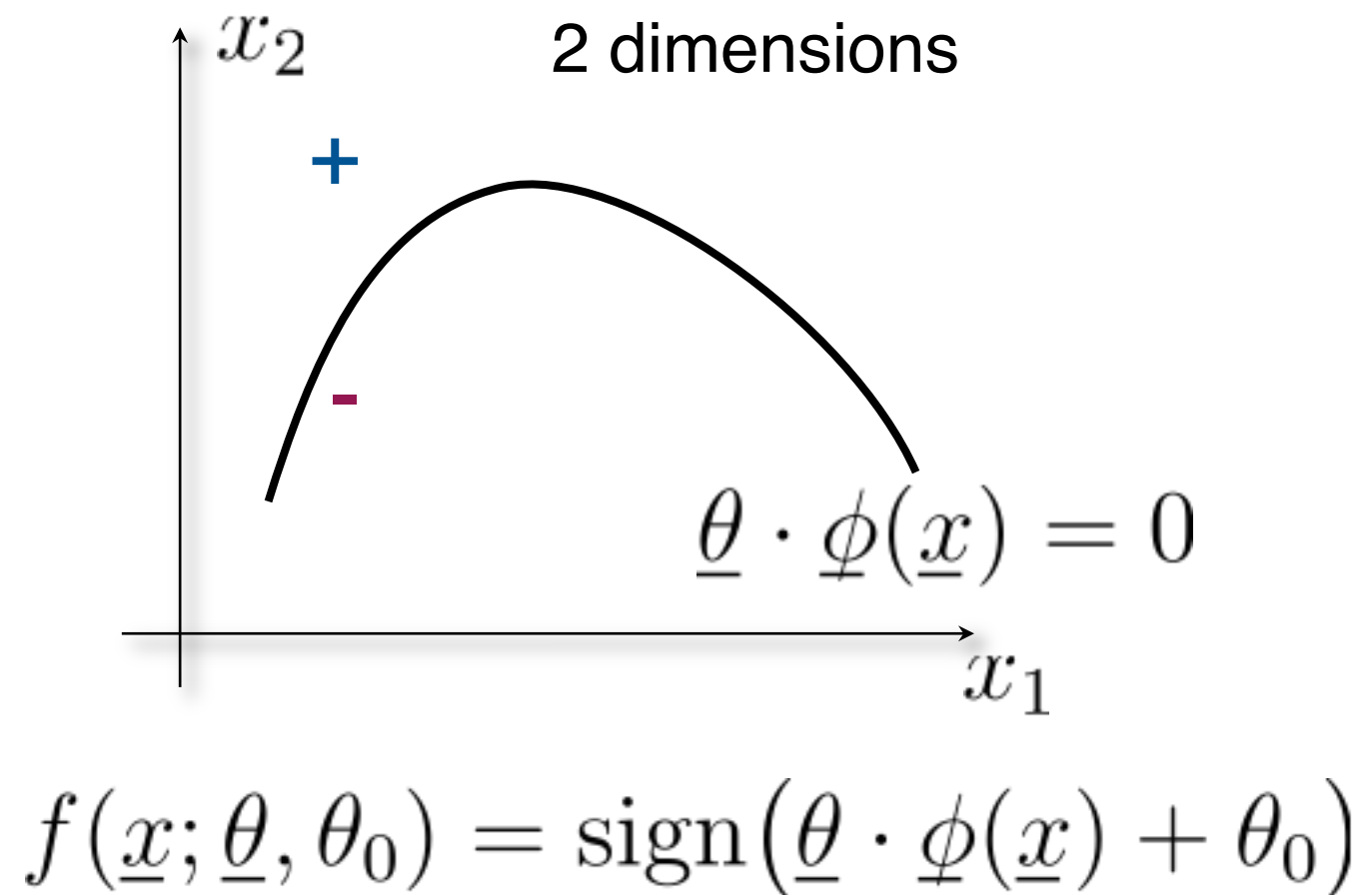
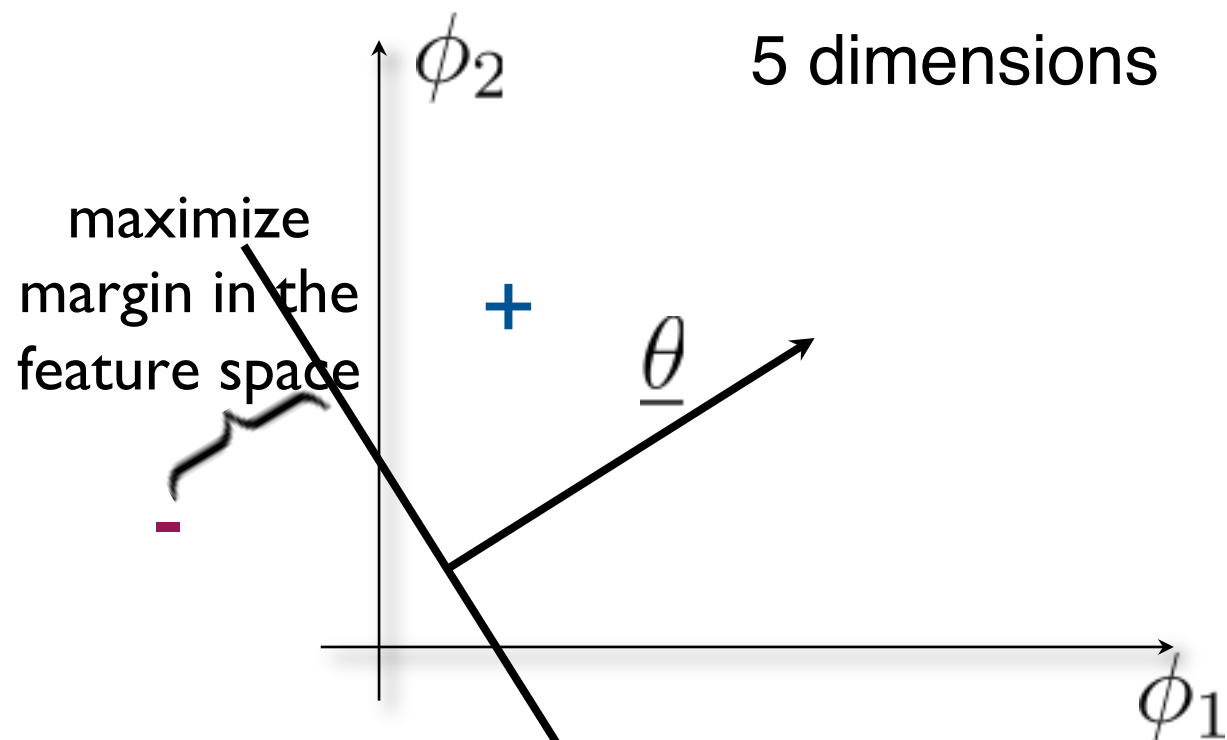
- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



$$f(\underline{\phi}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{\phi} + \theta_0)$$

# Non-linear feature mappings

- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



$$f(\underline{\phi}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{\phi} + \theta_0)$$



# Learning non-linear classifiers

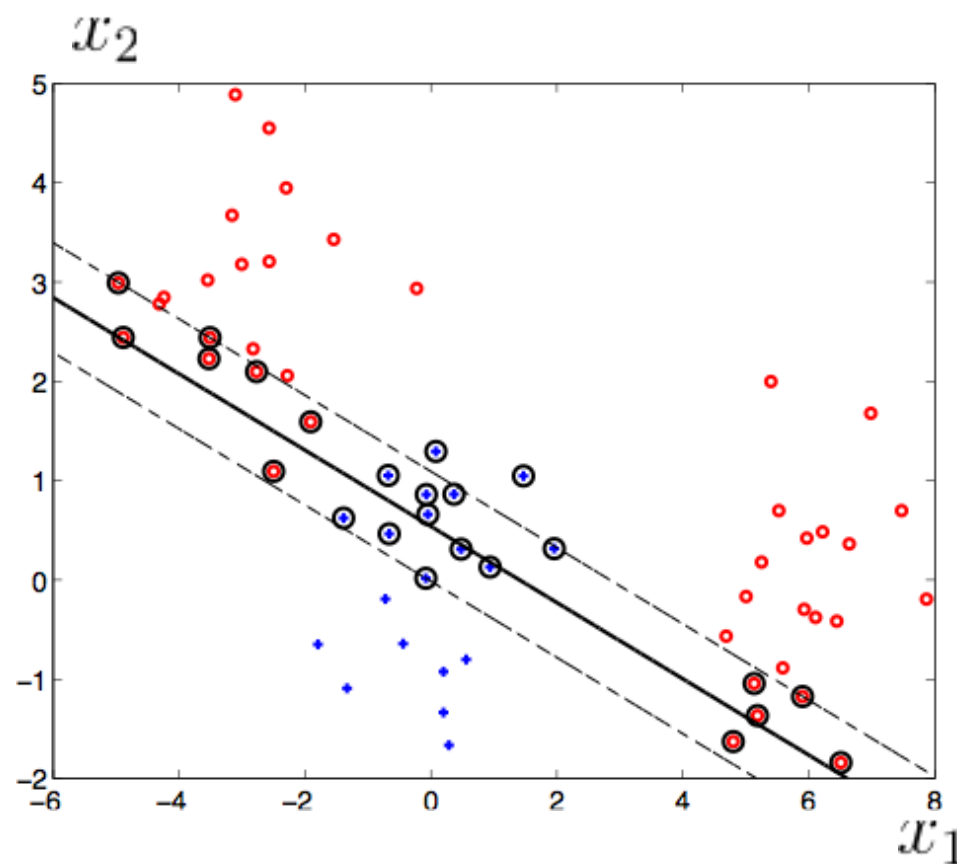
- We can apply the same SVM formulation, just replacing the input examples with (higher dimensional) feature vectors

$$\begin{aligned} \text{minimize } & \frac{1}{2} \|\underline{\theta}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{subject to} \\ & y_i(\underline{\theta} \cdot \underline{\phi}(x_i) + \theta_0) \geq 1 - \xi_i, \quad i = 1, \dots, n \\ & \xi_i \geq 0, \quad i = 1, \dots, n \end{aligned}$$

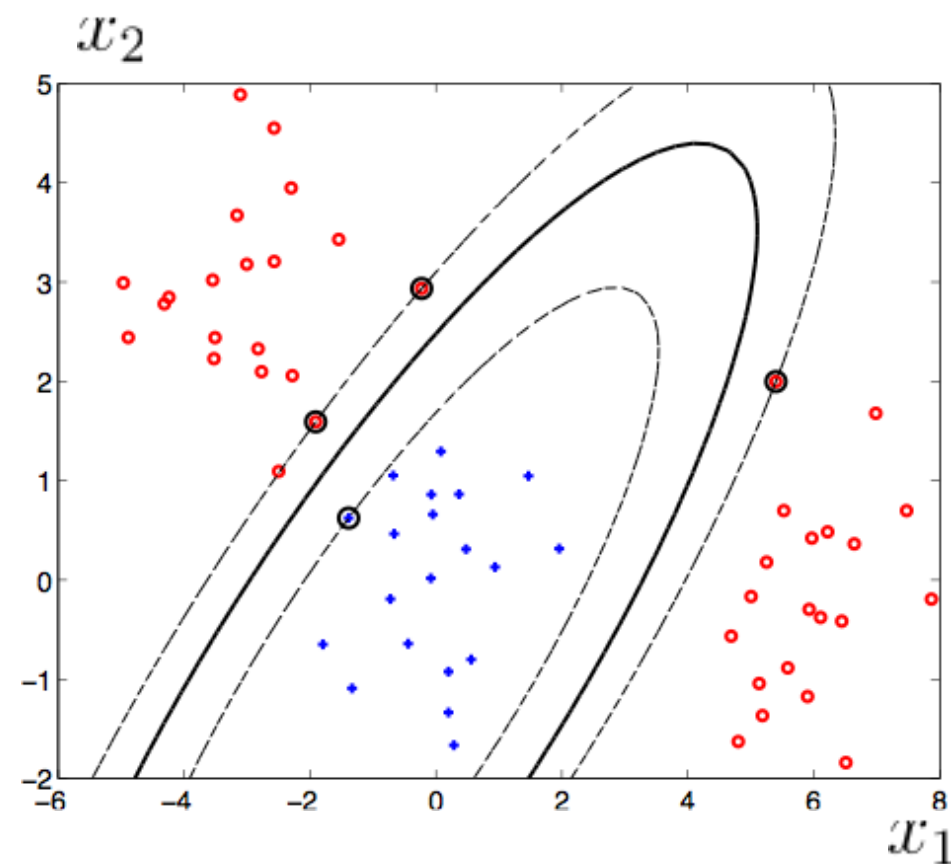
- Note that the cost of solving this quadratic programming problem increases with the dimension of the feature vectors

# Non-linear classifiers

- Many (low dimensional) problems are not solved well by a linear classifier even with slack
- By mapping examples to feature vectors, and maximizing a linear margin in the feature space, we obtain non-linear margin curves in the original space



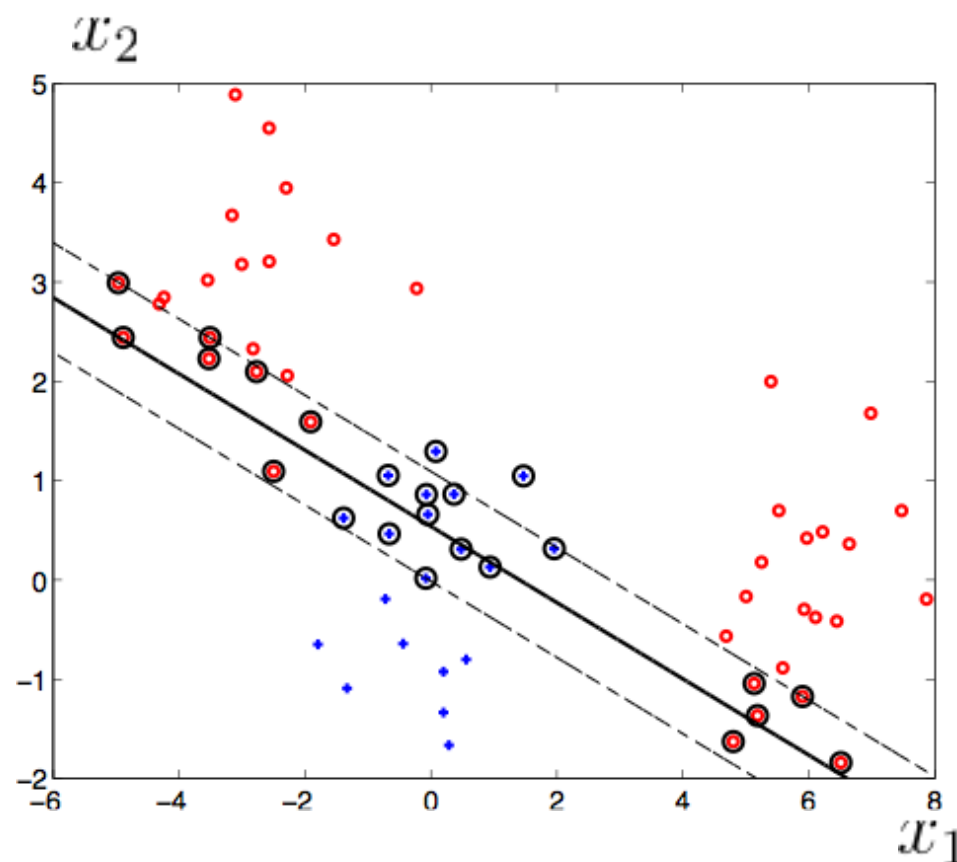
linear features



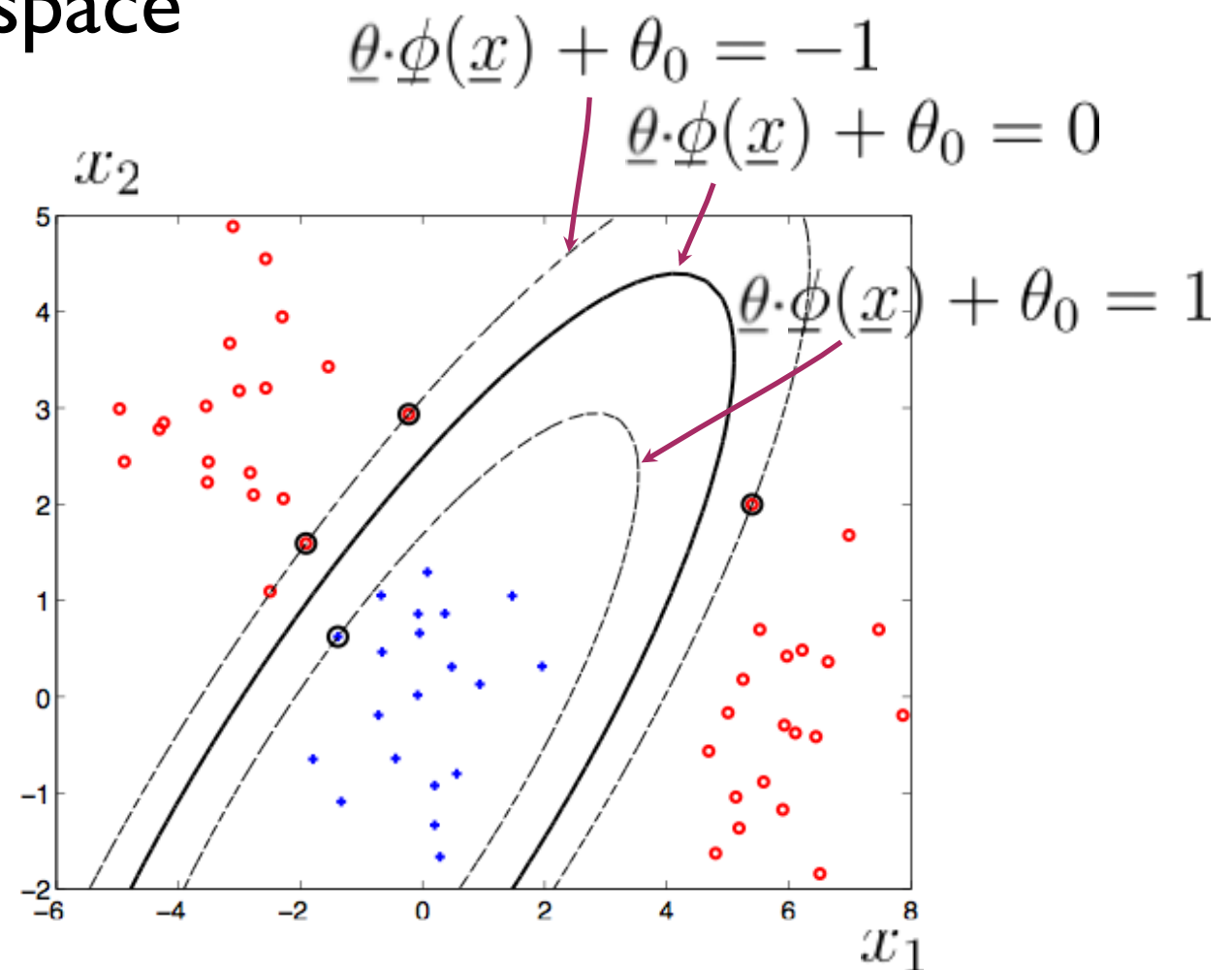
2nd order features

# Non-linear classifiers

- Many (low dimensional) problems are not solved well by a linear classifier even with slack
- By mapping examples to feature vectors, and maximizing a linear margin in the feature space, we obtain non-linear margin curves in the original space



linear features

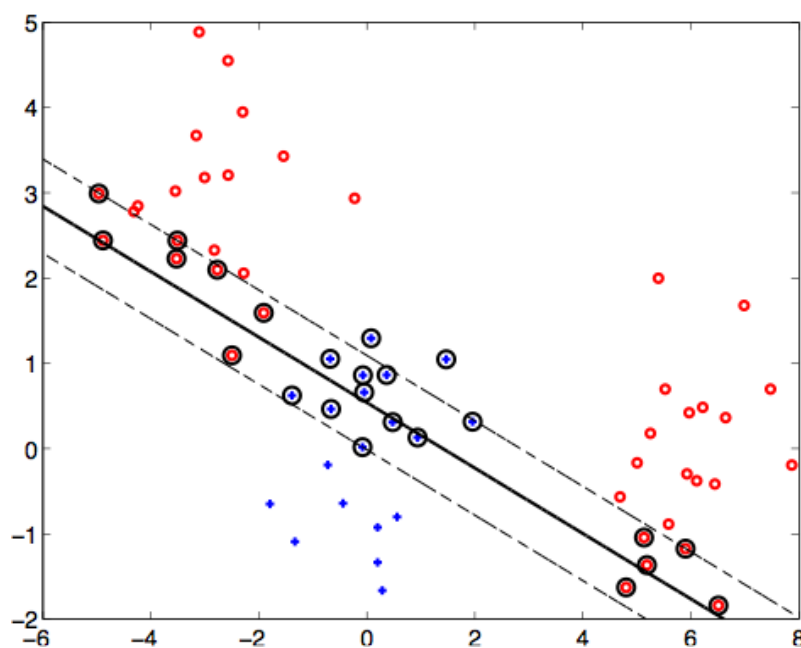


2nd order features

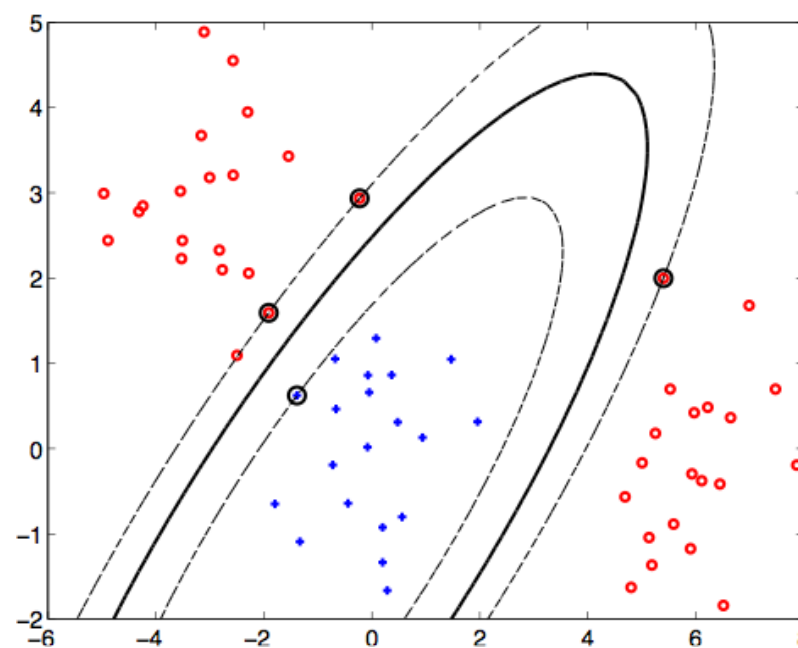
# Problems to resolve

By using non-linear feature mappings we get more powerful sets of classifiers

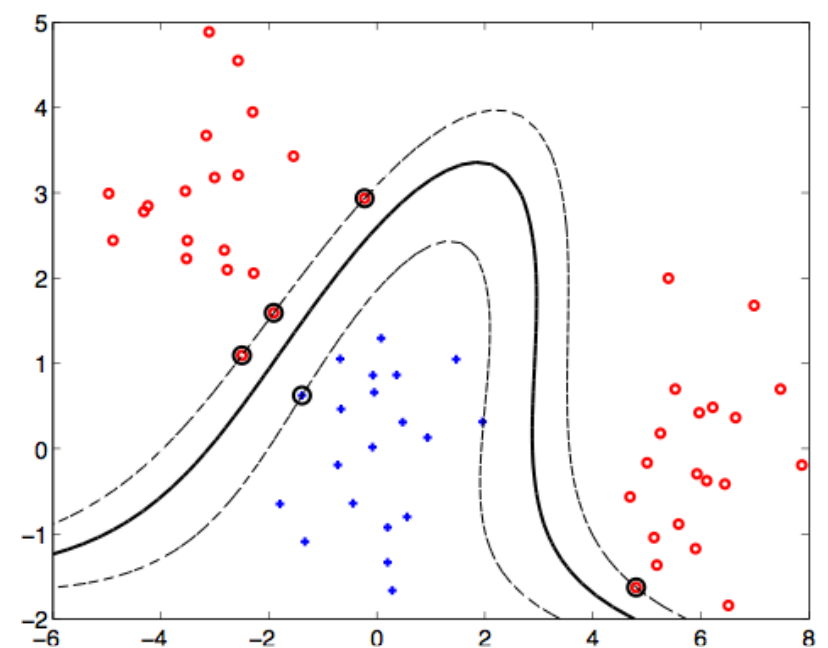
- Computational efficiency?
  - the cost of using higher dimensional feature vectors (seems to) increase with the dimension
- Model selection?
  - how do we choose among different feature mappings?



linear features



2nd order features



3rd order features

# Non-linear perceptron, kernels

- Non-linear feature mappings can be dealt with more efficiently through their inner products or “kernels”
- We will begin by turning the perceptron classifier with non-linear features into a “kernel perceptron”

- Predicting with feature mapping:

$$f(\underline{x}; \underline{\theta}) = \text{sign}(\underline{\theta} \cdot \underline{\phi}(\underline{x}))$$

- Perceptron with feature mapping:

Initialize:  $\underline{\theta} = 0$

Repeat until convergence:

for  $t = 1, \dots, n$

if  $y_t(\underline{\theta} \cdot \underline{\phi}(\underline{x}_t)) \leq 0$  (mistake)

$$\underline{\theta} \leftarrow \underline{\theta} + y_t \underline{\phi}(\underline{x}_t)$$

# On perceptron updates

- Each update adds  $y_t \phi(\underline{x}_t)$  to the parameter vector
- Repeated updates on the same example simply result in adding the same term multiple times
- We can therefore write the current perceptron solution as a function of how many times we performed an update on each training example

$$\underline{\theta} = \sum_{i=1}^n \alpha_i y_i \phi(\underline{x}_i)$$

$$\alpha_i \in \{0, 1, \dots\}, \quad \sum_{i=1}^n \alpha_i = \# \text{ of mistakes}$$

# Kernel perceptron

- By switching to the “count” representation, we can write the perceptron algorithm entirely in terms of inner products between the feature vectors

$$f(\underline{x}; \underline{\theta}) = \text{sign}(\underline{\theta} \cdot \underline{\phi}(\underline{x})) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i [\underline{\phi}(\underline{x}_i) \cdot \underline{\phi}(\underline{x})]\right)$$

- Algorithm:

Initialize:  $\alpha_i = 0, i = 1, \dots, n$

Repeat until convergence:

for  $t = 1, \dots, n$

if  $y_t \left( \sum_{i=1}^n \alpha_i y_i [\underline{\phi}(\underline{x}_i) \cdot \underline{\phi}(\underline{x}_t)] \right) \leq 0$  (mistake)

$\alpha_t \leftarrow \alpha_t + 1$

# Kernel perceptron

- By switching to the “count” representation, we can write the perceptron algorithm entirely in terms of inner products between the feature vectors

$$f(\underline{x}; \underline{\theta}) = \text{sign}(\underline{\theta} \cdot \underline{\phi}(\underline{x})) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i [\underline{\phi}(\underline{x}_i) \cdot \underline{\phi}(\underline{x})]\right)$$

- Algorithm:

Initialize:  $\alpha_i = 0, i = 1, \dots, n$

Repeat until convergence:

for  $t = 1, \dots, n$

if  $y_t \left( \sum_{i=1}^n \alpha_i y_i [\underline{\phi}(\underline{x}_i) \cdot \underline{\phi}(\underline{x}_t)] \right) \leq 0$  (mistake)

$\alpha_t \leftarrow \alpha_t + 1$



# Why inner products?

- For some feature mappings, the inner products can be evaluated efficiently, without first expanding the feature vectors

$$\phi(\underline{x}) \cdot \phi(\underline{x}') = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} x'_1 \\ x'_2 \\ x'^2_1 \\ \sqrt{2}x'_1x'_2 \\ x'^2_2 \end{bmatrix}$$

$$= (x_1x'_1) + (x_2x'_2) + (x_1x'_1)^2 + 2(x_1x'_1)(x_2x'_2) + (x_2x'_2)^2$$

$$= (x_1x'_1 + x_2x'_2) + (x_1x'_1 + x_2x'_2)^2$$

$$= (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2$$

# Why inner products?

- Instead of explicitly constructing feature vectors, we can try to explicate their inner product or “kernel”

$$\begin{aligned}\phi(\underline{x}) \cdot \phi(\underline{x}') &= \begin{bmatrix} ? \\ ? \end{bmatrix} \cdot \begin{bmatrix} ? \\ ? \end{bmatrix} \\ &= (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2\end{aligned}$$

- What is  $\phi(\underline{x})$ ?
- $$\phi(\underline{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

# Why inner products?

- Instead of explicitly constructing feature vectors, we can try to explicate their inner product or “kernel”

$$\phi(\underline{x}) \cdot \phi(\underline{x}') = \begin{bmatrix} ? \\ \end{bmatrix} \cdot \begin{bmatrix} ? \\ \end{bmatrix}$$

$$= (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2 + (\underline{x} \cdot \underline{x}')^3 + (\underline{x} \cdot \underline{x}')^4$$

- What is  $\phi(\underline{x})$  now? Does it even exist?

# Feature mappings and kernels

- In the kernel perceptron algorithm, the feature vectors appear only as inner products
- Instead of explicitly constructing feature vectors, we can try to explicate their inner product or kernel
- $K : \mathcal{R}^d \times \mathcal{R}^d \rightarrow \mathcal{R}$  is a kernel function if there exists a feature mapping such that

$$K(\underline{x}, \underline{x}') = \underline{\phi}(\underline{x}) \cdot \underline{\phi}(\underline{x}')$$

# Feature mappings and kernels

- In the kernel perceptron algorithm, the feature vectors appear only as inner products
- Instead of explicitly constructing feature vectors, we can try to explicate their inner product or kernel
- $K : \mathcal{R}^d \times \mathcal{R}^d \rightarrow \mathcal{R}$  is a kernel function if there exists a feature mapping such that

$$K(\underline{x}, \underline{x}') = \phi(\underline{x}) \cdot \phi(\underline{x}')$$

- Examples of polynomial kernels

$$K(\underline{x}, \underline{x}') = (\underline{x} \cdot \underline{x}')$$

$$K(\underline{x}, \underline{x}') = (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2$$

$$K(\underline{x}, \underline{x}') = (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2 + (\underline{x} \cdot \underline{x}')^3$$

$$K(\underline{x}, \underline{x}') = (1 + \underline{x} \cdot \underline{x}')^p, \quad p = 1, 2, \dots$$

# Composition rules for kernels

- We can construct valid kernels from simple components
- For any function  $f : R^d \rightarrow R$ , if  $K_1$  is a kernel, then so is

$$1) \quad K(\underline{x}, \underline{x}') = f(\underline{x})K_1(\underline{x}, \underline{x}')f(\underline{x}')$$

- The set of kernel functions is closed under addition and multiplication: if  $K_1$  and  $K_2$  are kernels, then so are

$$2) \quad K(\underline{x}, \underline{x}') = K_1(\underline{x}, \underline{x}') + K_2(\underline{x}, \underline{x}')$$

$$3) \quad K(\underline{x}, \underline{x}') = K_1(\underline{x}, \underline{x}')K_2(\underline{x}, \underline{x}')$$

- The composition rules are also helpful in verifying that a kernel is valid (i.e., corresponds to an inner product of some feature vectors)

# Radial basis kernel

- The feature “vectors” corresponding to kernels may also be infinite dimensional (functions)
- This is the case, e.g., for the radial basis kernel

$$K(\underline{x}, \underline{x}') = \exp \left( -\beta \|\underline{x} - \underline{x}'\|^2 \right), \quad \beta > 0$$

- Any distinct set of training points, regardless of their labels, are separable using this kernel function!

# Radial basis kernel

- The feature “vectors” corresponding to kernels may also be infinite dimensional (functions)
- This is the case, e.g., for the radial basis kernel

$$K(\underline{x}, \underline{x}') = \exp \left( -\beta \|\underline{x} - \underline{x}'\|^2 \right), \quad \beta > 0$$

- Any distinct set of training points, regardless of their labels, are separable using this kernel function!
- We can use the composition rules to show that this is indeed a valid kernel

$$\exp\{-\beta\|\underline{x} - \underline{x}'\|^2\} = \exp\{-\beta\underline{x} \cdot \underline{x} + 2\beta\underline{x} \cdot \underline{x}' - \beta\underline{x}' \cdot \underline{x}'\}$$



# Radial basis kernel

- The feature “vectors” corresponding to kernels may also be infinite dimensional (functions)
- This is the case, e.g., for the radial basis kernel

$$K(\underline{x}, \underline{x}') = \exp \left( -\beta \|\underline{x} - \underline{x}'\|^2 \right), \quad \beta > 0$$

- Any distinct set of training points, regardless of their labels, are separable using this kernel function!
- We can use the composition rules to show that this is indeed a valid kernel

$$\begin{aligned} \exp\{-\beta \|\underline{x} - \underline{x}'\|^2\} &= \exp\{-\beta \underline{x} \cdot \underline{x} + 2\beta \underline{x} \cdot \underline{x}' - \beta \underline{x}' \cdot \underline{x}'\} \\ &= \overbrace{\exp\{-\beta \underline{x} \cdot \underline{x}\}}^{f(\underline{x})} \exp\{2\beta \underline{x} \cdot \underline{x}'\} \overbrace{\exp\{-\beta \underline{x}' \cdot \underline{x}'\}}^{f(\underline{x}')} \end{aligned}$$

# Radial basis kernel

- The feature “vectors” corresponding to kernels may also be infinite dimensional (functions)
- This is the case, e.g., for the radial basis kernel

$$K(\underline{x}, \underline{x}') = \exp \left( -\beta \|\underline{x} - \underline{x}'\|^2 \right), \quad \beta > 0$$

- Any distinct set of training points, regardless of their labels, are separable using this kernel function!
- We can use the composition rules to show that this is indeed a valid kernel

$$\begin{aligned} \exp\{-\beta \|\underline{x} - \underline{x}'\|^2\} &= \exp\{-\beta \underline{x} \cdot \underline{x} + 2\beta \underline{x} \cdot \underline{x}' - \beta \underline{x}' \cdot \underline{x}'\} \\ &= \overbrace{\exp\{-\beta \underline{x} \cdot \underline{x}\}}^{f(\underline{x})} \exp\{2\beta \underline{x} \cdot \underline{x}'\} \overbrace{\exp\{-\beta \underline{x}' \cdot \underline{x}'\}}^{f(\underline{x}')} \\ &= f(\underline{x}) \underbrace{\left( 1 + 2\beta(\underline{x} \cdot \underline{x}') + \dots \right)}_{\text{Infinite Taylor series expansion}} f(\underline{x}') \end{aligned}$$

$$e^z = \sum_{k=0}^{\infty} \frac{z^k}{k!} = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \dots$$

← Infinite Taylor series expansion

# Kernel perceptron cont'd

- We can now apply the kernel perceptron algorithm without ever explicating the feature vectors

$$f(\underline{x}; \alpha) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(\underline{x}_i, \underline{x})\right)$$

- Algorithm:

Initialize:  $\alpha_i = 0, i = 1, \dots, n$

Repeat until convergence:

for  $t = 1, \dots, n$

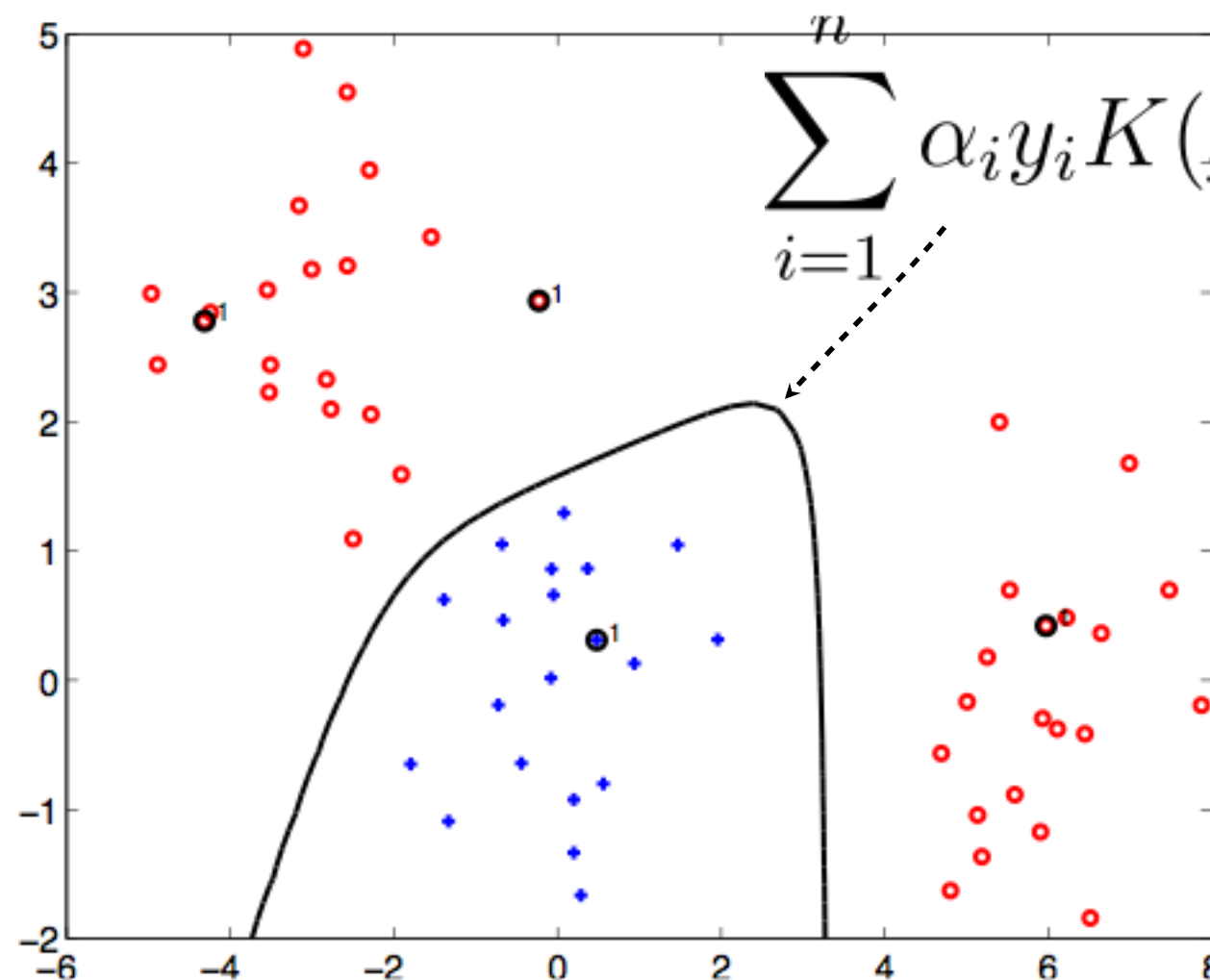
if  $y_t \left( \sum_{i=1}^n \alpha_i y_i K(\underline{x}_i, \underline{x}_t) \right) \leq 0$  (mistake)

$\alpha_t \leftarrow \alpha_t + 1$

# Kernel perceptron: example

- With a radial basis kernel

$$f(\underline{x}; \alpha) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(\underline{x}_i, \underline{x})\right)$$



only four  
updates in  
the perceptron  
algorithm

# Kernel SVM

- We can also turn SVM into its dual (kernel) form and implicitly find the max-margin linear separator in the feature space, e.g., corresponding to the radial basis kernel

$$f(\underline{x}; \alpha) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(\underline{x}_i, \underline{x}) + \theta_0\right)$$

