# CMSC 478
# Machine Learning

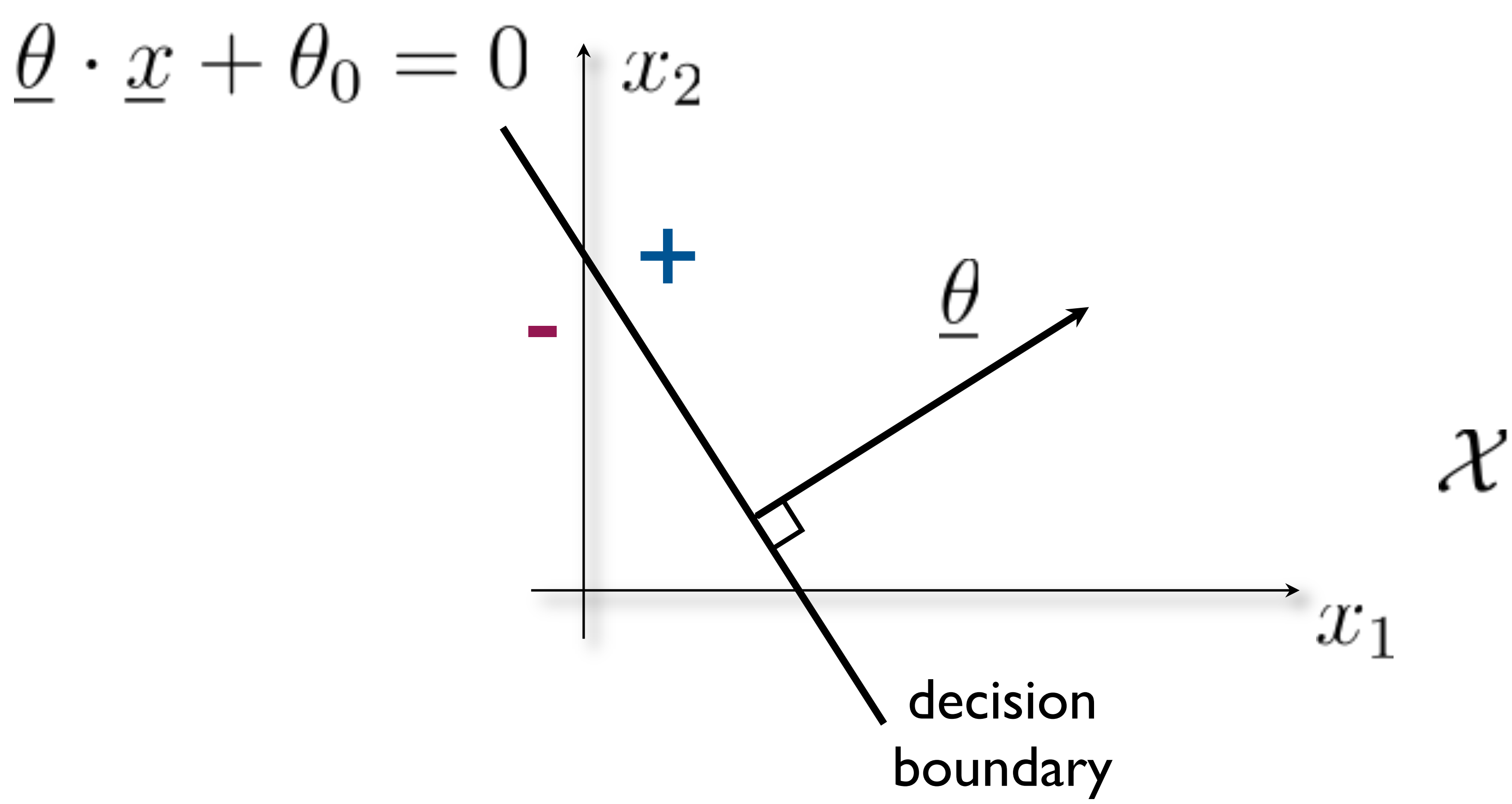
KMA Solaiman
ksolaima@umbc.edu

*(originally prepared by Tommi Jaakkola, MIT CSAIL)*

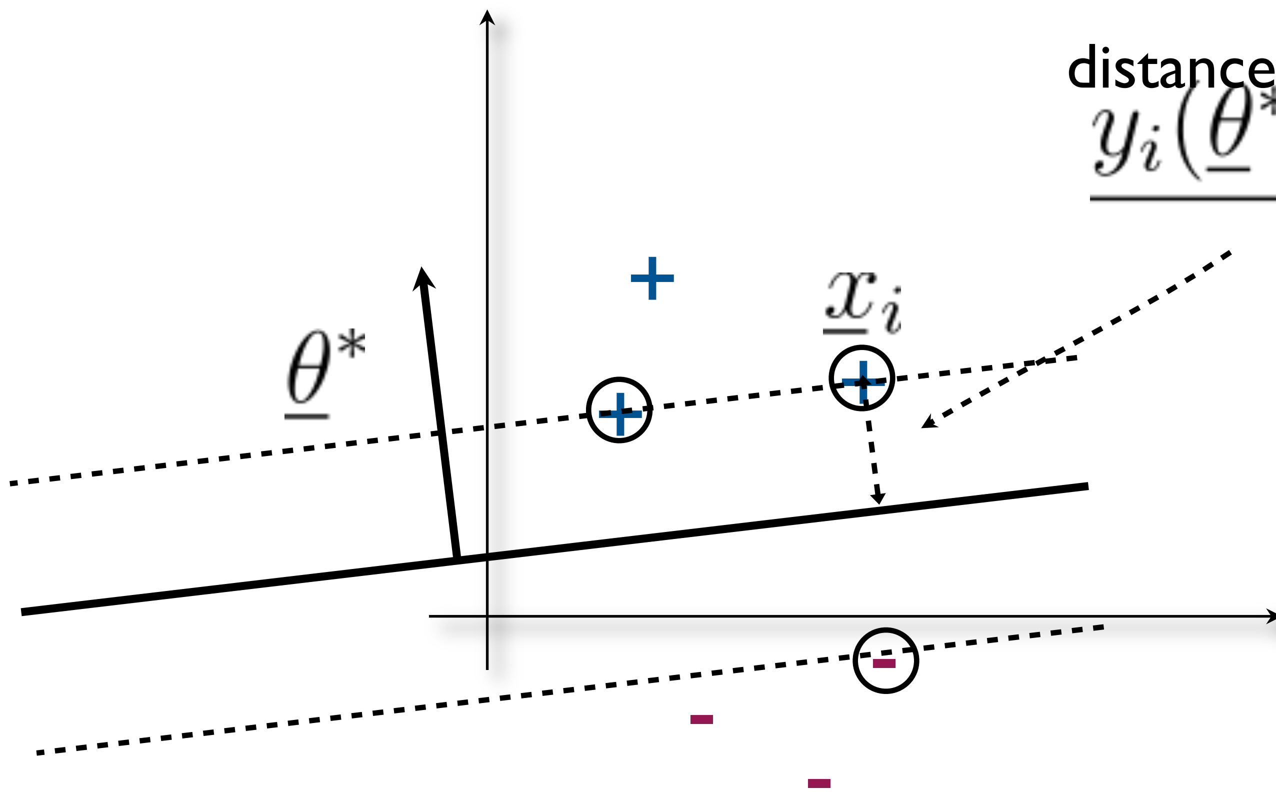
# Linear classifiers (with offset)

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

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



# Support vector machine




distance to the boundary

$$\frac{y_i(\underline{\theta}^* \cdot \underline{x}_i + \theta_0^*)}{\|\underline{\theta}^*\|} \geq \gamma_g$$

geometric margin

$$\gamma_g = \frac{1}{\|\underline{\theta}^*\|}$$

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

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

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

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

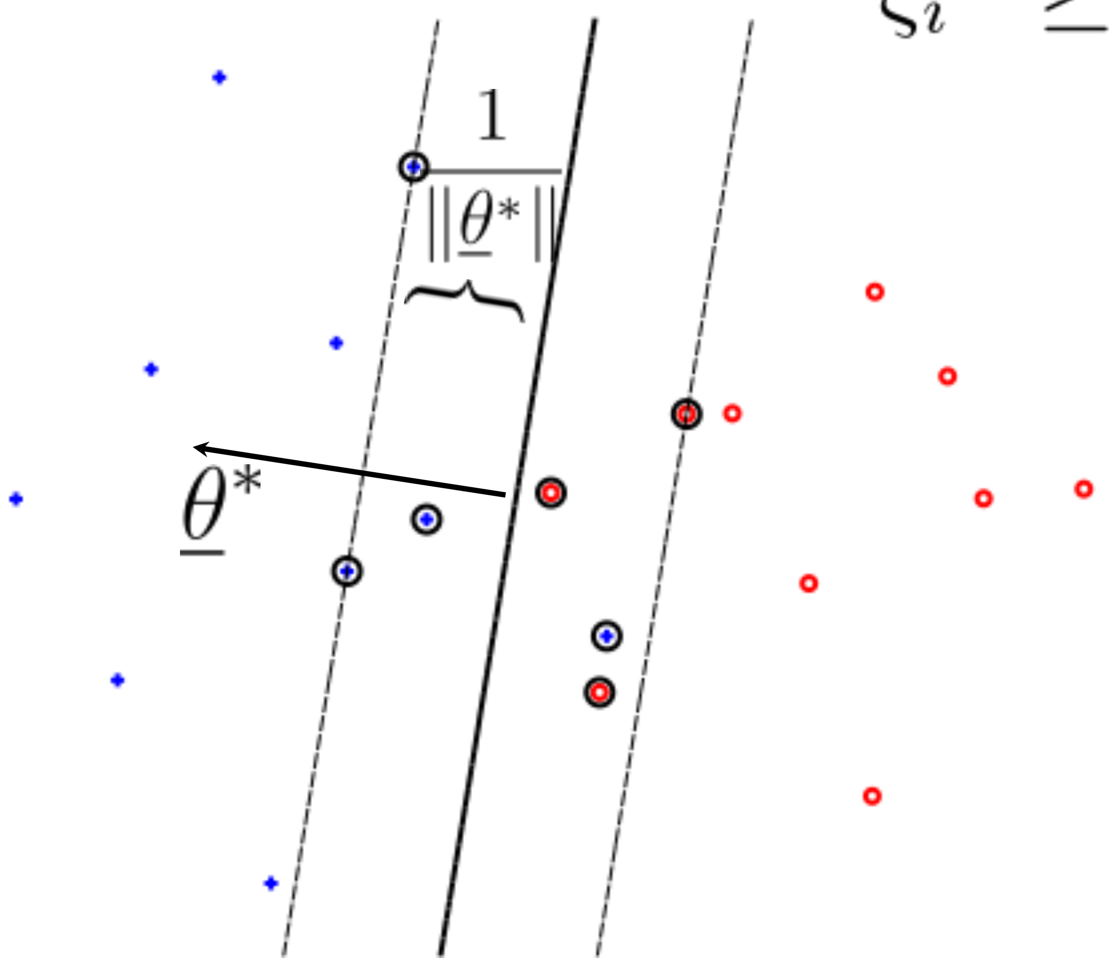
# Support vector machine

- Relaxed quadratic optimization problem

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

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

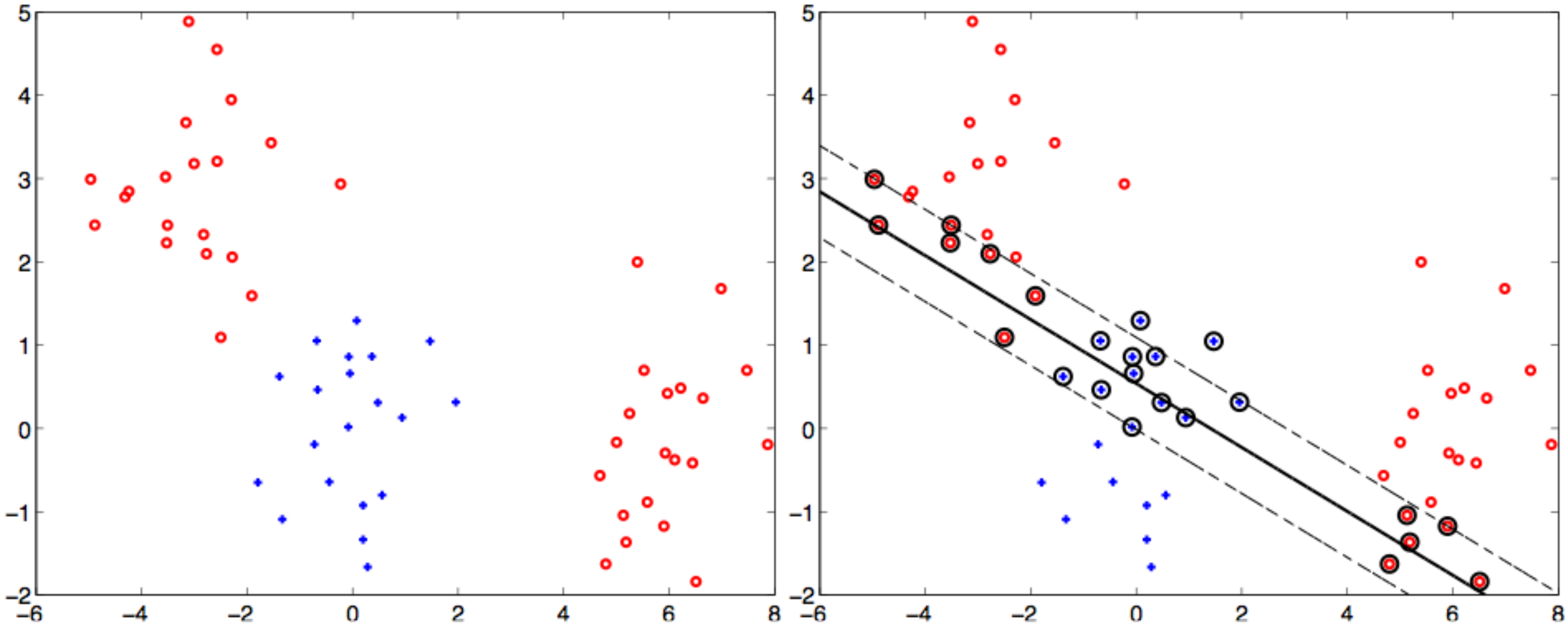
$$\xi_i \geq 0, \quad i=1,\ldots,n$$




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

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

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

linear classifier

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

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

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

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

linear classifier

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

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

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

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

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

linear classifier

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

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

linear decision boundary

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

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

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

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

linear classifier

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

non-linear classifier

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

# 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

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

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

linear classifier

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

non-linear classifier

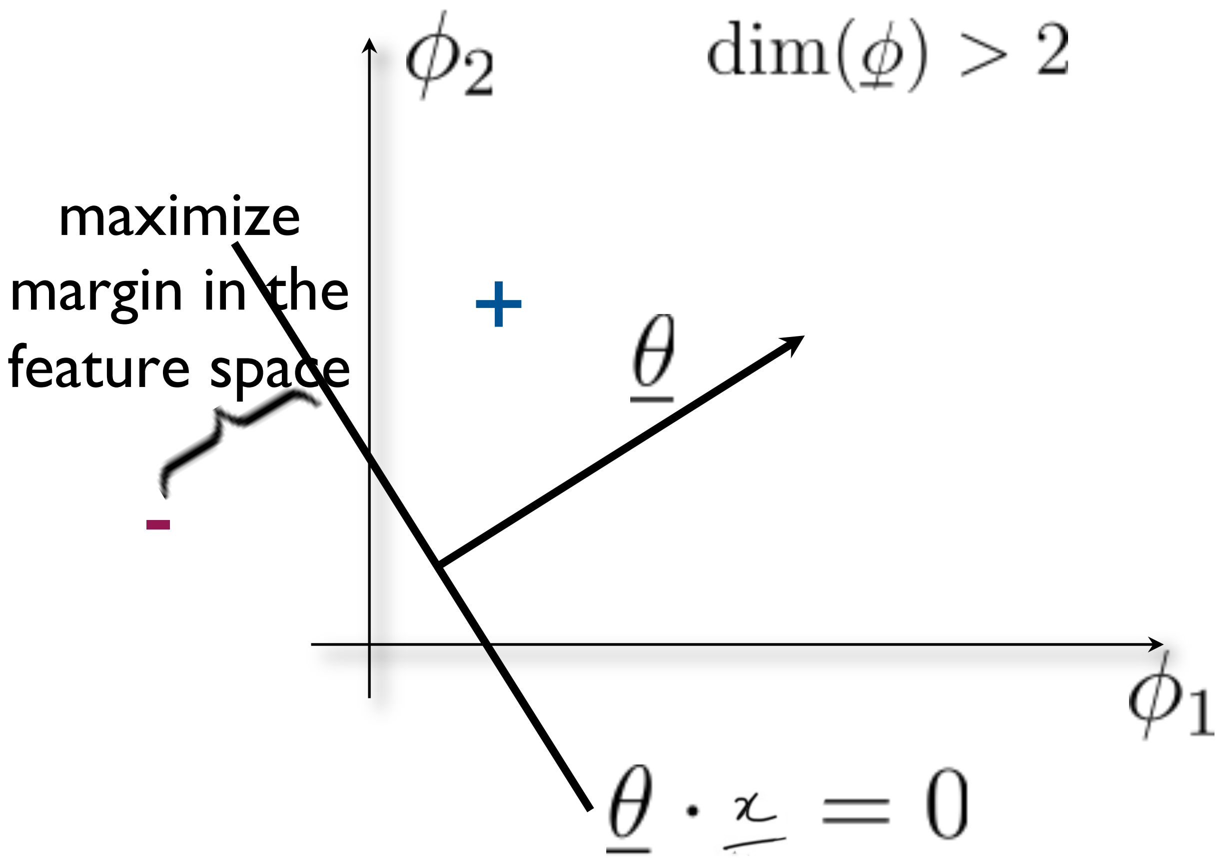
$$\underline{\theta} \cdot \underline{\phi}(\underline{x}) + \theta_0 = 0$$

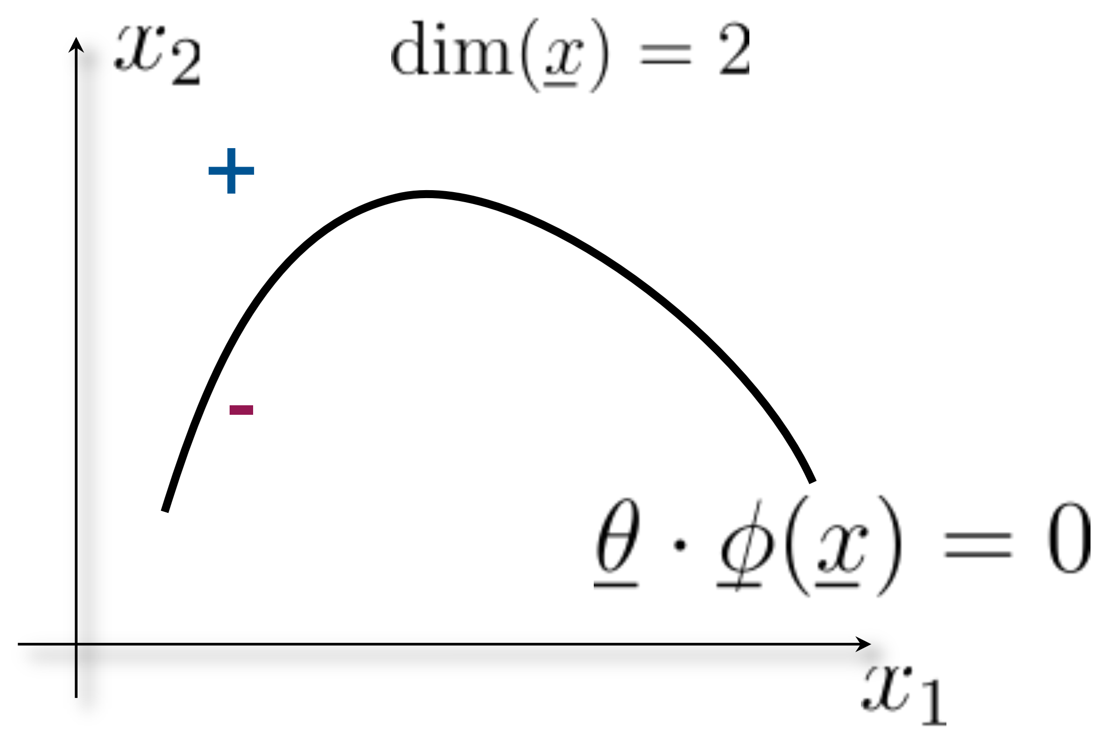
$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 \sqrt{2} x_1 x_2 + \theta_5 x_2^2 + \theta_0 = 0$$

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




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

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

# Learning non-linear classifiers

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

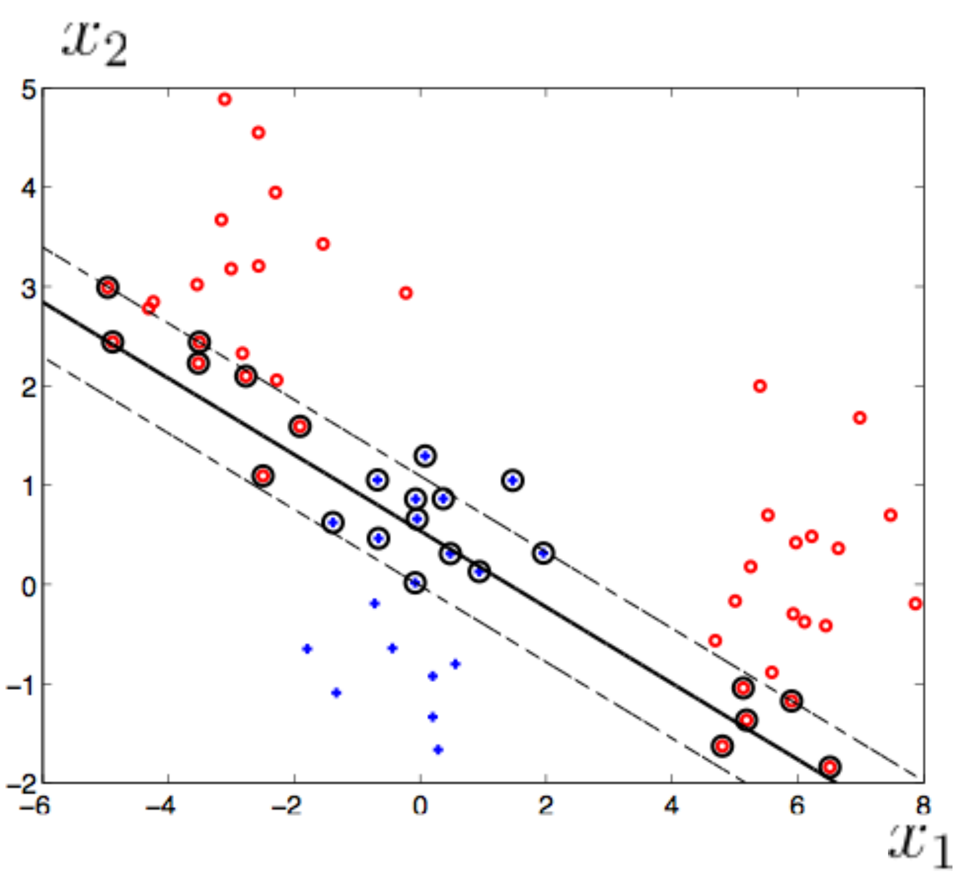
$$\text{minimize} \quad \frac{1}{2}\|\underline{\theta}\|^2 \quad + \quad C\sum_{i=1}^{n}\xi_i \quad \text{subject to}$$

$$\begin{aligned} y_i(\underline{\theta} \cdot \underline{\phi}(\underline{x}_i) + \theta_0) &\geq 1 - \xi_i, \quad i = 1, \ldots, n \\ \xi_i &\geq 0, \quad i = 1, \ldots, n \end{aligned}$$

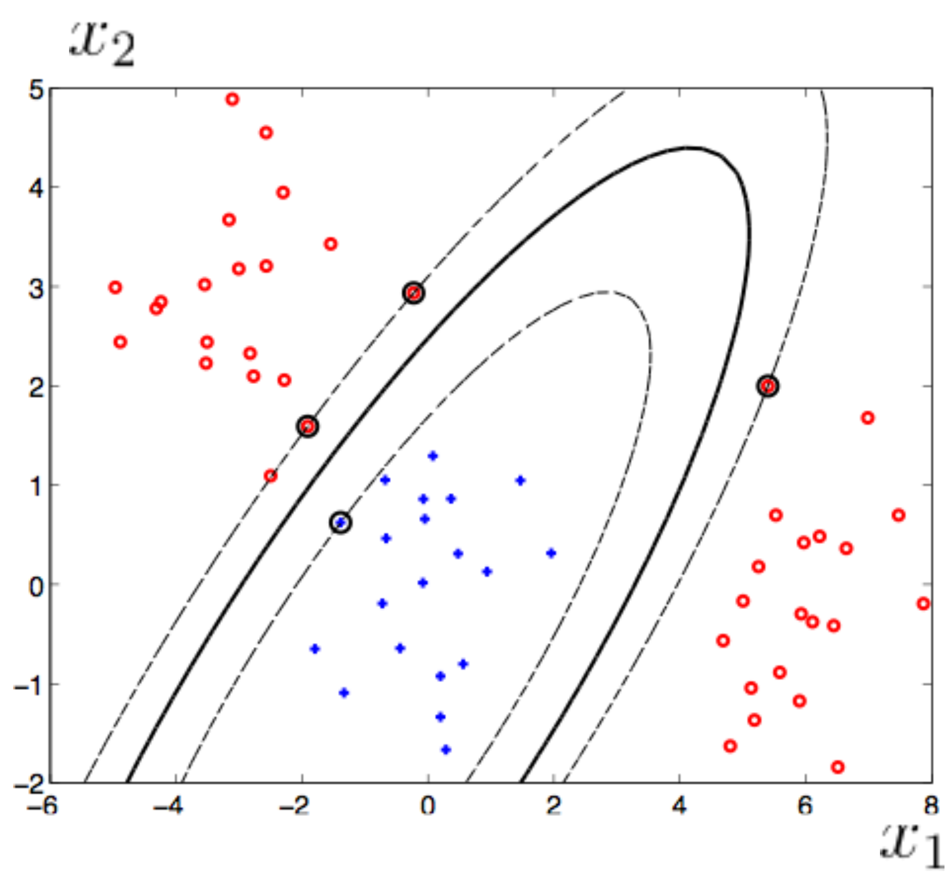
- Note that the cost of solving this quadratic programming problem increases with the dimension of the feature vectors (we will avoid this issues by solving the dual instead)

# 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



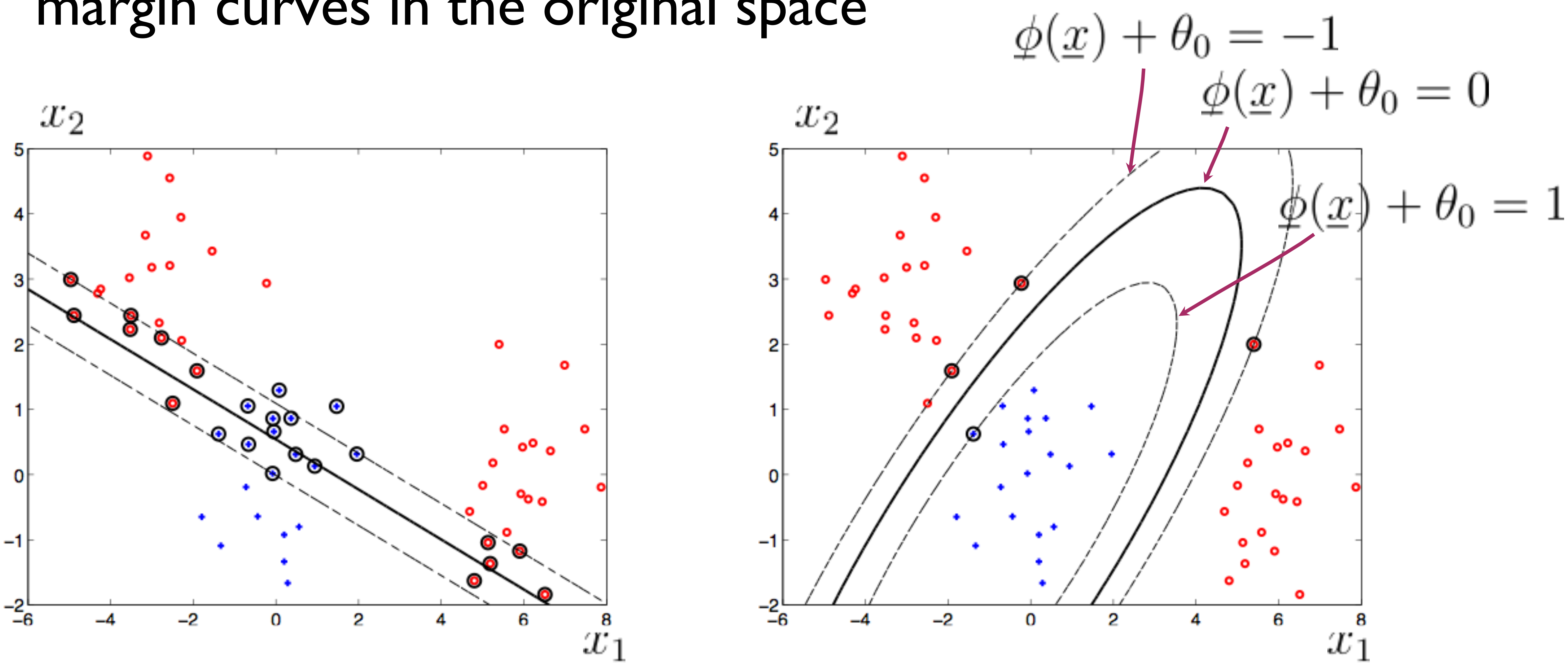

linear features




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



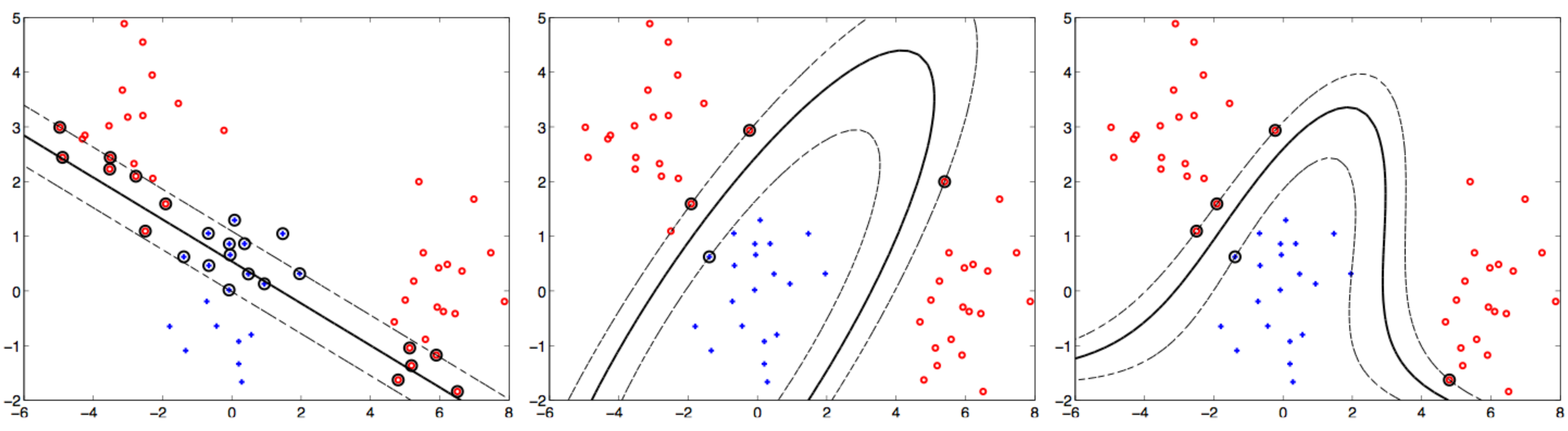

linear features

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"
- For simplicity, we drop the offset parameter

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

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

$$\text{For } t = 1, 2, \ldots$$

(applied in a sequence or repeatedly over a fixed training set)

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

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

# 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

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

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

# Kernel perceptron

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

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

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

Repeat until convergence:

for $t = 1,\ldots,n$

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

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

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

Repeat until convergence:

for $t = 1,\ldots,n$

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

$\alpha_t \leftarrow \alpha_t + 1$

# 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

$$\begin{aligned}
K(\underline{x}, \underline{x}') &= (\underline{x} \cdot \underline{x}') \\
K(\underline{x}, \underline{x}') &= (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2 \\
K(\underline{x}, \underline{x}') &= (\underline{x} \cdot \underline{x}') + (\underline{x} \cdot \underline{x}')^2 + (\underline{x} \cdot \underline{x}')^3 \\
K(\underline{x}, \underline{x}') &= (1 + \underline{x} \cdot \underline{x}')^p, \quad p = 1, 2, \ldots
\end{aligned}$$

# 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

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

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

# Kernel perceptron cont'd

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

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

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

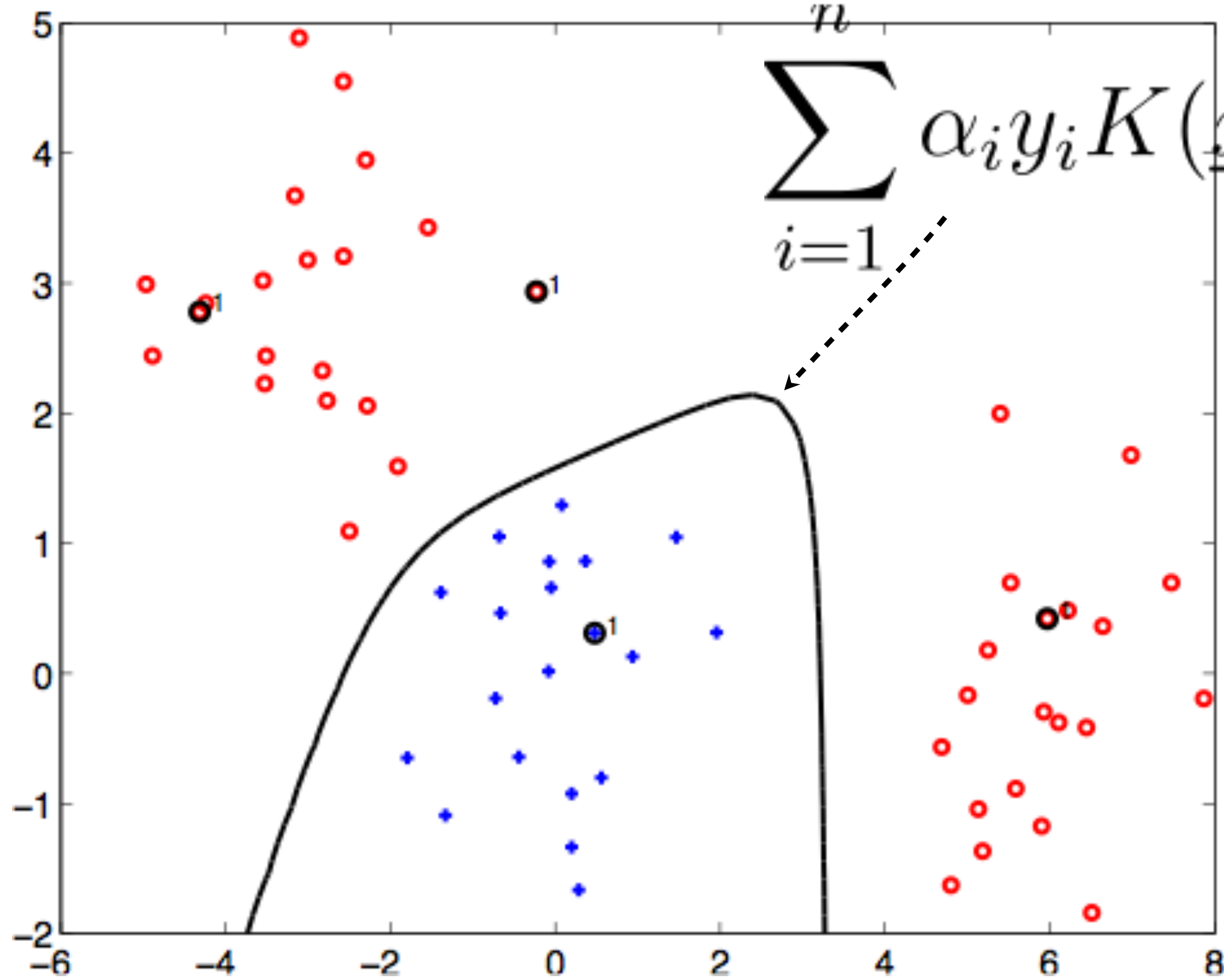
Repeat until convergence:

$$\begin{array}{l} \text{for } t = 1, \ldots, n \\ \quad \text{if } y_t\Big(\sum_{i=1}^{n} \alpha_i y_i K(\underline{x}_i, \underline{x}_t)\Big) \leq 0 \ \text{(mistake)} \\ \qquad \alpha_t \leftarrow \alpha_t + 1 \end{array}$$

# Kernel perceptron: example

- With a radial basis kernel

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



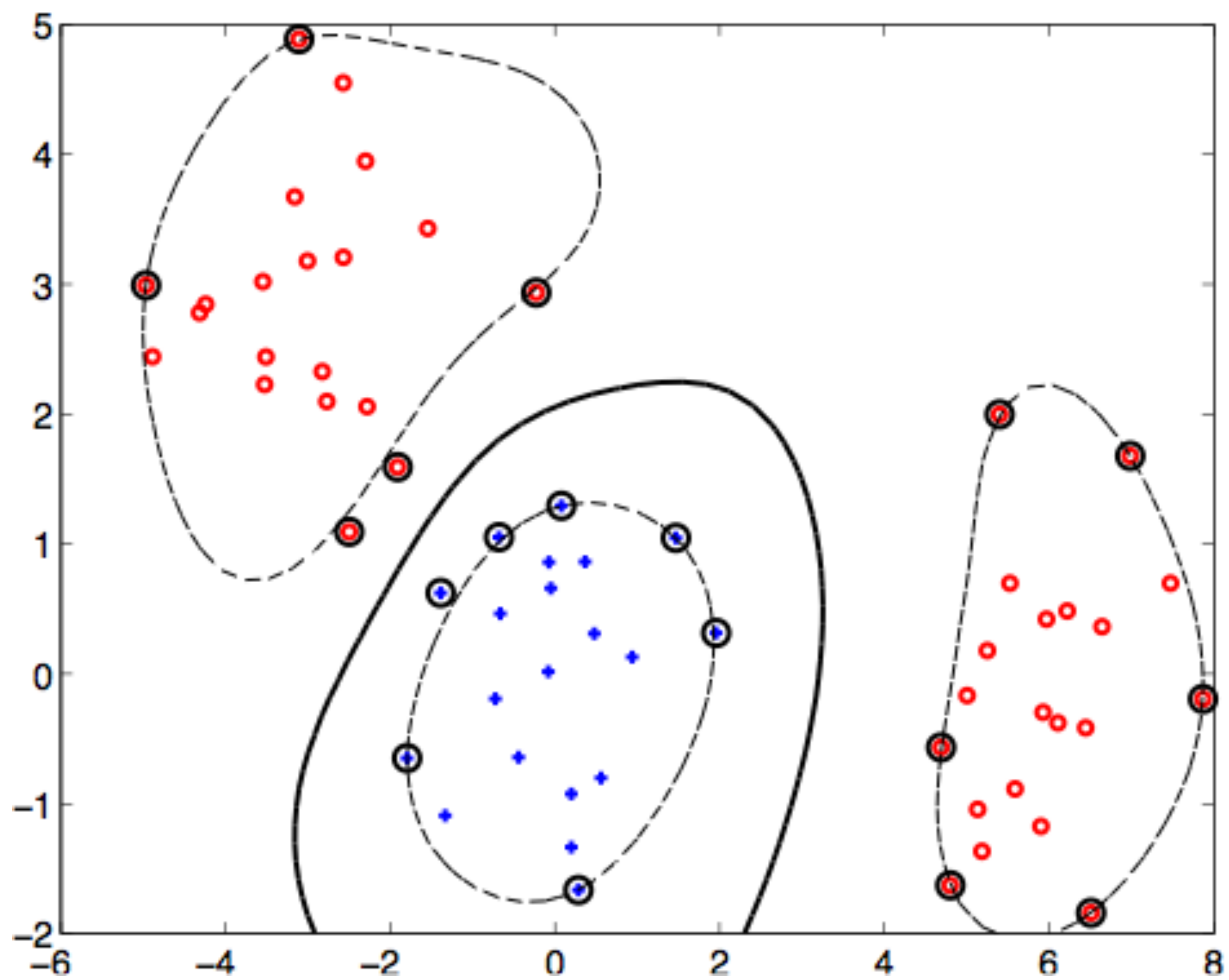

$$\sum_{i=1}^{n} \alpha_i y_i K(\underline{x}_i, \underline{x}) = 0$$

only four mistakes!

# 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

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



# Extra Slides

# 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

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

# 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

$$\begin{aligned} \exp\{-\beta\|\underline{x} - \underline{x}'\|^2\} &= \exp\{-\beta \underline{x}\cdot\underline{x} + 2\beta \underline{x}\cdot\underline{x}' - \beta \underline{x}'\cdot\underline{x}'\} \\ &= \overbrace{\exp\{-\beta \underline{x}\cdot\underline{x}\}}^{f(\underline{x})} \exp\{2\beta \underline{x}\cdot\underline{x}'\} \overbrace{\exp\{-\beta \underline{x}'\cdot\underline{x}'\}}^{f(\underline{x}')} \\ &= f(\underline{x}) \underbrace{\left(1 + 2\beta(\underline{x}\cdot\underline{x}') + \ldots\right)} f(\underline{x}') \end{aligned}$$

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

← **Infinite Taylor series expansion**

# Kernels

- By writing the algorithm in a "kernel" form, we can try to work with the kernel (inner product) directly rather than explicating the high dimensional feature vectors

$$\begin{aligned} K(\underline{x}, \underline{x}') &= \underline{\phi}(\underline{x}) \cdot \underline{\phi}(\underline{x}') \\ &= \begin{bmatrix} \\ ? \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \\ ? \\ \\ \end{bmatrix} \\ &= \exp(-\|\underline{x} - \underline{x}'\|^2) \end{aligned}$$

(e.g.)

- All we need to ensure is that the kernel is "valid", i.e., there exists some underlying feature representation

# Valid kernels

- A kernel function is valid (is a kernel) if there exists some feature mapping such that

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

- Equivalently, a kernel is valid if it is symmetric and for all training sets, the Gram matrix

$$\begin{bmatrix} K(\underline{x}_1, \underline{x}_1) & \cdots & K(\underline{x}_1, \underline{x}_n) \\ \cdots & \cdots & \cdots \\ K(\underline{x}_n, \underline{x}_1) & \cdots & K(\underline{x}_n, \underline{x}_n) \end{bmatrix}$$

is positive semi-definite