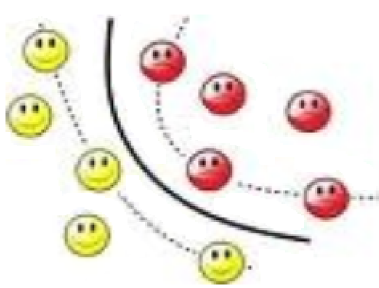
CIS 606 Machine Learning
Spring 2013
Lecture 10
Classification III

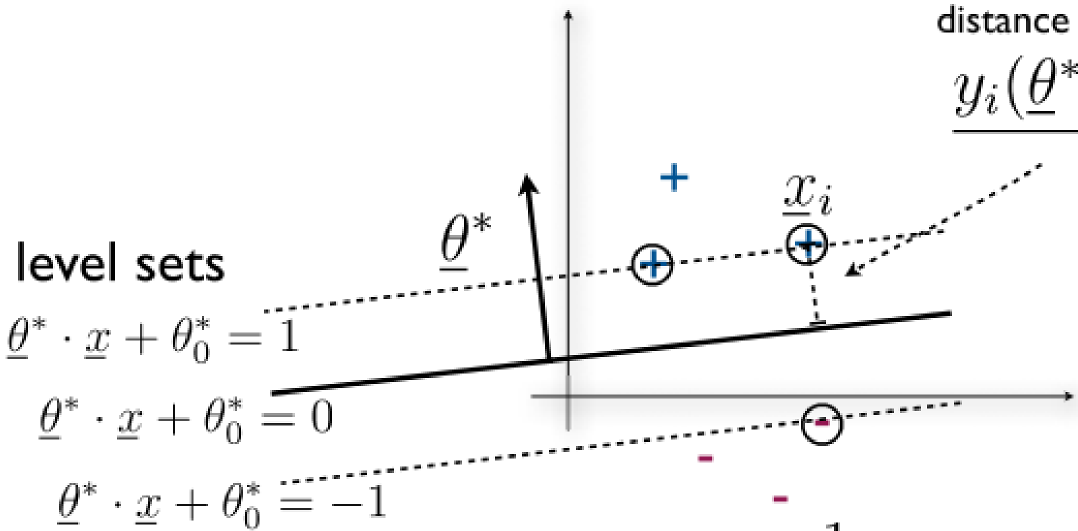
Wei Lee Woon and Zeyar Aung



Today's topics

- Review of support vector machines...
- Case for more powerful classifiers
- Feature mappings, non-linear classifiers, kernels
  - non-linear feature mappings
  - kernels, kernel perceptron

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

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

$$\underset{\underline{\theta}, \theta_0}{\text{minimize}} \; \frac{1}{2}\|\underline{\theta}\|^2 \text{ subject to}$$

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

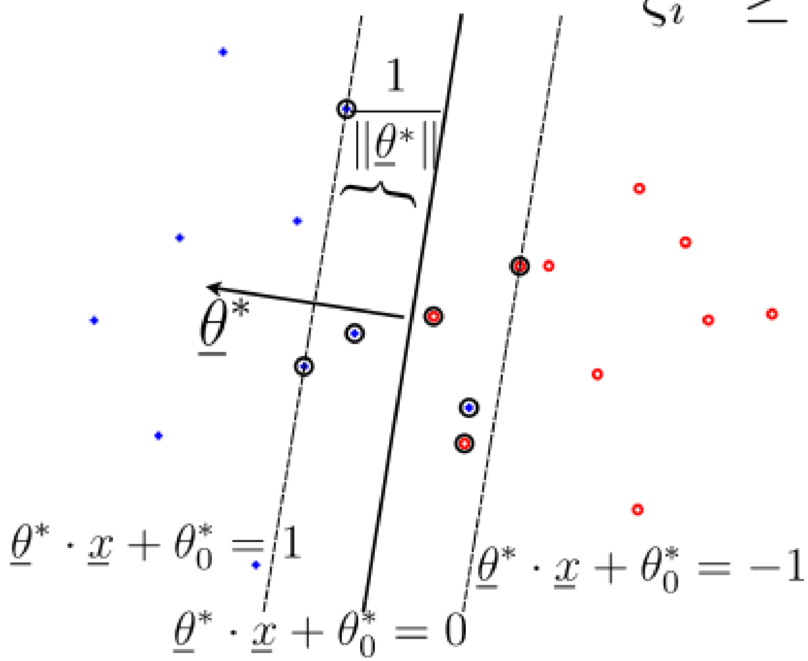
- We get a max-margin decision boundary by solving a quadratic programming problem
- The solution is unique and sparse in terms of active constraints (support vectors)

Support vector machine and slack

- Relaxed quadratic programming problem

$$\underset{\underline{\theta}, \theta_0, \xi}{\text{minimize}} \quad \frac{1}{2}\|\underline{\theta}\|^2 \quad + \quad C\sum_{i=1}^{n} \xi_i \quad \text{subject to}$$

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



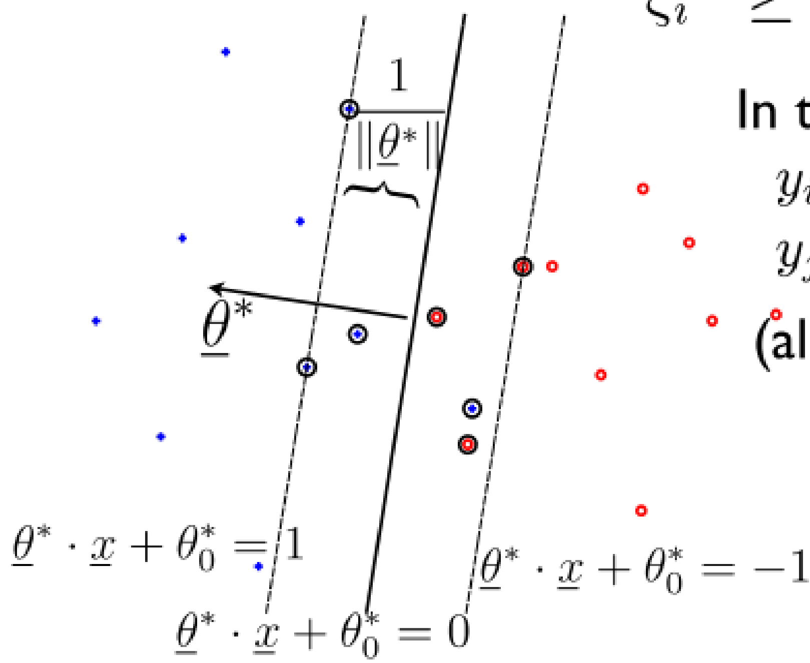
Some of the points may now violate the margin constraints (positive slack) or even be misclassified

Support vector machine and slack

- Relaxed quadratic programming problem

$$\underset{\underline{\theta}, \theta_0, \xi}{\text{minimize}} \quad \frac{1}{2}\|\underline{\theta}\|^2 \quad + \quad C\sum_{i=1}^{n}\xi_i \quad \text{subject to}$$

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



In the solution, we will have either

$$\begin{aligned} y_i(\underline{\theta}^* \cdot \underline{x}_i + \theta_0^*) \quad &= 1 - \xi_i^*, \quad \xi_i^* \geq 0 \\ y_j(\underline{\theta}^* \cdot \underline{x}_j + \theta_0^*) \quad &> 1, \qquad\quad \xi_j^* = 0 \end{aligned}$$

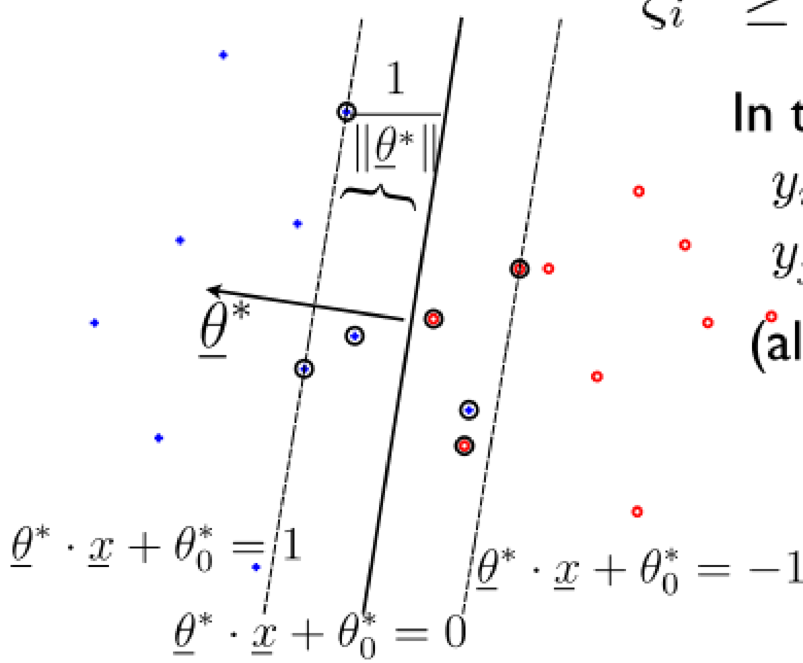
(all the active constraints are SVs)

Support vector machine and slack

- Relaxed quadratic programming problem

$$\underset{\underline{\theta}, \theta_0, \xi}{\text{minimize}} \quad \frac{1}{2}\|\underline{\theta}\|^2 \quad + \quad C\sum_{i=1}^{n} \xi_i \quad \text{subject to}$$

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



In the solution, we will have either

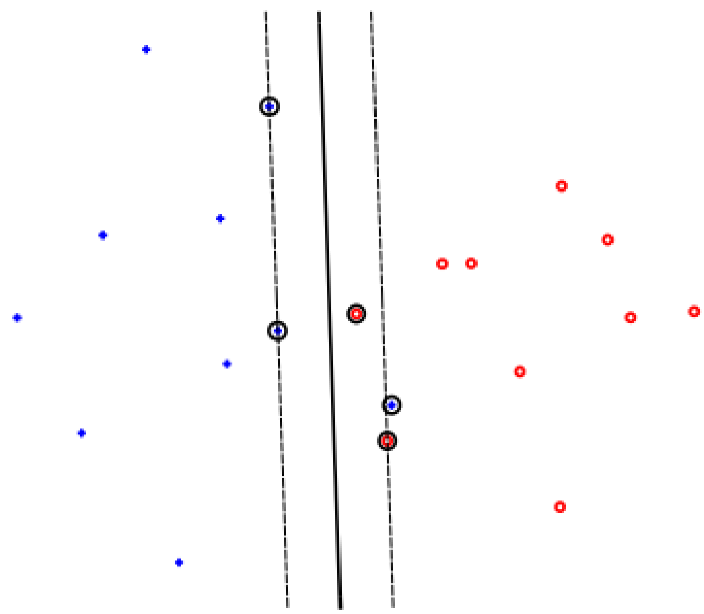
$$\begin{aligned} y_i(\underline{\theta}^* \cdot \underline{x}_i + \theta_0^*) \quad &= 1 - \xi_i^*, \quad \xi_i^* \geq 0 \\ y_j(\underline{\theta}^* \cdot \underline{x}_j + \theta_0^*) \quad &> 1, \qquad\quad \xi_j^* = 0 \end{aligned}$$

(all the active constraints are SVs)

The solution need not be unique in terms of θ_0, ξ

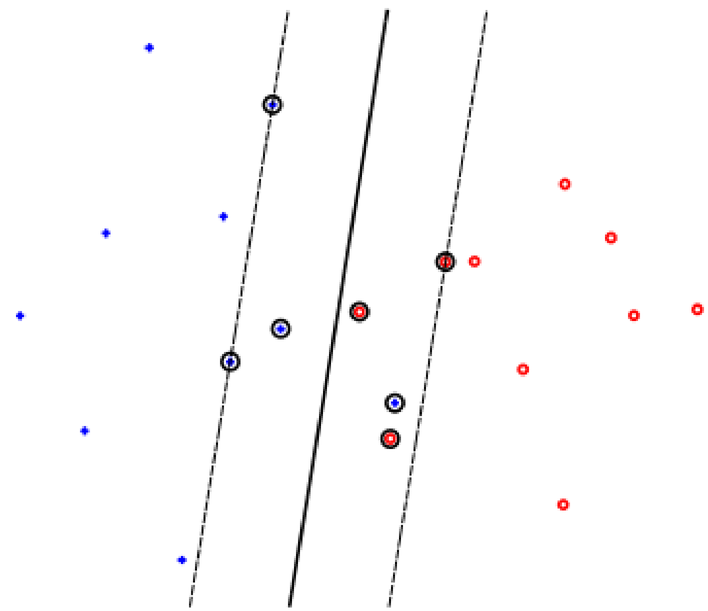
Examples

- C=10



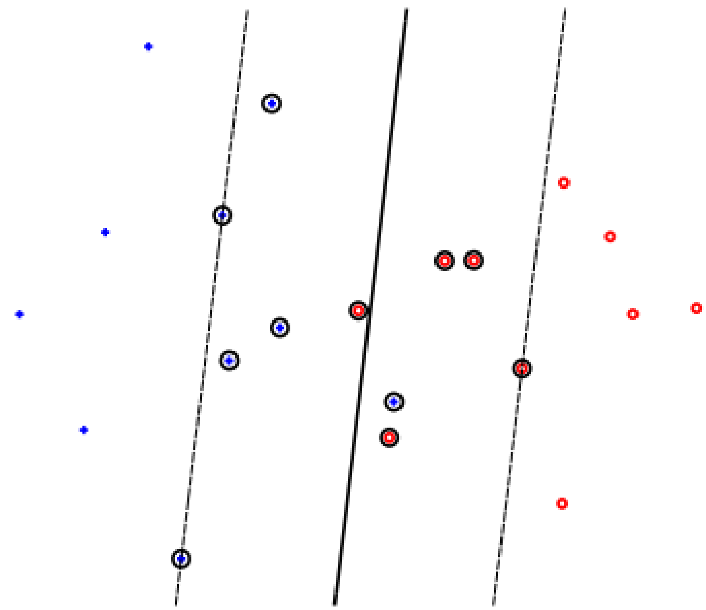
Examples

- C=1



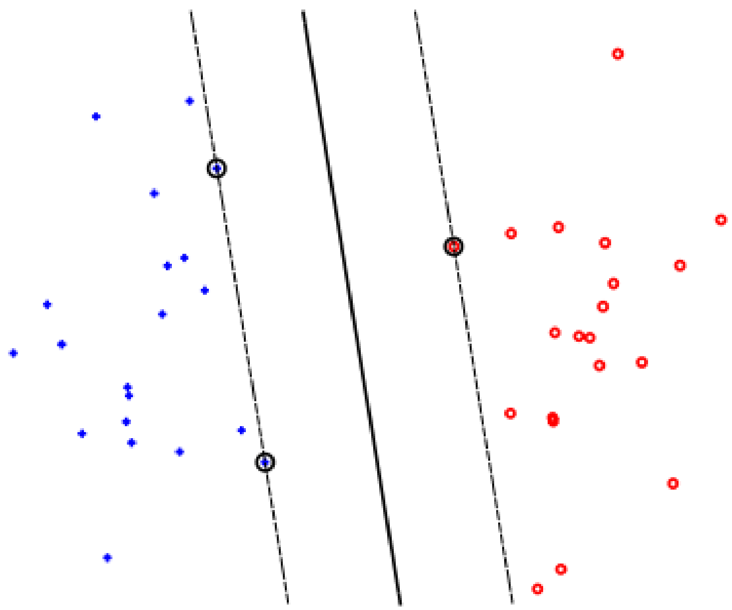
Examples

- C=0.1



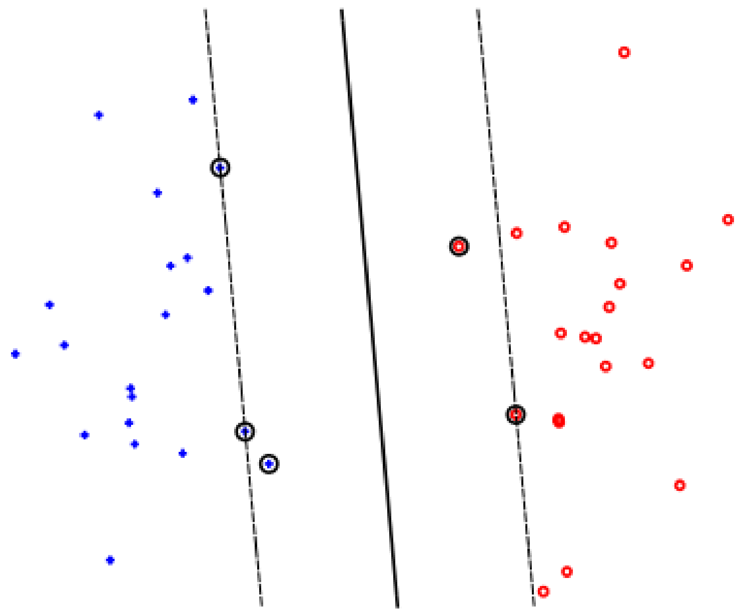
Examples

- C potentially affects the solution even in the separable case
- C = 1



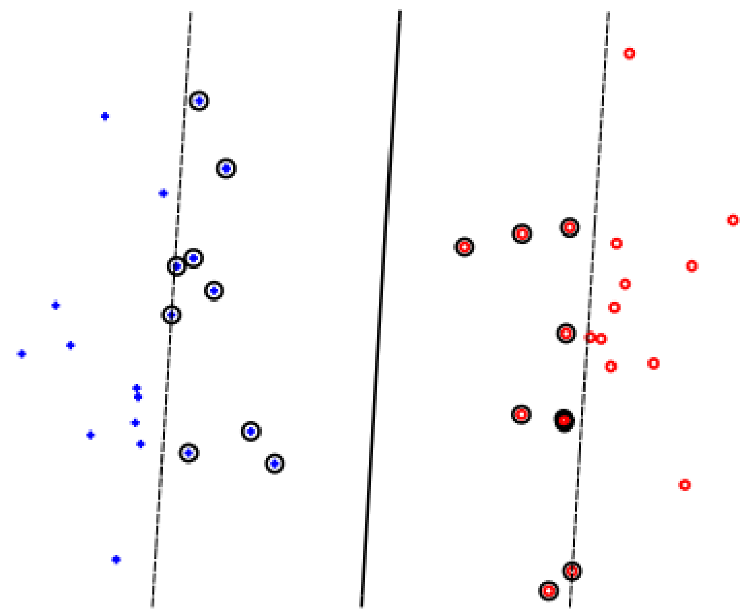
Examples

- C potentially affects the solution even in the separable case
- C = 0.1



Examples

- C potentially affects the solution even in the separable case
- C = 0.01

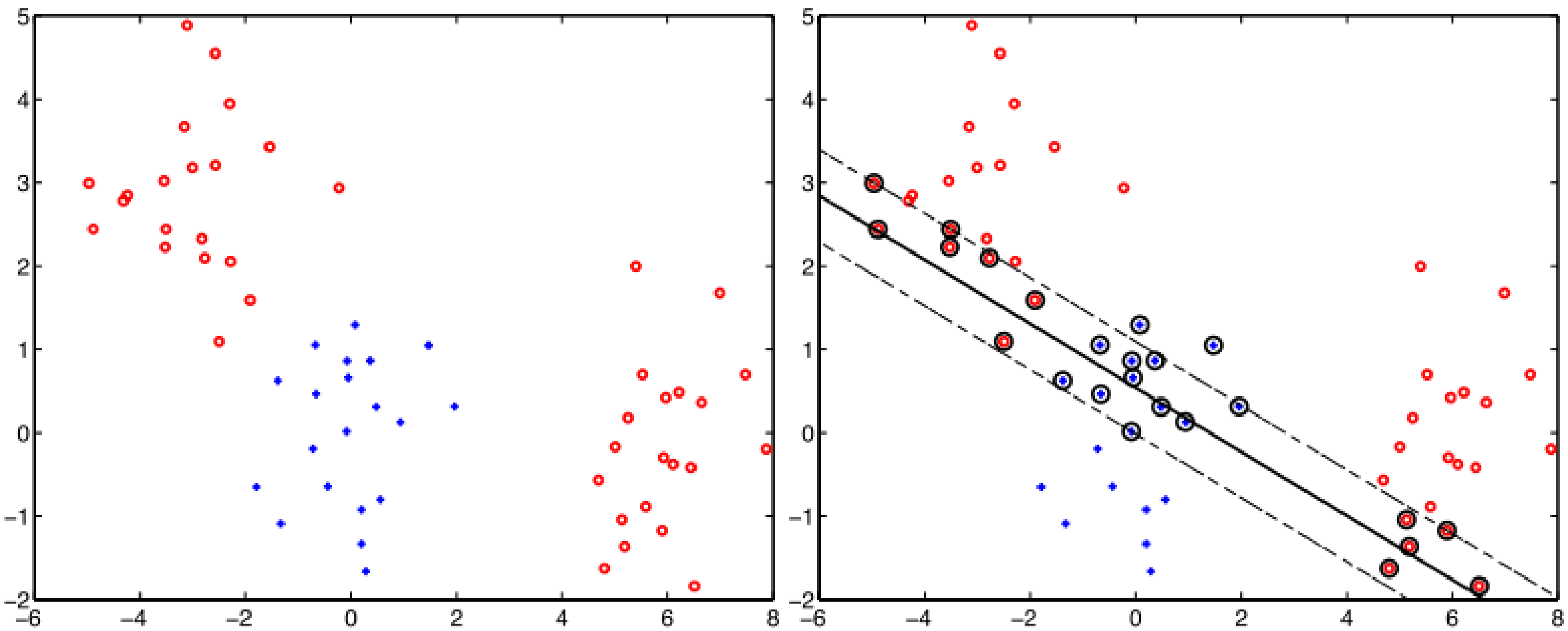


Today's topics

- Review of support vector machines...
- Case for more powerful classifiers
- Feature mappings, non-linear classifiers, kernels
  - non-linear feature mappings
  - kernels, kernel perceptron

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

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

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

linear classifier

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

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

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

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

linear classifier

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

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

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

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

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

linear classifier

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

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

linear decision boundary

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

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

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

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

linear classifier

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

non-linear classifier

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

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

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

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

linear classifier

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

non-linear classifier

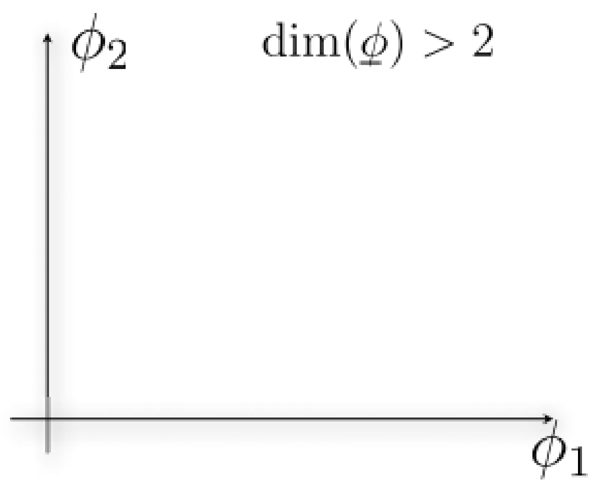
$$\underline{\theta} \cdot \underline{\phi}(\underline{x}) + \theta_0 = 0$$

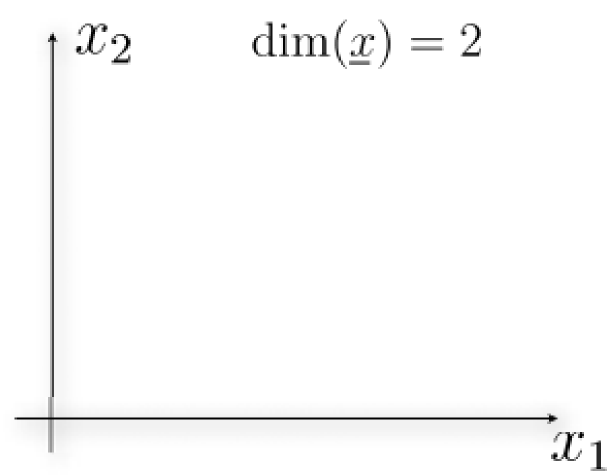
$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 \sqrt{2} x_1 x_2 + \theta_5 x_2^2 + \theta_0 = 0$$

non-linear decision boundary

Non-linear feature mappings

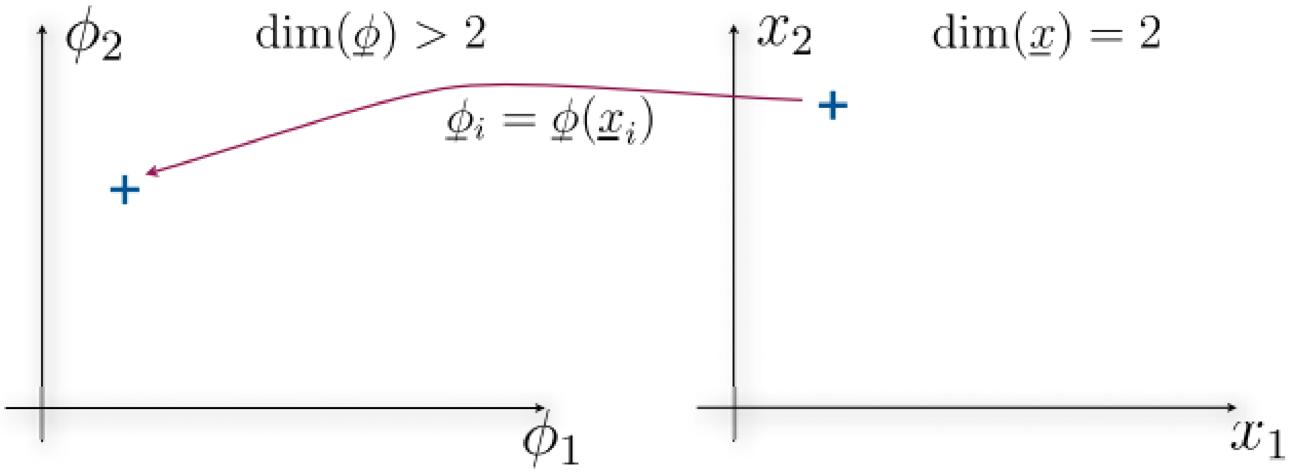
- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates





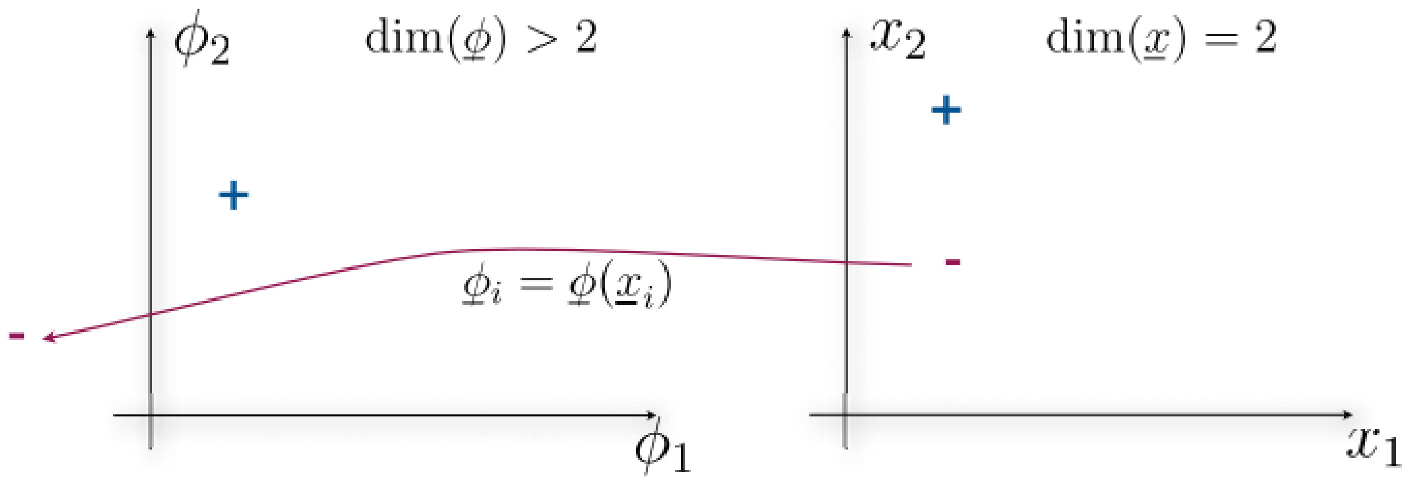
Non-linear feature mappings

- 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



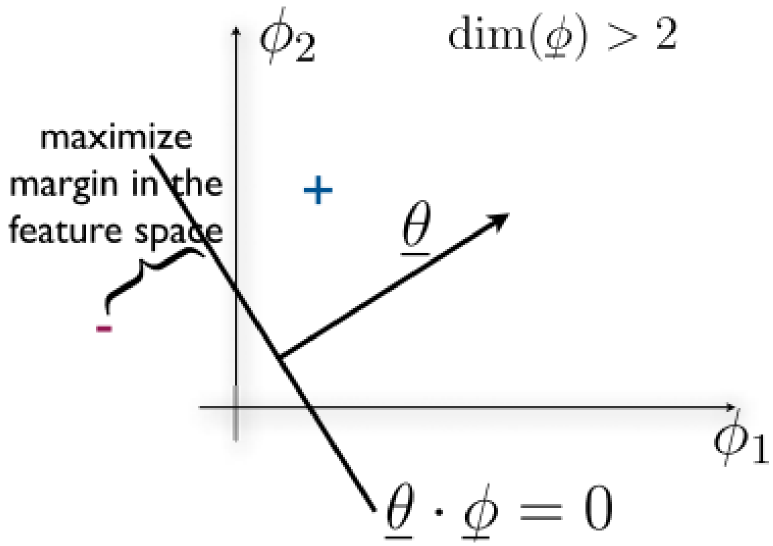
Non-linear feature mappings

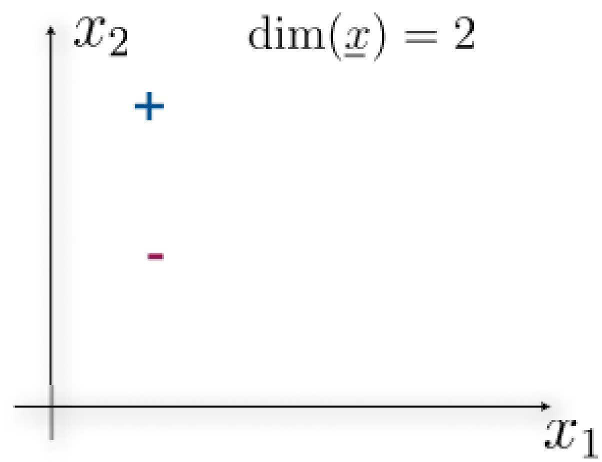
- By expanding the feature coordinates, we still have a linear classifier in the new feature coordinates but a non-linear classifier in the original coordinates



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

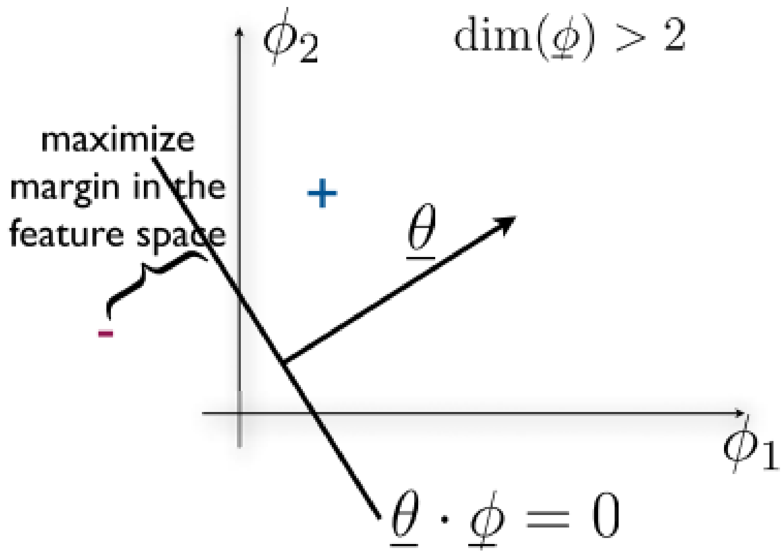




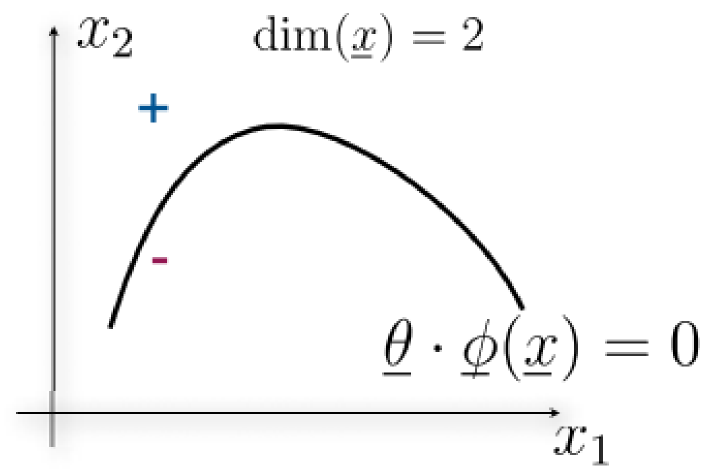
$$f(\underline{\phi}; \underline{\theta}, \theta_0) = \text{sign}\big(\underline{\theta} \cdot \underline{\phi} + \theta_0\big)$$

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



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



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

Learning non-linear classifiers

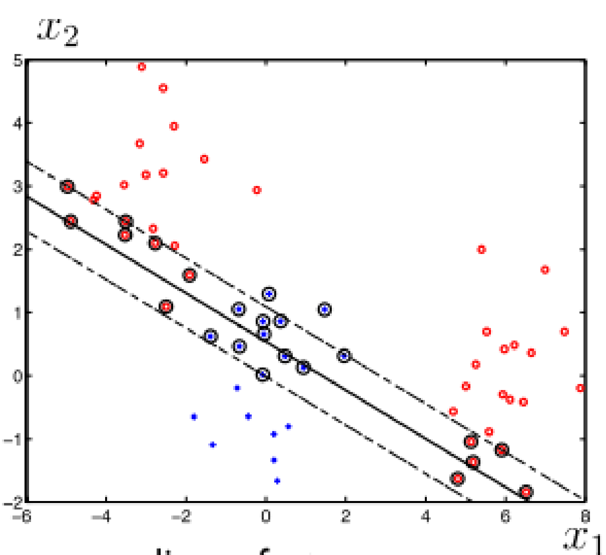
- We can apply the same SVM formulation, just replacing the input examples with (higher dimensional) feature vectors

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

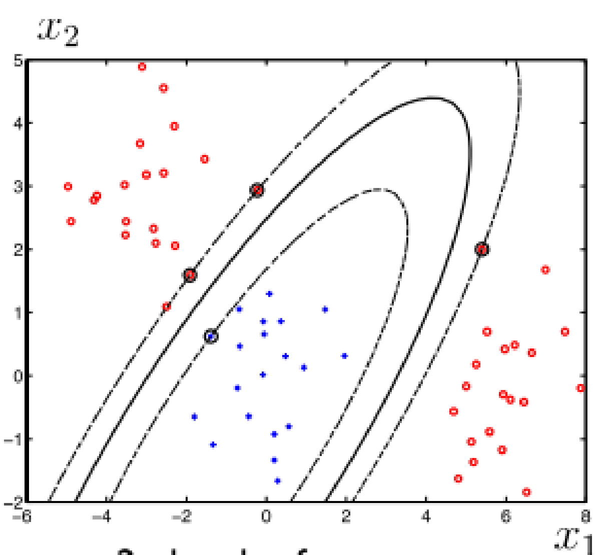
- Note that the cost of solving this quadratic programming problem increases with the dimension of the feature vectors (we will avoid this issues by solving the dual instead)

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



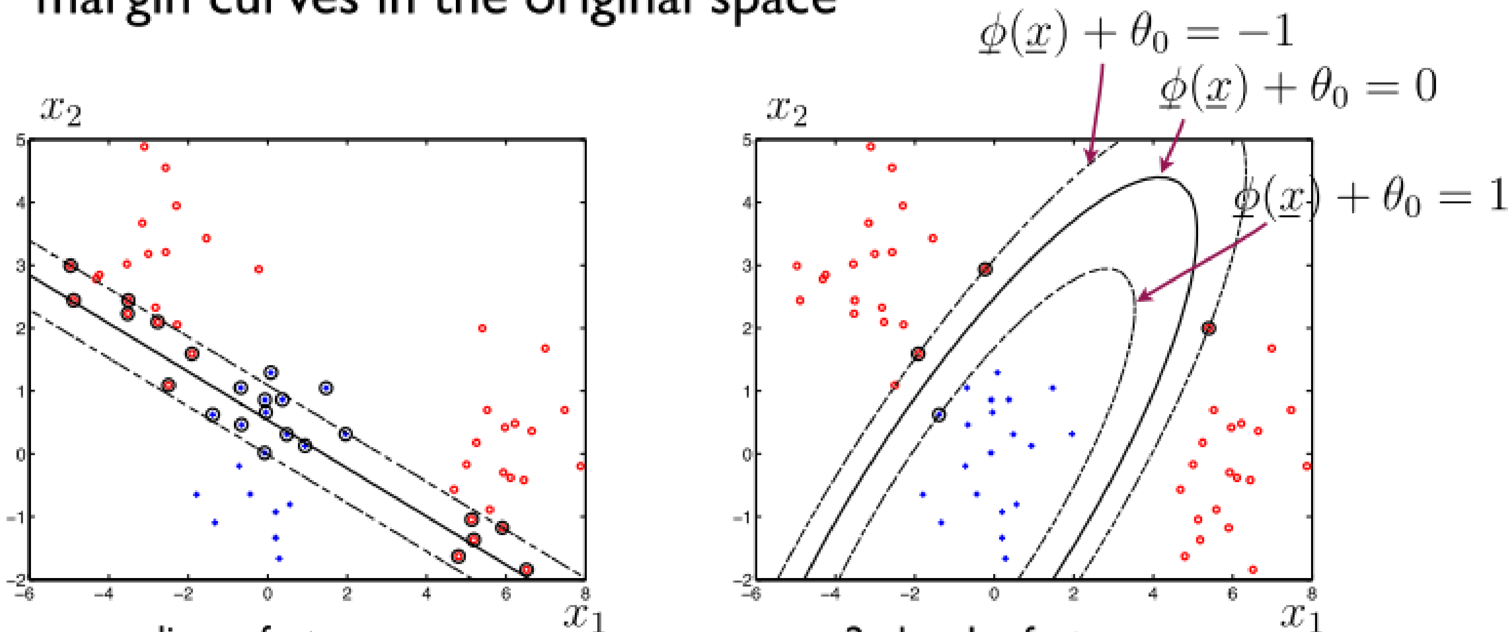
linear features



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

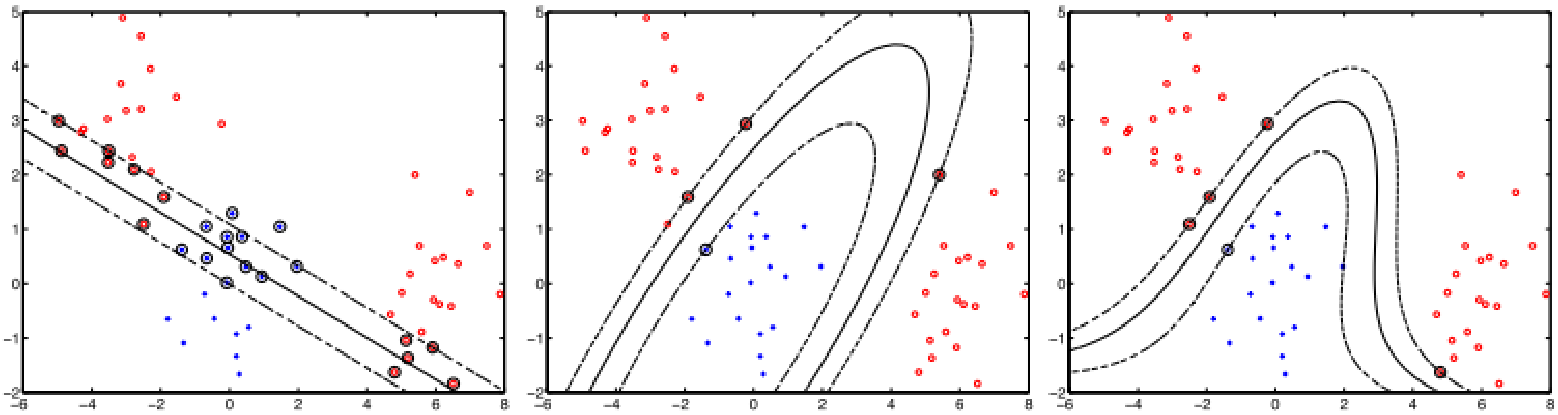


linear features   2nd order features

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 $y_t\underline{\phi}(\underline{x}_t)$ 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

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

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

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 for $t = 1,\ldots,n$

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

$$\alpha_t \leftarrow \alpha_t + 1$$

Kernel perceptron

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

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

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

$$\text{Repeat for } t = 1,\ldots,n$$

$$\quad \text{if } y_t\Big(\textstyle\sum_{i=1}^{n}\alpha_i y_i[\underline{\phi}(\underline{x}_i)\cdot\underline{\phi}(\underline{x}_t)]\Big) \le 0 \text{ (mistake)}$$

$$\qquad \alpha_t \leftarrow \alpha_t + 1$$

Why inner products?

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

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

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

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

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

Why inner products?

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

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

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

- What is $\underline{\phi}(\underline{x})$?

Why inner products?

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

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

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

- What is $\underline{\phi}(\underline{x})$ 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

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

- Examples of polynomial kernels

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

Composition rules for kernels

- We can construct valid kernels from simple components
- For any function $f : R^d \rightarrow R$, if K1 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 K1 and K2 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

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

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

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!
- We can use the composition rules to show that this is indeed a valid kernel

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

Kernel perceptron cont'd

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

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

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

Repeat for $t = 1, \ldots, n$

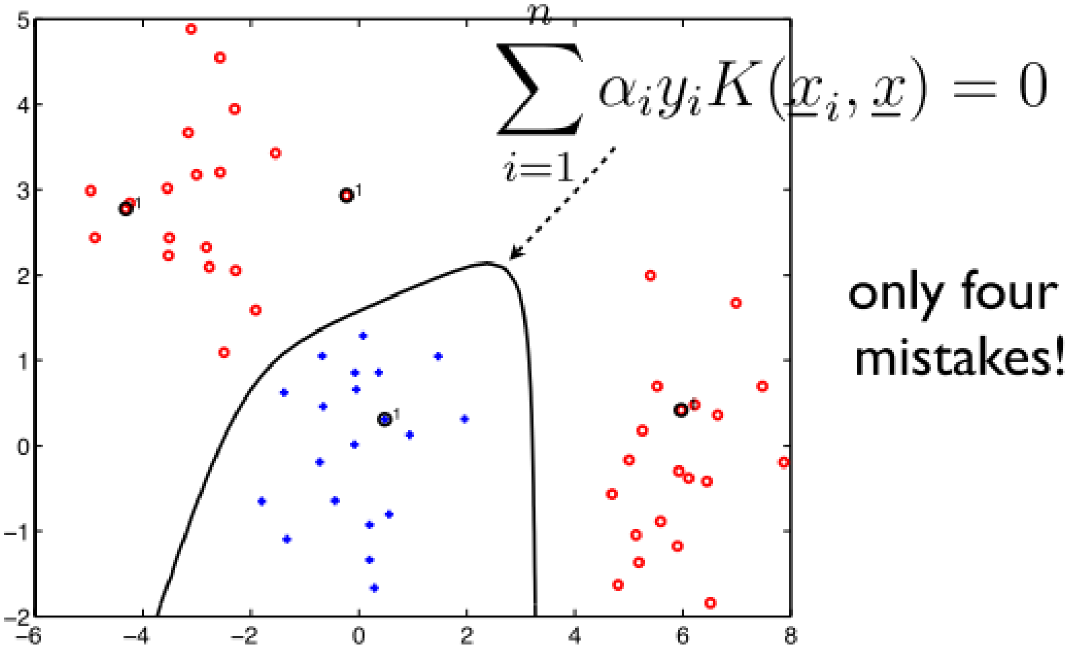
$\quad$ if $y_t\big(\sum_{i=1}^{n} \alpha_i y_i K(\underline{x}_i, \underline{x}_t)\big) \leq 0$ (mistake)

$\quad\quad \alpha_t \leftarrow \alpha_t + 1$

Kernel perceptron: example

- With a radial basis kernel

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

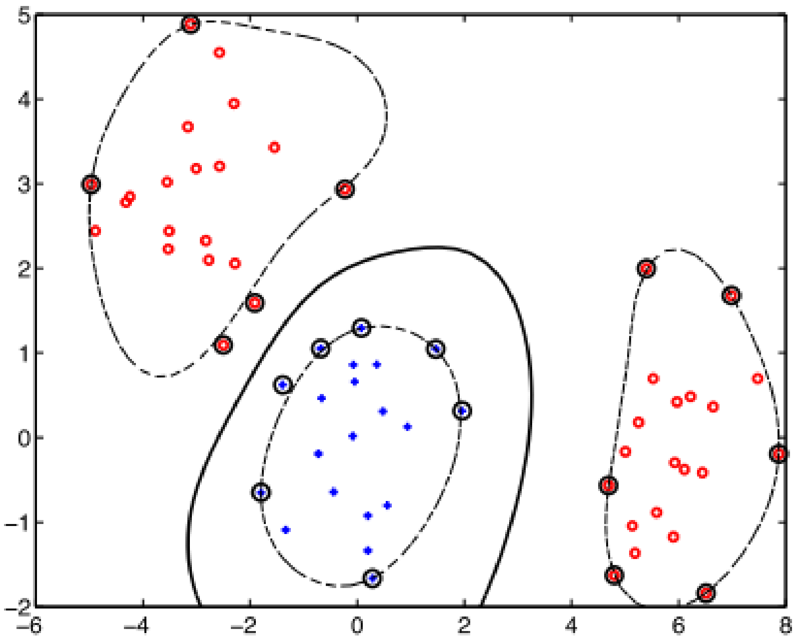


only four mistakes!

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

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



Original source: MIT Course 6.867 Machine Learning (Fall 2010) by Prof. Tommi Jaakkola