# Lecture 21

# Support Vector Machines II

# Slack variables

What if data is not linearly separable??

$$\min \quad \left[ \frac{1}{2} ||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i \right]$$

subject to constraints (for all $i$):

$y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i$

$\xi_i \geq 0$

example lies on wrong side of hyperplane: corresponding $\xi_i \geq 1$
so $\sum_i \xi_i$ is upper bound on number of training errors

$\lambda$ trades off training error versus model complexity

this is known as the *soft-margin* extension

# Non-linear decision boundaries

note that both the quadratic programming problem and final decision function

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{sign}(\mathbf{x} \cdot \mathbf{w}) \\ &= \mathbf{sign}(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i)) \end{aligned}$$

depend only on dot products between patterns

how to form non-linear decision boundaries in input space?

basic idea:

1. map data into feature space: $\mathbf{x} \rightarrow \Phi(\mathbf{x})$
2. replace dot products between patterns: $\mathbf{x}_i \cdot \mathbf{x}_j \rightarrow \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$
3. find linear decision boundary in feature space

problem – what is good $\Phi()$?

# Kernel trick

*kernel trick*: dot-products in feature space can be computed as a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

idea: work directly on $\mathbf{x}$, avoid having to compute $\Phi(\mathbf{x})$ at all

example:

$$\begin{aligned}
K(\mathbf{a}, \mathbf{b}) &= (\mathbf{a} \cdot \mathbf{b})^3 = ((a_1, a_2) \cdot (b_1, b_2))^3 \\
&= (a_1 b_1 + a_2 b_2)^3 \\
&= a_1^3 b_1^3 + 3 a_1^2 b_1^2 a_2 b_2 + 3 a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\
&= ((a_1^3, \sqrt{3} a_1^2 a_2, \sqrt{3} a_1 a_2^2, a_2^3) \cdot (b_1^3, \sqrt{3} b_1^2 b_2, \sqrt{3} b_1 b_2^2, b_2^3)) \\
&= \Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})
\end{aligned}$$

# Kernels

examples:

1. polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^z$
2. Gaussian kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2)$
3. sigmoid kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa(\mathbf{x}_i \cdot \mathbf{x}_j) + a)$

each kernel computation corresponds to dot product calculation for particular mapping $\Phi(\mathbf{x})$ – implicitly maps to high-dim space

why useful?

- rewrite training examples using more complex features
- dataset not linearly separable in original space may be linearly separable in higher-dimensional space

# Classification with non-linear SVMs

non-linear SVM using kernel function $K()$:

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

minimize $L$ wrt $\{\alpha\}$, under constraints $\alpha_i \geq 0$

unlike linear SVM, cannot express $\mathbf{w}$ as linear combination of support vectors, now must retain the support vectors to classify new examples

final decision function

$$f(\mathbf{x}) = \mathbf{sign}(\sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i))$$

# Kernel functions

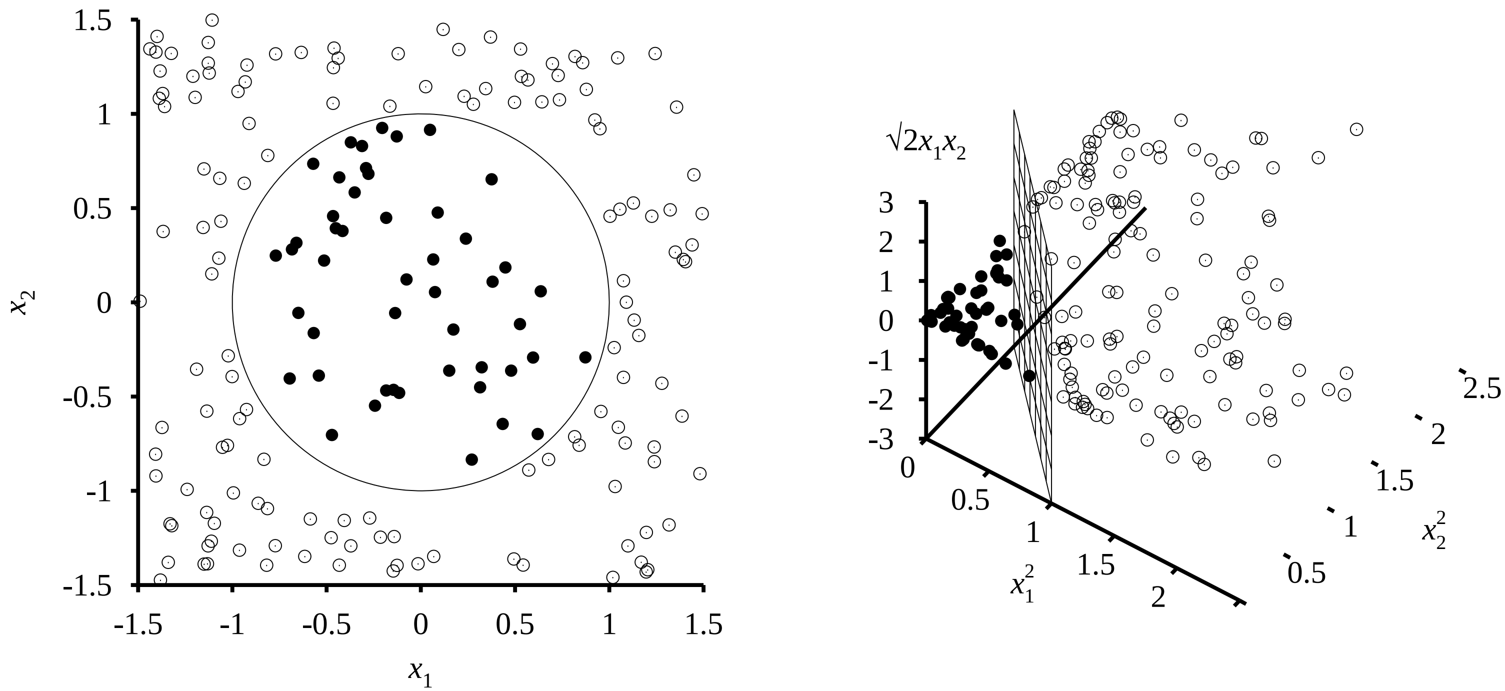
Mercer's Theorem (1909): any reasonable kernel function corresponds to some feature space

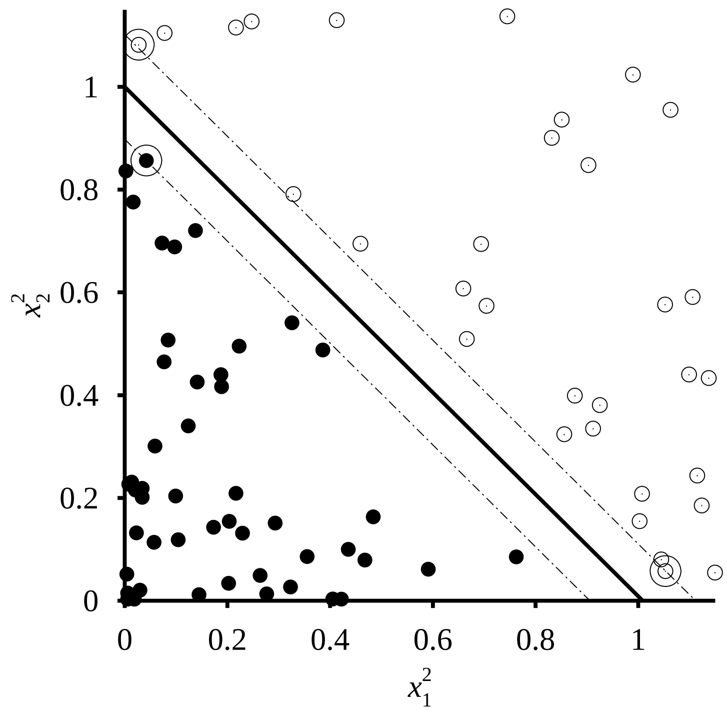
reasonable: $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite

features space can be very large, e.g., polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$ corresponds to feature space exponential in $d$

linear separators in these super high-dim spaces correspond to highly nonlinear decision boundaries in input space

# Kernel function example





0
0.2
0.4
0.6
0.8
1
$x_1^2$
$x_2^2$

# Summary

advantages:

- kernels allow very flexible hypotheses
- poly-time exact optimization rather than approximate methods
- soft-margin extension permits mis-classified examples
- variable sized hypothesis space
- excellent results (1.1% error rate on handwritten digit recognition, vs. LeNet's 0.9%)

disadvantages:

- must choose kernel, parameters
- very large problems computationally intractable
- batch algorithm