Lecture 21
Support Vector Machines II
**Slack variables**

What if data is not linearly separable??

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

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

\[ f(x) = \text{sign}(x \cdot w) = \text{sign} \left( \sum_{i} \alpha_{i} y_{i} (x \cdot x_{i}) \right) \]

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: \( x \rightarrow \Phi(x) \)
2. replace dot products between patterns: \( x_{i} \cdot x_{j} \rightarrow \Phi(x_{i}) \cdot \Phi(x_{j}) \)
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 \( K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \)

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

example:

\[
K(a, b) = (a \cdot b)^3 = ((a_1, a_2) \cdot (b_1, b_2))^3 \\
= (a_1 b_1 + a_2 b_2)^3 \\
= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\
= ((a_1^3, \sqrt{3}a_2^2, \sqrt{3}a_1 a_2, a_2^3) \cdot (b_1^3, \sqrt{3}b_1^2 b_2, \sqrt{3}b_1 b_2^2, b_2^3)) \\
= \Phi(a) \cdot \Phi(b)
\]
Kernels

examples:

1. polynomial kernel: \( K(x_i, x_j) = (x_i \cdot x_j + 1)^z \)

2. Gaussian kernel: \( K(x_i, x_j) = \exp(-||x_i - x_j||^2/2\sigma^2) \)

3. sigmoid kernel: \( K(x_i, x_j) = \tanh(\kappa(x_i \cdot x_j) + a) \)

each kernel computation corresponds to dot product calculation
for particular mapping \( \Phi(x) \) – 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 $K()$:

$$L_K = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

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

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

final decision function

$$f(x) = \text{sign}(\sum_i \alpha_i y_i K(x, x_i))$$
**Kernel functions**

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

reasonable: $K_{ij} = K(x_i, x_j)$ is positive definite

features space can be very large, e.g., polynomial kernel $K(x_i, x_j) = (1 + x_i \cdot x_j)^d$ 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
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