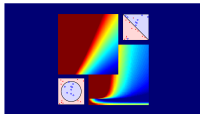


Machine Learning Techniques

(機器學習技巧)



Lecture 3: Kernel SVM

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)



Agenda

Lecture 3: Kernel SVM

- Kernel Trick
- Polynomial Kernel
- Gaussian Kernel
- Comparison of Kernels

Dual SVM Revisited

goal: SVM **without dependence on $\tilde{d}$**

half-way done:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \mathbf{1}^T \alpha \\ \text{subject to} \quad & \mathbf{y}^T \alpha = 0; \\ & \alpha_n \geq 0, \text{ for } n = 1, 2, \dots, N \end{aligned}$$

- $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$: **inner product** in $\mathbb{R}^{\tilde{d}}$
- need: $\mathbf{z}_n^T \mathbf{z}_m = \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m)$ calculated faster than $O(\tilde{d})$

can we do so?

Fast Inner Product for Φ_2

2nd order polynomial transform

$$\Phi_2(\mathbf{x}) = (1, x_1, x_2, \dots, x_d, x_1^2, x_1x_2, \dots, x_1x_d, x_2x_1, x_2^2, \dots, x_2x_d, \dots, x_d^2)$$

—include both x_1x_2 & x_2x_1 for ‘simplicity’ :-)

$$\begin{aligned} \Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') &= 1 + \sum_{i=1}^d x_i x'_i + \sum_{i=1}^d \sum_{j=1}^d x_i x_j x'_i x'_j \\ &= 1 + \sum_{i=1}^d x_i x'_i + \sum_{i=1}^d x_i x'_i \sum_{j=1}^d x_j x'_j \\ &= 1 + \mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')(\mathbf{x}^T \mathbf{x}') \end{aligned}$$

for Φ_2 , transform + inner product can be carefully done in $O(d)$ instead of $O(d^2)$

Kernel: Transform + Inner Product

transform $\Phi \iff$ **kernel function**: $K_{\Phi}(\mathbf{x}, \mathbf{x}') \equiv \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$

$$\Phi_2 \iff K_{\Phi_2}(\mathbf{x}, \mathbf{x}') = 1 + (\mathbf{x}^T \mathbf{x}') + (\mathbf{x}^T \mathbf{x}')^2$$

- quadratic coefficient $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m = y_n y_m K(\mathbf{x}_n, \mathbf{x}_m)$
- optimal bias b ? from **SV** $(\mathbf{x}_m, y_m)$,

$$b = y_m - \mathbf{w}^T \mathbf{z}_m = y_m - \left(\sum_{n=1}^N \alpha_n y_n \mathbf{z}_n \right)^T \mathbf{z}_m = y_m - \sum_{n=1}^N \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_m)$$

- optimal hypothesis g_{SVM} : for **test input** $\mathbf{x}$,

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}(\mathbf{w}^T \Phi(\mathbf{x}) + b) = \text{sign}\left(\sum_{n=1}^N \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$$

kernel trick: plug in **efficient kernel function**
to avoid dependence on $\tilde{d}$

Kernel SVM with QP

Kernel Hard-Margin SVM Algorithm

- ① $q_{n,m} = y_n y_m K(\mathbf{x}_n, \mathbf{x}_m)$; $\mathbf{c} = -\mathbf{1}_N$; $(\mathbf{P}, \mathbf{r})$ for equ./bound constraints
- ② $\alpha \leftarrow \text{QP}(\mathbf{Q}, \mathbf{c}, \mathbf{P}, \mathbf{r})$
- ③ $b \leftarrow \left(y_m - \sum_{\text{SV indices } n} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_m) \right)$ with SV $(\mathbf{x}_m, y_m)$
- ④ return SVs and their α_n as well as b such that for new $\mathbf{x}$,

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign} \left(\sum_{\text{SV indices } n} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b \right)$$

- ①: time complexity $O(N^2)$ · (kernel evaluation)
- ②: QP with N variables and $N + 1$ constraints
- ③ & ④: time complexity $O(\#\text{SV})$ · (kernel evaluation)

kernel SVM:

use computational shortcut to avoid $\tilde{d}$ & predict with SV only

Fun Time

General Poly-2 Kernel

$$\Phi_2(\mathbf{x}) = (1, x_1, \dots, x_d, x_1^2, \dots, x_d^2) \Leftrightarrow K_{\Phi_2}(\mathbf{x}, \mathbf{x}') = 1 + \mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')^2$$

$$\Phi_2(\mathbf{x}) = (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, x_1^2, \dots, x_d^2) \Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')^2$$

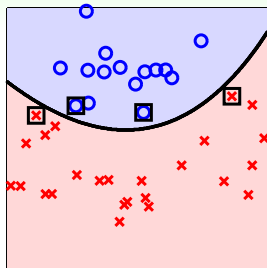
$$\Phi_2(\mathbf{x}) = (1, \sqrt{2a}x_1, \dots, \sqrt{2a}x_d, ax_1^2, \dots, ax_d^2) \\ \Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2a\mathbf{x}^T \mathbf{x}' + a^2(\mathbf{x}^T \mathbf{x}')^2$$

$$K_2(\mathbf{x}, \mathbf{x}') = (1 + a\mathbf{x}^T \mathbf{x}')^2 \text{ with } a > 0$$

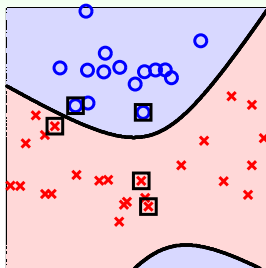
- K_2 : somewhat '**easier**' to calculate than K_{Φ_2}
- Φ_2 and Φ_2 : equivalent **power**,
different inner product $\Rightarrow$ different **geometry**

K_2 commonly used

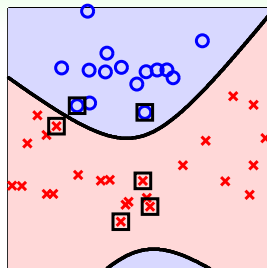
Poly-2 Kernels in Action



$$(1 + 0.001 \mathbf{x}^T \mathbf{x}')^2$$



$$1 + \mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')^2$$



$$(1 + 1000 \mathbf{x}^T \mathbf{x}')^2$$

- g_{SVM} **different**, SVs **different**
—‘hard’ to say which is better before learning
- change of **kernel** $\Leftrightarrow$ change of **margin definition**

need selecting K , just like selecting Φ

General Polynomial Kernel

$$K_2(\mathbf{x}, \mathbf{x}') = (r + a\mathbf{x}^T\mathbf{x}')^2 \text{ with } a > 0, r \geq 0$$

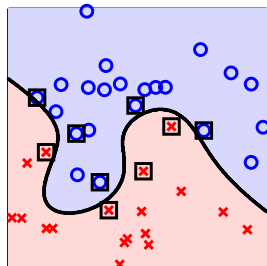
$$K_3(\mathbf{x}, \mathbf{x}') = (r + a\mathbf{x}^T\mathbf{x}')^3 \text{ with } a > 0, r \geq 0$$

$$\vdots$$

$$K_Q(\mathbf{x}, \mathbf{x}') = (r + a\mathbf{x}^T\mathbf{x}')^Q \text{ with } a > 0, r \geq 0$$

- embeds Φ_Q specially with parameters (a, r)
- allows computing large-margin **polynomial** classification **without dependence on $\tilde{d}$**

SVM + **Polynomial** Kernel: **Polynomial** SVM

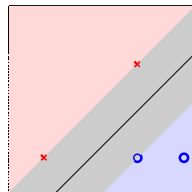


10-th order polynomial
with margin 0.1

Special Case: Linear Kernel

$$\begin{aligned} K_1(\mathbf{x}, \mathbf{x}') &= (0 + 1 \cdot \mathbf{x}^T \mathbf{x}')^1 \\ &\vdots \\ K_Q(\mathbf{x}, \mathbf{x}') &= (r + a\mathbf{x}^T \mathbf{x}')^Q \text{ with } a > 0, r \geq 0 \end{aligned}$$

- K_1 : just **usual inner product**, called **linear kernel**
- 'even easier': can be solved (often in primal form) **efficiently**



linear first, remember? :-)

Fun Time

Kernel of Infinite Dimensional Transform

infinite dimensional $\Phi(\mathbf{x})$? Yes, if $K(\mathbf{x}, \mathbf{x}')$ **efficiently computable**!

$$\begin{aligned}
 \text{when } \mathbf{x} = (x), \quad K(x, x') &= \exp(-(x - x')^2) \\
 &= \exp(-(x)^2) \exp(-(x')^2) \exp(2xx') \\
 &\stackrel{\text{Taylor}}{=} \exp(-(x)^2) \exp(-(x')^2) \left(\sum_{i=0}^{\infty} \frac{(2xx')^i}{i!} \right) \\
 &= \sum_{i=0}^{\infty} \left(\exp(-(x)^2) \exp(-(x')^2) \sqrt{\frac{2^i}{i!}} \sqrt{\frac{2^i}{i!}} (x)^i (x')^i \right) \\
 &= \Phi(x)^T \Phi(x')
 \end{aligned}$$

with infinite dimensional $\Phi(x) = \exp(-x^2) \cdot \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \dots \right)$

more generally, **Gaussian kernel**

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2) \text{ with } \gamma > 0$$

Hypothesis of Gaussian SVM

Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$

$$\begin{aligned} g_{\text{SVM}}(\mathbf{x}) &= \text{sign} \left(\sum_{\text{SV}} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b \right) \\ &= \text{sign} \left(\sum_{\text{SV}} \alpha_n y_n \exp(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2) + b \right) \end{aligned}$$

- linear combination of Gaussians centered at SVs $\mathbf{x}_n$
- also called Radial Basis Function (RBF) kernel

Gaussian SVM:

find α_n to combine Gaussians centered at $\mathbf{x}_n$
& achieve large margin in infinite-dim. space

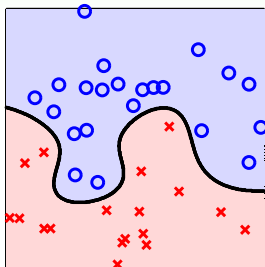
Support Vector Mechanism

	large-margin hyperplanes + higher-order transforms with kernel trick
#	not many
boundary	sophisticated

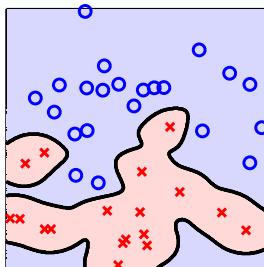
- transformed vector $\mathbf{z} = \Phi(\mathbf{x}) \implies$ efficient kernel $K(\mathbf{x}, \mathbf{x}')$
- store optimal $\mathbf{w} \implies$ store a few SVs and α_n

new possibility by Gaussian SVM:
infinite-dimensional linear classification, with
generalization 'guarded by' large-margin :-)

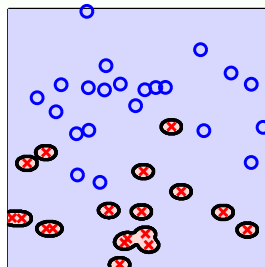
Gaussian SVM in Action



$$\exp(-1 \|\mathbf{x} - \mathbf{x}'\|^2)$$



$$\exp(-10 \|\mathbf{x} - \mathbf{x}'\|^2)$$



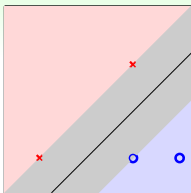
$$\exp(-100 \|\mathbf{x} - \mathbf{x}'\|^2)$$

- large $\gamma \implies$ sharp Gaussians $\implies$ 'overfit'?
- **warning: SVM can still overfit :-)**

Gaussian SVM: need careful selection of γ

Fun Time

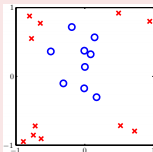
Linear Kernel: Cons and Pros



$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

Cons

- restricted
—**not always separable?!**

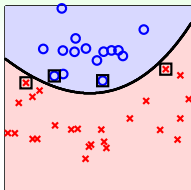


Pros

- safe—**linear first, remember? :-)**
- fast—with **special QP solver** in primal
- very explainable—**w and SVs** say something

linear kernel: an important **basic** tool

Polynomial Kernel: Cons and Pros



$$K(\mathbf{x}, \mathbf{x}') = (r + a\mathbf{x}^T \mathbf{x}')^Q$$

Cons

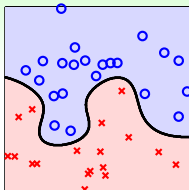
- **numerical difficulty** for large Q
 - $|r + a\mathbf{x}^T \mathbf{x}'| < 1: K \rightarrow 0$
 - $|r + a\mathbf{x}^T \mathbf{x}'| > 1: K \rightarrow \text{big}$
- three parameters (a, r, Q)
—**more difficult to select**

Pros

- **less restricted** than linear
- strong physical control
—‘knows’ **degree Q**

polynomial kernel: perhaps **small- Q only**
—sometimes efficiently done by **linear on $\Phi_Q(\mathbf{x})$**

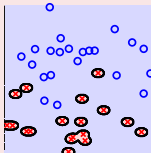
Gaussian Kernel: Cons and Pros



$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

Cons

- **mysterious**—no $\mathbf{w}$
- **slower** than linear
- **too powerful?!**



Pros

- **more powerful** than linear/poly.
- bounded—**less numerical difficulty** than poly.
- one parameter only—**easier to select** than poly.

Gaussian kernel: **one of most popular** but shall **use with care**

Other Valid Kernels

- **kernel** represents **special** similarity: $\Phi(\mathbf{x})^T \Phi(\mathbf{x}')$
- any similarity $\implies$ valid kernel? **not really**
- necessary & **sufficient** conditions for valid kernel:
Mercer's condition
 - symmetric
 - let $k_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, the matrix $\mathbf{K}$

$$\begin{aligned}
 &= \begin{bmatrix} \Phi(\mathbf{x}_1)^T \Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_1)^T \Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_1)^T \Phi(\mathbf{x}_N) \\ \Phi(\mathbf{x}_2)^T \Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_2)^T \Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_2)^T \Phi(\mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi(\mathbf{x}_N)^T \Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_N)^T \Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_N)^T \Phi(\mathbf{x}_N) \end{bmatrix} \\
 &= \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_N \end{bmatrix}^T \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_N \end{bmatrix} \\
 &= \mathbf{Z}\mathbf{Z}^T \text{ must always be positive semi-definite}
 \end{aligned}$$

define your own kernel: possible, **but hard**

Fun Time

Summary

Lecture 3: Kernel SVM

- Kernel Trick

kernel as shortcut of transform + inner product

- Polynomial Kernel

embeds specially-scaled polynomial transform

- Gaussian Kernel

embeds infinite dimensional transform

- Comparison of Kernels

linear for efficiency or Gaussian for power