Kernel Support Vector Machine

一、Kernel SVM

我们解决$d^{\sim}$的问题，因为必须把Z维度里面所有的东西都要算出来，这个复杂度比较高

我们尝试使用kernel trick

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

half-way done:

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

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

can we do so?

先看一下2-dimension的linear transformation

$\phi_2(x) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_1x_d, x_2^2, \ldots, x_2x_d, \ldots, x_d^2)$

一个非常简单的推导 $d^{\sim} = \frac{d(d+3)}{2} = O(d^2)$

换个方式：

$$\begin{aligned} \mathbf{\Phi}_2(\mathbf{x})^T \mathbf{\Phi}_2(\mathbf{x}') &= 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j' \\ &= 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} x_i x_i' \sum_{j=1}^{d} x_j x_j' \\ &= 1 + \mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')(\mathbf{x}^T \mathbf{x}') \end{aligned}$$

我们只需要计算$X^T X'$就OK了！

一般的kernel function

$$\phi \Leftrightarrow kernel\ function : K_\phi(x, x') \equiv \phi(x)^T \phi(x') \tag{1}$$

计算b的时候也可以优化步骤：（这里w的计算其实就被省略了，因为我们只需要计算inner product即可

- 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 $(\mathbf{x}_s, y_s)$,

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

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

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

总结一下kernel SVM的步骤：

Kernel SVM with QP

Kernel Hard-Margin SVM Algorithm

1. $q_{n,m} = y_n y_m K(\mathbf{x}_n, \mathbf{x}_m)$; $\mathbf{p} = -\mathbf{1}_N$; $(\mathrm{A}, \mathbf{c})$ for equ./bound constraints
2. $\alpha \leftarrow \text{QP}(\mathrm{Q_D}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
3. $b \leftarrow \left(y_s - \sum_{\text{SV indices } n} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_s)\right)$ with SV $(\mathbf{x}_s, y_s)$
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 $O(N^2) \cdot$ (kernel evaluation)
- (2): QP with N variables and $N+1$ constraints
- (3) & (4): time complexity $O(\#\text{SV}) \cdot$ (kernel evaluation)

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

至于C和A的解决：注意A不是对称矩阵，而是(N+2,N)的矩阵

$$\text{optimal } \alpha \leftarrow \text{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$$

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^T \mathrm{Q}\alpha + \mathbf{p}^T\alpha \\ \text{subject to} & \mathbf{a}_i^T\alpha \geq c_i, \\ & \text{for } i = 1, 2, \ldots \end{array}$$

- $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$
- $\mathbf{p} = -\mathbf{1}_N$
- $\mathbf{a}_{\geq} = \mathbf{y}, \mathbf{a}_{\leq} = -\mathbf{y};$
  $\mathbf{a}_n^T = n\text{-th unit direction}$
- $c_{\geq} = 0, c_{\leq} = 0; c_n = 0$

二、Kernel Trick

1. Polynomial Kernel

$$K_Q(x, x) = (\zeta + \gamma x^T x')^Q \; with \; \gamma > 0, \zeta \geq 0 \tag{2}$$

K_2 经常使用!

Special Case: Linear Kernel

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

原则：先使用线性核!

2. Gaussian Kernel(**RBF** kernel)——无限空间的转换!

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

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

$$
\phi(x,x') = \exp(-x^2)(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \ldots) \tag{3}
$$

more generally:

$$
K(x,x') = \exp(-\gamma||x-x'||^2)\ with\ \gamma > 0 \tag{4}
$$

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

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

- 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

于是我们终于解决了问题，总结一下

Support Vector Mechanism

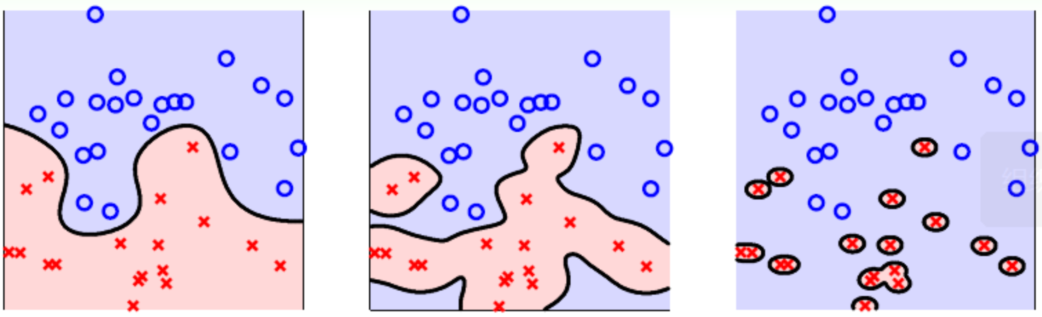
	large-margin hyperplanes + higher-order transforms with kernel trick
#	not many
boundary	sophisticated

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

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

3. 存在的问题：

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 $\gamma \Longrightarrow$ sharp Gaussians $\Longrightarrow$ 'overfit'?
- **warning: SVM can still overfit :-(**

Gaussian SVM: need careful selection of γ

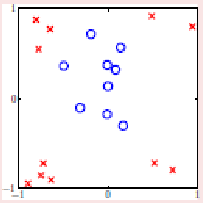
如果γ没有好好选择是很容易过拟合的，因此一般γ不能选的太大！

4. 选择kernel的方法

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

Cons

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



Pros

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

polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma \mathbf{x}^T \mathbf{x}')^Q$

Cons

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

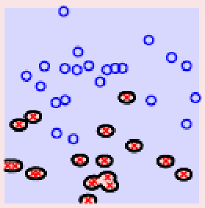
Pros

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

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

Cons

- **mysterious**—no **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.**

最常用的kernel! 但是解释性比较差

Other Valid Kernels

- kernel represents **special** similarity: $\Phi(\mathbf{x})^T \Phi(\mathbf{x}')$
- any similarity $\Longrightarrow$ 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 K

$$
\begin{aligned}
&= \begin{bmatrix} \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2) & \ldots & \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_N) \\ \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_2) & \ldots & \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_N) \\ \ldots & \ldots & \ldots & \ldots \\ \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_2) & \ldots & \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_N) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \ldots & \mathbf{z}_N \end{bmatrix}^T \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \ldots & \mathbf{z}_N \end{bmatrix} \\
&= \mathrm{ZZ}^T \text{ must } \textbf{always} \text{ be } \textbf{positive semi-definite}
\end{aligned}
$$

如果用的话，必须要先证明可行性——Mercer's Condition(可以证明是一个充要条件)，possible but hard!