

Kernel Support Vector Machine

1. Kernel Trick

推导的dual SVM是如下形式：

half-way done:

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

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

其中 α 是拉格朗日因子，共N个，这是我们要求解的，而条件共有N+1个。

向量 Q_D 中的 $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$ ， $\mathbf{z}_n^T \mathbf{z}_m$ 的内积中会引入 \hat{d} 。即，如果 \hat{d} 很大，计算 $\mathbf{z}_n^T \mathbf{z}_m$ 的复杂度也会很高，会影响QPI问题的计算效率。

so, $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$ 这一步是计算的瓶颈所在。

其实问题的关键在于 $\mathbf{z}_n^T \mathbf{z}_m$ 内积求解上。我们知道， \mathbf{z} 是由 \mathbf{x} 经过特征转换而来：

$$\mathbf{z}_n^T \mathbf{z}_m = \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m)$$

如果从 \mathbf{x} 空间来看的话， $\mathbf{z}_n^T \mathbf{z}_m$ 分为两个步骤：

1. 进行特征转换 $\Phi(\mathbf{x}_n)$ 和 $\Phi(\mathbf{x}_m)$ ；
2. 计算 $\Phi(\mathbf{x}_n)$ 与 $\Phi(\mathbf{x}_m)$ 的内积。

这种先转换再计算内积的方式，必然会引入 \hat{d} 参数，从而在 \hat{d} 很大的时候影响计算速度。尝试将两个步骤联合起来。

我们先来看一个简单的例子，对于二阶多项式转换，各种排列组合为：

2nd order polynomial transform

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

—include both $x_1 x_2$ & $x_2 x_1$ for 'simplicity' :-)

内积推导：

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

其中 $\mathbf{x}^T \mathbf{x}'$ 是 \mathbf{x} 空间中特征向量的内积。

所以， $\Phi_2(\mathbf{x})$ 与 $\Phi_2(\mathbf{x}')$ 的内积的复杂度由原来的 $O(d^2)$ 变成 $O(d)$ ，只与 \mathbf{x} 空间的维度 d 有关，而与 \mathbf{z} 空间的维度 \hat{d} 无关。

我们把合并特征转换和计算内积这两个步骤的操作叫做Kernel Function，用大写字母K表示。例如刚刚讲的二阶多项式例子，它的kernel function为：

$$K_{\Phi}(x, x') = \Phi(x)^T \Phi(x')$$

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

有了kernel function之后，我们来看看它在SVM里面如何使用。在dual SVM中，二次项系数 $q_{n,m}$ 中有 \mathbf{z} 的内积计算，就可以用kernel function替换：

$$q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m = y_n y_m K(x_n, x_m)$$

所以，直接计算出 $K(x_n, x_m)$ ，再代入上式，就能得到 $q_{n,m}$ 的值。

$q_{n,m}$ 值计算之后，就能通过QP得到拉格朗日因子 α_n 。然后，下一步就是计算 b （取 $\alpha_n > 0$ 的点，即SV）， b 的表达式中包含 z ，可以作如下推导：

$$b = y_s - w^T z_s = y_s - \left(\sum_{n=1}^N \alpha_n y_n z_n \right)^T z_s = y_s - \sum_{n=1}^N \alpha_n y_n (K(x_n, x_s))$$

这样得到的 b 就可以用 kernel function 表示，而与 z 空间无关。

最终我们要求的 g_{SVM} 可以作如下推导：

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

至此，dual SVM 中我们所有需要求解的参数都已经得到了，而且整个计算过程中都没有在 z 空间作内积，即与 z 无关。我们把这个过程称为 kernel trick，也就是把特征转换和计算内积两个步骤结合起来，用 kernel function 来避免计算过程中受 \hat{d} 的影响，从而提高运算速度。

总结一下，引入 kernel function 后，svm 算法变成：

Kernel Hard-Margin SVM Algorithm

- 1 $q_{n,m} = y_n y_m K(\mathbf{x}_n, \mathbf{x}_m)$; $\mathbf{p} = -\mathbf{1}_N$; (\mathbf{A}, \mathbf{c}) for equ./bound constraints
- 2 $\alpha \leftarrow \text{QP}(\mathbf{Q}_D, \mathbf{p}, \mathbf{A}, \mathbf{c})$
- 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} ,

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

每个步骤的时间复杂度为：

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

我们把这种引入 kernel function 的 SVM 称为 kernel SVM，它是基于 dual SVM 推导而来的。kernel SVM 同样只用 SV ($\alpha_n > 0$) 就能得到最佳分类面，而且整个计算过程中摆脱了 \hat{d} 的影响，大大提高了计算速度。

2. Polynomial Kernel

二次多项式的 kernel 形式多样。

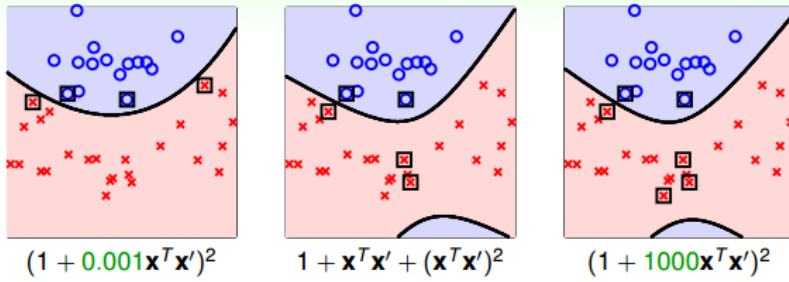
$$\begin{aligned} \Phi_2(\mathbf{x}) &= (1, x_1, \dots, x_d, x_1^2, \dots, x_d^2) \Leftrightarrow K_{\Phi_2}(\mathbf{x}, \mathbf{x}') = 1 + \mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')^2 \\ \Phi_2(\mathbf{x}) &= (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, x_1^2, \dots, x_d^2) \Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\mathbf{x}^T \mathbf{x}' + (\mathbf{x}^T \mathbf{x}')^2 \\ \Phi_2(\mathbf{x}) &= (1, \sqrt{2\gamma}x_1, \dots, \sqrt{2\gamma}x_d, \gamma x_1^2, \dots, \gamma x_d^2) \\ &\Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\gamma \mathbf{x}^T \mathbf{x}' + \gamma^2 (\mathbf{x}^T \mathbf{x}')^2 \end{aligned}$$

系数不同，内积就会有差异，就会代表不同的距离，最终可能会得到不同的 SVM margin。所以，系数不同，可能会得到不同的 SVM 分界线。

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

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

不同的转换，对应到不同的几何距离



引入 $\zeta \geq 0$ 和 $\gamma > 0$ ，对于Q次多项式一般的kernel形式可表示为：

$$\begin{aligned} K_2(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma \mathbf{x}^T \mathbf{x}')^2 \text{ with } \gamma > 0, \zeta \geq 0 \\ K_3(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma \mathbf{x}^T \mathbf{x}')^3 \text{ with } \gamma > 0, \zeta \geq 0 \\ &\vdots \\ K_Q(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma \mathbf{x}^T \mathbf{x}')^Q \text{ with } \gamma > 0, \zeta \geq 0 \end{aligned}$$

所以，使用高阶的多项式kernel有两个优点：

1. 得到最大SVM margin，SV数量不会太多，分类面不会太复杂，防止过拟合，减少复杂度
2. 计算过程避免了对 \hat{d} 的依赖，大大简化了计算量。

3. Gaussian Kernel

如果是无限多维的转换 $\Phi(x)$ ，也还能通过kernel的思想，来简化SVM的计算 先举个例子，简单起见，假设原空间是一维的，只有一个特征 x ，我们构造一个kernel function为高斯函数：

$$K(x, x') = e^{-(x-x')^2}$$

构造的过程正好与二次多项式kernel的相反，利用反推法，先将上式分解并做泰勒展开：

$$\begin{aligned} \text{when } \mathbf{x} = (x), K(\mathbf{x}, \mathbf{x}') &= \exp(-(x-x')^2) \\ &= \exp(-(x)^2) \exp(-(x')^2) \exp(2xx') \\ &\stackrel{\text{Taylor}}{=} \exp(-(x)^2) \exp(-(x')^2) \left(\sum_{i=0}^{\infty} \frac{(2xx')^i}{i!} \right) \\ &= \sum_{i=0}^{\infty} \left(\exp(-(x)^2) \exp(-(x')^2) \sqrt{\frac{2^i}{i!}} \sqrt{\frac{2^i}{i!}} (x)^i (x')^i \right) \\ &= \Phi(x)^T \Phi(x') \\ \text{with infinite dimensional } \Phi(x) &= \exp(-x^2) \cdot \left(1, \sqrt{\frac{2}{1!}} x, \sqrt{\frac{2^2}{2!}} x^2, \dots \right) \end{aligned}$$

将构造的 $K(x, x')$ 推导展开为两个 $\Phi(x)$ 和 $\Phi(x')$ 的乘积，其中：

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

通过反推，我们得到了 $\Phi(x)$ ， $\Phi(x)$ 是无限多维的，它就可以当成特征转换的函数，且 \hat{d} 是无限的。这种 $\Phi(x)$ 得到的核函数即为Gaussian kernel。

更一般地，对于原空间不止一维的情况（ $d > 1$ ），引入缩放因子 $\gamma > 0$ ，它对应的Gaussian kernel表达式为：

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

那么引入了高斯核函数，将有限维度的特征转换拓展到无限的特征转换中。根据本节课上一小节的内容，由 K ，计算得到 α_n 和 b ，进而得到矩 g_{SVM} 。将其中的核函数 K 用高斯核函数代替，得到：

$$g_{SVM}(x) = \text{sign} \left(\sum_{SV} \alpha_n y_n K(x_n, x) + b \right) = \text{sign} \left(\sum_{SV} \alpha_n y_n e^{(-\gamma \|x - x_n\|^2)} + b \right)$$

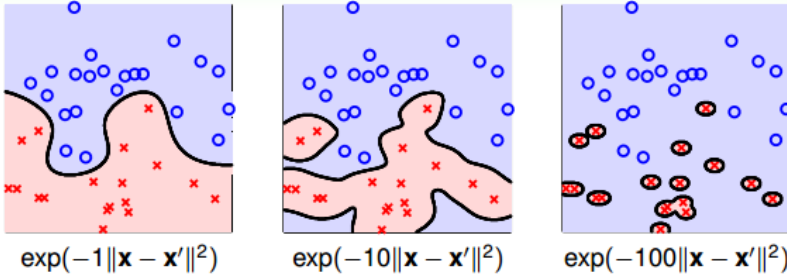
通过上式可以看出， g_{SVM} 有 n 个高斯函数线性组合而成，其中 n 是SV的个数。而且，每个高斯函数的中心都是对应的SV。通常我们也把高斯核函数称为径向基函数（Radial Basis Function, RBF）。

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

$$= \text{sign} \left(\sum_{\text{SV}} \alpha_n y_n \exp(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2) + b \right)$$

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

缩放因子 γ 取值不同，会得到不同的高斯核函数，hyperplanes不同，分类效果也有很大的差异。举个例子， γ 分别取1, 10, 100时对应的分类效果如下：



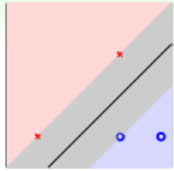
从图中可以看出，当 γ 比较小的时候，分类线比较光滑，当 γ 越来越大的时候，分类线变得越来越复杂和扭曲。因为 γ 越大，其对应的高斯核函数越尖瘦，那么有限个高斯核函数的线性组合就比较离散，分类效果并不好。所以，SVM也会出现过拟合现象， γ 的正确选择尤为重要，不能太大。

4. Comparison of Kernels

对几种核进行比较。

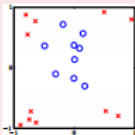
Linear Kernel是最基本最简单的核，平面上对应一条直线，三维空间内对应一个平面。

Linear Kernel可以使用Dual SVM中的QP直接计算得到。

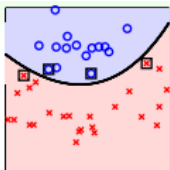


$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

优点是计算简单、快速，可以直接使用QP快速得到参数值，而且从视觉上分类效果非常直观，便于理解
缺点是如果数据不是线性可分的情况，Linear Kernel就不能使用了。

Cons	Pros
<ul style="list-style-type: none"> • restricted —not always separable?! 	<ul style="list-style-type: none"> • safe—linear first, remember? :-) • fast—with special QP solver in primal • very explainable—\mathbf{w} and SVs say something

Polynomial Kernel的hyperplanes是由多项式曲线构成。



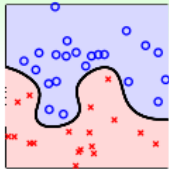
$$K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma \mathbf{x}^T \mathbf{x}')^Q$$

优点是阶数 Q 可以灵活设置，相比linear kernel限制更少，更贴近实际样本分布

缺点是当Q很大时，K的数值范围波动很大，而且参数个数较多，难以选择合适的值。

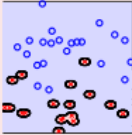
Cons	Pros
<ul style="list-style-type: none"> numerical difficulty for large Q <ul style="list-style-type: none"> $\zeta + \gamma \mathbf{x}^T \mathbf{x}' < 1: K \rightarrow 0$ $\zeta + \gamma \mathbf{x}^T \mathbf{x}' > 1: K \rightarrow \text{big}$ three parameters (γ, ζ, Q) —more difficult to select 	<ul style="list-style-type: none"> less restricted than linear strong physical control —'knows' degree Q

对于Gaussian Kernel，表示为高斯函数形式。



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

优点是边界更加复杂多样，能最准确地区分数据样本，数值计算K值波动较小，而且只有一个参数，容易选择。缺点是特征转换到无限维度中，w没有求解出来，计算速度要低于linear kernel，而且可能会发生过拟合。

Cons	Pros
<ul style="list-style-type: none"> mysterious—no w slower than linear too powerful?! 	<ul style="list-style-type: none"> more powerful than linear/poly. bounded—less numerical difficulty than poly. one parameter only—easier to select than poly.

kernel代表的是两笔资料x和x'，特征变换后的相似性即内积。

但是不能说任何计算相似性的函数都可以是kernel。

有效的kernel还需满足几个条件：

- 1.K是对称的
- 2.K是半正定的

这两个条件不仅是必要条件，同时也是充分条件。

只要我们构造的K同时满足这两个条件，那它就是一个有效的kernel(Mercer定理)

- 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 **K**

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