

Kernels

Today, we talk about kernels. These are magical beings that allow us to turn (nearly) any linear model into a non-linear model!

Consider 1D, non-linearly separable data: “+ + + - - - + + +”. Now, map each location x to a 2D point (x, x^2) . Then, in 2D, the data *is* linearly separable.

We can generalize: for $x \in \mathbb{R}^D$, map $\Phi : x \mapsto (x_1^2, x_2^2, \dots, x_D^2, x_1x_2, x_1x_3, \dots, x_1x_D, \dots, x_{D-1}x_D)$.

Or maybe we want cubic function! This is computationally a nightmare!

Recall SVM (dual) optimization:

$$\begin{aligned} \text{minimize}_{\alpha} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{n,m} y_n y_m \alpha_n \alpha_m x_n^\top x_m \\ \text{subject to} \quad & \sum_n y_n \alpha_n = 0 \\ & \alpha_n \geq 0 \quad , \quad (\forall n) \end{aligned}$$

And SVM prediction:

$$f(x') = \text{sign} \left[\sum_{n \in \text{SV}} \alpha_n y_n x_n^\top x' \right]$$

In both cases, inputs only ever appear in dot products!

So, we just replace all x s with $\Phi(x)$ s:

$$\begin{aligned} \text{minimize}_{\alpha} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{n,m} y_n y_m \alpha_n \alpha_m \Phi(x_n)^\top \Phi(x_m) \\ \text{subject to} \quad & \sum_n y_n \alpha_n = 0 \\ & \alpha_n \geq 0 \quad , \quad (\forall n) \\ \text{prediction :} \quad & \text{sign} \left[\sum_{n \in \text{SV}} \alpha_n y_n \Phi(x_n)^\top \Phi(x') \right] \end{aligned}$$

Insight: for many Φ , we can compute dot products without actually computing the resulting vectors!

Consider the quadratic case in two dimensions (with an addition of some constant terms). The dot product can be computed as:

$$\begin{aligned} \Phi(x)^\top \Phi(z) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2)^\top (z_1^2, z_2^2, \sqrt{2}z_1z_2) \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (x^\top z)^2 \end{aligned}$$

So the cost of computing the dot product in the high-dimensional space is the same as the cost in the low-dimensional space.

Can additionally include a linear term: $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$ by using $(1 + x^\top z)^2$.

Such constructions are called *kernel products*: $K_\Phi(x, z) = \Phi(x)^\top \Phi(z)$ (often subscript of Φ is dropped from K when it is clear from context).

A bit of theory...

We write the original space (the “input space”) as \mathcal{X} and the output of Φ (the “feature space”) as \mathcal{F} .

- $\Phi : \mathcal{X} \rightarrow \mathcal{F}$
- $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{R}$, $K(x, z) = \Phi(x)^\top \Phi(z)$
- The previous means that \mathcal{F} must be a *vector space* with a *dot product*: these are called *Hilbert spaces*.

Question: can I used *any* function K as a kernel?

Answer: No! There must exist some Hilbert space \mathcal{F} for which K is a dot product.

Question: Ack! How can I know that?

Answer: A sufficient condition is that K be positive definite:

$$\int dx \int dz f(x)K(x, z)f(z) > 0 \quad , \quad (\forall f \in L_2)$$

This is equivalent to saying that the resulting matrix for any data set be positive definite; define $K_{ij} = K(x_i, x_j)$ and require that K be positive definite. (Mercer’s condition)

Why does this matter? It helps us build kernels...

Building kernels...

If K_1, K_2 are kernels and α is a positive real number, then the following are also all kernels:

- $K(x, z) = K_1(x, z) + K_2(x, z)$ – direct sum
- $K(x, z) = \alpha K_1(x, z)$ – scalar product
- $K(x, z) = K_1(x, z)K_2(x, z)$ – direct product

Combining these, any (positive) linear combination of kernels is a kernel.

Okay, so where do they come from in the first place? We’ve already seen two:

- Linear: $K(x, z) = x^\top z$
- Quadratic: $K(x, z) = (1 + x^\top z)^2$

We can actually generalize the second to a polynomial degree d kernel:

$$K(x, z) = (1 + x^\top z)^d$$

The final “standard” kernel is the RBF kernel. This is harder to understand; it is defined by:

$$K^{\text{rbf}}(x, z) = \exp \left[-\gamma \|x - z\|^2 \right]$$

Here, K is parameterized by a h.p., γ .

This is actually an *infinite-dimensional* kernel: there is no finite space \mathcal{F} such that K corresponds to a dot product in \mathcal{F} .

The RBF kernel produces decision boundaries that look like blobs.

Think of a kernel as a similarity function and everything makes more sense (hopefully!).

Using kernels...

For any kernel you want, just plug it in to the SVM optimization and you get a non-linear classifier. (Most software supports this.)

How do I know the best kernel for my problem? Try them all? Do you have intuition?

Usually non-linear kernels introduce extra h.p.s to tune.

Can kernelize other algorithms than just SVM: eg., perceptron is also easy.