Kernalized linear regression

If we are using basis functions, we are transforming the input space.

eg if we want polynomial of degree n then the basis function becomes more and more complex

x	degree	d	m	$\phi(x)$
$[x_1]$	2	1	2	$[x_1^2, x_1]$
$[x_1, x_2]$	2	2	5	$[x_1^2, x_2^2, x_1x_2, x_1, x_2]$
$[x_1\ x_2\ x_3]$	2	3	9	$[x_1^2\ x_2^2\ x_3^2\ x_1x_2\ x_2x_3\ x_1x_3 \ldots]$
	k	d	${}^{k+d}C_d - 1$	

The amount of computation grows with combination, in terms of factorial.

We are doing the computation on millions of datapoints

eg n: 1000, $k = 3$ (cubic)
$d = 5$

then $x = 1000 \times 5 = 5000$ element matrix

$\Phi = {}^{5+3}C_5 \times 1000 = 56{,}000$ element matrix

Time to compute the kernel is high
Time to compute OLS from such a huge matrix also is high

What if we want to fit a polynomial with degree 15?
the degree of Φ would be 8!

Another form of OLS

$w = (X^T X)^{-1} X^T y$, p is a datapoint

$\hat{y} = \underset{1\times d}{p^T} \underset{d\times 1}{w} = p^T (X^T X)^{-1} X^T y$

Using identity $(X^T X)^{-1} X^T = X^T (X X^T)^{-1}$

$\hat{y} = \underset{1\times d}{p^T} \underset{d\times n}{X^T} \underset{n\times d}{(X X^T)^{-1}} \underset{n\times 1}{y}$

But we can write $p^T X^T$ as

$$p^T X^T = \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & & x_{n2} \\ \vdots & \vdots & & \vdots \\ x_{1d} & x_{2d} & & x_{nd} \end{bmatrix}$$

$$= \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \cdots & x_n \\ 1 & 1 & & 1 \end{bmatrix}$$

Where $x_i \in \mathbb{R}^d$ is a datapoint $i \in (0, n)$

$$\therefore p^T X^T = \begin{bmatrix} \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ 1 \end{bmatrix}, & \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ 1 \end{bmatrix} & \cdots \end{bmatrix}$$

$$= \begin{bmatrix} p^T x_1, & p^T x_2, & \cdots & p^T x_n \end{bmatrix}$$

$$\therefore \hat{y} = \underset{n\times d\, d\times n}{\begin{bmatrix} p^T x_1, & p^T x_2, & \cdots & p^T x_n \end{bmatrix}} \cdot \underset{n\times 1}{(X X^T)^{-1} y}$$

Let $(X X^T)^{-1} y$ be α. $\alpha \in \mathbb{R}^{n\times 1}$

$$\therefore \hat{y} = \begin{bmatrix} p^T x_1, & p^T x_2, & \cdots & p^T x_n \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

$$= \sum_{i=0}^{n} \underset{1\times d\, d\times 1}{p^T x_i} \underset{1\times 1}{\alpha_i}$$ Both terms are scalar

$$\hat{y} = \sum_{i=0}^{n} \alpha_i (p^T x_i)$$

Now, $\alpha = (X X^T)^{-1} y$

$$X X^T = \begin{bmatrix} -x_1- \\ -x_2- \\ \vdots \\ -x_n- \end{bmatrix} \begin{bmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \cdots & x_n \\ 1 & 1 & & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \underset{1\times d\, d\times 1}{x_1^T x_1} & x_1^T x_2 & \cdots & x_1^T x_n \\ x_2^T x_1 & x_2^T x_2 & \cdots & x_2^T x_n \\ \vdots & & & \\ x_n^T x_1 & x_n^T x_2 & \cdots & x_n^T x_n \end{bmatrix}$$

Notice that whenever x comes, it comes in the form of $x_i^T x_j$

The Kernel trick

If we use $x = \phi(x)$ then we get

$$\hat{y} = \sum_i^n \alpha_i \, \phi(p)^T \phi(x_i) \qquad x_i \in \mathbb{R}^d$$

$$\phi: \mathbb{R}^d \to \mathbb{R}^m$$

$$\alpha = (\Phi^T \Phi)^{-1} y$$

Instead of calculating Φ everytime, why dont we directly calculate α

Instead of ever computing $\phi(x)$ we directly calculate $\phi(x_i)^T \phi(x_j)$

That is we directly calculate a bivariate function

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

$$\therefore \quad \hat{y} = \sum_i^n \alpha_i \; k(p, x_i)$$

$$\alpha = (\Phi^T \phi)^{-1} y$$

But $\Phi^T \phi = \begin{bmatrix} \Phi_1^T\Phi_1 & \Phi_1^T\Phi_2 & \cdots & \Phi_1^T\Phi_n \\ \Phi_2^T\Phi_1 & \Phi_2^T\Phi_2 & \cdots & \Phi_2^T\Phi_n \\ \vdots & \vdots & & \vdots \\ \Phi_n^T\Phi_1 & \Phi_n^T\Phi_2 & \cdots & \Phi_n^T\Phi_n \end{bmatrix}$

$$= \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & \cdots & \phi(x_1)^T\phi(x_n) \\ \phi(x_2)^T\phi(x_1) & \phi(x_2)^T\phi(x_2) & \cdots & \phi(x_2)^T\phi(x_n) \\ \vdots & & & \\ \phi(x_n)^T\phi(x_1) & \phi(x_n)^T\phi(x_2) & \cdots & \phi(x_n)^T\phi(x_n) \end{bmatrix}$$

$n \times m \quad m \times 1$

$$= \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & \vdots & & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{bmatrix} = K$$

$$\therefore \quad \alpha = K^{-1} y \qquad K \in \mathbb{R}^{n \times n}$$

So instead of calculating Φ, then $\Phi^T\Phi$, we can directly calculate K

eg. →

$$\phi(x) = \begin{bmatrix} x_1 & x_2 & x_1x_2 & x_1^2 & x_2^2 \end{bmatrix}^T$$

$$\phi(x)^T \phi(x') = x_1x_1' + x_2x_2' + x_1x_2x_1'x_2' + x_1^2x_1'^2 + x_2^2x_2'^2$$

$$k(x, x') = (x^T x' + 1)^2$$

$$= (x_1x_1' + x_2x_2' + 1)^2$$

$$= x_1x_1' + x_2x_2' + x_1x_2x_1'x_2' + x_1^2x_1'^2 + x_2^2x_2'^2$$

Both give the same terms (constants dont matter) but number of computations for $k(x_1, x_2)$ is much lesser (only one squaring)

Kernel Trick

The advantage of using the kernel trick is that we don't really ever calculate $\phi(x)$ hence we never need to find it in closed form

This allows for complex cases like the RGF kernel which has ∞ dimensions.

Properties of kernel function →

① Symmetry $k(x_1, x_2) = k(x_2, x_1)$

② Positive semidefiniteness of Gram matrix $z^T K z \geq 0 \quad \forall z \in \mathbb{R}^n$

$$K = \Phi^T \Phi = \begin{bmatrix} \Phi_1^T\Phi_1 & \Phi_1^T\Phi_2 & \cdots & \Phi_1^T\Phi_n \\ \Phi_2^T\Phi_1 & \Phi_2^T\Phi_2 & \cdots & \Phi_2^T\Phi_n \\ \vdots & \vdots & & \vdots \\ \Phi_n^T\Phi_1 & \Phi_n^T\Phi_2 & \cdots & \Phi_n^T\Phi_n \end{bmatrix}$$

Gram Matrix

By Mackers theorem, if the above two properties are satisfied, then there exists a feature map such that

$$k(x, x') = \phi^T(x)\, \phi(x')$$

Common types of kernels

① stationary kernels

$k(x, x') =$ function of $(x - x')$

Invariant to translations in input space

② Homogenous / radial basis functions

$k(x, x') =$ function of $\|x - x'\|$

depend only on the magnitude of the distance

Property of K

if $K(x_i, x_j)$ can be broken down as $\phi(x_i)^T \phi(x_j)$ then it is a kernel function.

$K(x_i, x_j)$ is a kernel function iff

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x_i, x_j) \geq 0 \quad \forall a_i, a_j \in \mathbb{R}$$

Note → k is symmetric $k(a,b) = k(b,a)$

Proof → $A \rightarrow B$

if $K(x) = \phi(x_j)^T \phi(x_j)$

then $\sum_{i}^{N} \sum_{j}^{N} a_i a_j K(x_i, x_j) = \sum_{i}^{N} \sum_{j}^{N} a_i a_j \phi(x_i)^T \phi(x_j)$

$$= \left(\sum_{j}^{N} a_j \phi^T(x_j) \right) \left(a_0 \phi(x_0) \right) + \left(\sum_{j}^{N} a_j \phi^T(x_j) \right) \left(a_1 \phi(x_1) \right) + \cdots + \left(\sum_{j}^{N} a_j \phi^T(x_j) \right) a_n \phi(x_n)$$

$$= \left(\sum_{j}^{N} a_j \phi(x_j) \right)^T \left(\sum_{j=0}^{N} a_j \phi(x_j) \right) = \left\| \sum_{j=1}^{N} a_j \phi(x_j) \right\|_2^2$$

This is always ≥ 0.

Hence if $K(x) = \phi(x_i)^T \phi(x_j)$ then

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x_i, x_j) \geq 0 \quad \forall a_i, a_j \in \mathbb{R}$$

Proof $B \rightarrow A$

We can write $\sum a_i a_j K(x_i, x_j)$ as

$$\begin{bmatrix} a_1 a_2 \cdots a_n \end{bmatrix} \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \cdots & K(x_1,x_n) \\ K(x_2,x_1) & K(x_2,x_2) & \cdots & \\ \vdots & & & \\ & & & K(x_n,x_1) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \geq 0$$

$$a^T K a \geq 0$$

$\therefore$ K is P.S.D. , $\therefore$ K can be factorized as $Q Q^T$

$\therefore$ $k(x_i, x_j)$ can be written as $\phi(x_i) \phi(x_j)$

Corollary Properties

① $k_{ij}^2 \leq k_{ii}\, k_{jj}$

Proof → $k_{ij}^2 = \left(\phi^T(x_i)\ \phi(x_j)\right)^2$

$k_{ii} = \phi^T(x_i)\ \phi(x_i) = \|\phi(x_i)\|^2$

$k_{jj} = \phi^T(x_j)\ \phi(x_j) = \|\phi(x_j)\|^2$

By Cauchy Schwarz

$$\left(\phi^T(x_i)\ \phi(x_j)\right)^2 \leq \|\phi(x_i)\|^2 \|\phi(x_j)\|^2$$

② Diagonal positivity

$K = V \Lambda V^T$ (eigendecomposition)

$V \rightarrow$ eigenvectors

$\Lambda \geq 0$ (P.S.D.)

ϕ can be constructed as

$$\phi(x_i) = \sqrt{\lambda_i}\ V_i^T$$

$$k_{ij} = \lambda_i \|V_i\|_2^2$$

Proof →

$$\phi = \begin{bmatrix} \sqrt{\lambda_1}\, V_1^T \\ \sqrt{\lambda_2}\, V_2^T \\ \vdots \\ \sqrt{\lambda_n}\, V_n^T \end{bmatrix}$$

$$\phi^T\phi = \begin{bmatrix} \lambda_1 V_1 V_1^T \\ \lambda_2 V_2 V_2^T \\ \vdots \\ \lambda_n V_n V_n^T \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 V_1 & \lambda_2 V_2 \cdots \lambda_n V_n \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \\ \vdots \\ V_n^T \end{bmatrix}$$

$$= \begin{bmatrix} V_1 & V_2 & \cdots & V_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 & \cdot \\ 0 & \lambda_2 & 0 & \cdots \\ \vdots & & & \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \\ \vdots \\ V_n^T \end{bmatrix}$$

$= V \Lambda V^T$ Hence proved

Note → because K is symmetric & P.S.D.

SVD gives $U \Sigma V^T$ with $U = V$, $\Sigma = \Lambda$ } same as eigendecomposition

∞ dimensional Vectors

Consider $x = [a, a^2, \ldots\ldots a^\infty]^T$, $a < 1$

$x \in \mathbb{R}^\infty$ is an ∞ dimensional vector

what is $x^T x$? $x^T x = \sum_{n=1}^{\infty} a^{2n} = \frac{1}{1-2a}$

Even if we cant represent an ∞ dimensional vector, we can represent its dot product with another ∞ dimensional vector!

Even if we dont have closed form for $\phi(x)$ we can have a closed form for $\phi(x) \cdot \phi(y)$

That means even if we dont know $\phi(x)$ but we know $\phi(x)\,\phi(y)$ it's okay.

Or rather we dont need to know $\phi(x)$

Composition of Kernels

Given 2 valid kernels $k_1(x, x')$ & $k_2(x, x')$ the following is also a kernel

1. $c\, k_1(x, x')$
2. $f(x)\, k_1(x, x')\, f(x')$
3. $q(k_1(x, x'))$

($q \rightarrow$ polynomial with non negative coefficients)

4. $\exp(k_1(x, x'))$
5. $k_1(x, x') + k_2(x, x')$
6. $k_1(x, x')\, k_2(x, x')$
7. $k(f(x), f(x'))$

$f : \mathbb{R}^d \rightarrow \mathbb{R}^M$

For more details on "kernel engineering", see Shawe-Taylor & Cristianini (2004)

Kernalized Ridge Regression

We have

$$w^* = \underset{w}{\operatorname{argmin}} \| y - Xw \|_2^2 + \lambda \| w \|_2^2$$

Xw is an N dimensional vector

$$Xw = \begin{bmatrix} x_1^T w \\ x_2^T w \\ \vdots \\ x_N^T w \end{bmatrix} \qquad x_i \in \mathbb{R}^d, \; w \in \mathbb{R}^d$$

By orthogonal decomposition theorem, we can write w as

$$w = w_{\parallel} + w_{\perp}$$

$w_{\parallel}$: Parallel component — $w_{\parallel} \in$ Span of rows of X

$w_{\perp}$: Perpendicular — $w_{\perp} \perp$ Span of rows of X, $Xw_{\perp} = 0$

$w_{\parallel} \in \mathbb{R}^d$, $w_{\perp} \in \mathbb{R}^d$

$$\therefore Xw = X(w_{\parallel} + w_{\perp}) = Xw_{\parallel} + Xw_{\perp} = Xw_{\parallel}$$

By Pythagores theorem

$$\| w \|_2^2 = \| w_{\parallel} \|_2^2 + \| w_{\perp} \|_2^2$$

Hence objective becomes

$$\underset{w_{\perp}, w_{\parallel}}{\operatorname{argmin}} \| y - Xw_{\parallel} \|_2^2 + \| w_{\parallel} \|_2^2 + \| w_{\perp} \|_2^2$$

Now we have 2 seperate variables $w_{\parallel}$ & $w_{\perp}$ to optimize over

Since $\| w_{\perp} \|^2 \geq 0$, the objective is strictly increasing. Hence the minimum occurs when $w_{\perp} = 0$

That is, the optimal $w_{\perp} = 0$.

Hence, the optimal w^* has $w_{\perp} = 0$, ie w^* lies entirely in span of X

Since $w^* \in \text{span}\{x_1, x_2, \ldots x_N\}$

$$w^* = X^T \alpha \quad \text{for some } \alpha \in \mathbb{R}^N$$

$(d \times 1) \quad (d \times N) \; (N \times 1)$

(by span property)

This is called as "representer trick"

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \cdots & \cdots & x_{1N} \\ x_{21} & \cdots & & x_{2N} \\ \vdots & & & \\ x_{d1} & \cdots & & x_{dN} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=0}^{N} x_{1i} \alpha_i \\ \sum_{i=0}^{N} x_{2i} \alpha_i \\ \vdots \\ \sum x_{di} \alpha_i \end{bmatrix}$$

(x_{2i} → 2nd dimension)

ie $w_k = \sum_{i=0}^{N} x_{ki} \alpha_i$

(x_{ki}: k^{th} dimension of x)

Representer Theorem

Whenever w comes in dot products, we can use the kernel trick. This is called the representer theorem.

The training data lives in subspace spanned by $\{\phi(x_1), \phi(x_2), \ldots \phi(x_n)\}$

Any component outside the subspace doesn't affect the loss.

Hence w^* too lies in the span

Hence w^* can be written as a linear combination of mapped training points

$$w = \sum_i \alpha_i \phi(x_i)$$

Kernalized Ridge Regression

$$\text{loss} = \|y - Xw\|_2^2 + \lambda \|w\|_2^2 \qquad \text{let } w = X^T\alpha$$

$$\text{loss} = (y - XX^T\alpha)^T (y - XX^T\alpha) + \lambda (X^T\alpha)^T (X^T\alpha)$$

at minima

$$\frac{\partial \text{loss}}{\partial \alpha} = 0$$

$$\frac{\partial L}{\partial \alpha} = -2XX^T(y - XX^T\alpha) + 2\lambda XX^T\alpha = 0$$

$$\therefore \quad (XX^T + \lambda I)\alpha = y$$

$$\therefore \quad \alpha = (XX^T + \lambda I)^{-1} y$$

as we have seen $K = XX^T$

$$\therefore \quad \alpha = (K + \lambda I)^{-1} y$$

for inference on point p

$$\hat{y} = p^T W = \sum_{i=1}^{d} p_i w_i$$

$$\hat{y} = \underset{1\times d}{p^T} \; \underset{d\times N}{X^T} \; \underset{N\times 1}{\alpha}$$

$$= \begin{bmatrix} p_1 & p_2 & \cdots & p_d \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \cdots & \cdots & x_{1N} \\ x_{21} & \cdots & - & & x_{2N} \\ \vdots & & & & \\ x_{d1} & \cdots & \cdots & & x_{dN} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{d} p_i x_{i1} & \sum_{i=1}^{d} p_i x_{i2} & \cdots & \sum_{i=1}^{d} p_i x_{iN} \end{bmatrix} \alpha$$

$$= \begin{bmatrix} p^T x_1 & p^T x_2 & \cdots & p^T x_N \end{bmatrix} \alpha$$

$$= \begin{bmatrix} k(p, x_1) & k(p, x_2) & \cdots & k(p, x_N) \end{bmatrix} \alpha$$

Thus we can avoid all dot products and write in form of k