Ines Pancorbo

Kernel Machines

1. Using the dot product for similarity

- $u^T v = ||u||||v||cos\theta \implies \frac{u^T v}{||u||||v||} = cos\theta \in [-1, 1]$
- We can think u is/is not similar to v in terms of direction, using the above.
- Remember the dot product $u^T v$ is just the projection of vector u onto vector v.

Note: The difference between scalar projection and vector projection.

2. What is a Kernel?

Kernel is a way of computing the dot product of two vectors x and y in some feature space (possibly very high dimensional). Suppose we have a mapping $\phi : R^n \rightarrow R^m$ where $m >> n$ that brings our vectors in R^n to some feature space R^m. Then the dot product in this space is $\phi(x)^T \phi(y)$. A kernel is a function K that corresponds to this dot product, i.e., $K(x, y) = \phi(x)^T \phi(y)$.

Why would this be useful?

Kernels give a way to compute dot products in some feature space (must be Hilbert space as we need the dot product) without even knowing what the space is and what ϕ is.

For example: Consider a simple polynomial kernel $K(x, y) = (1 + x^T y)^2, x, y \in R^2$. This doesn't seem to correspond to any mapping function ϕ, it is just a function that returns a scalar. Assuming $x = (x_1, x_2)$ and $y = (y_1, y_2)$ let us expand this expression.

$K(x, y) = (1 + x^T y)^2 = 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2$. Note this is nothing but a dot product between two vectors $\phi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$ and $\phi(y) = (1, y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2)$. So the Kernel $K(x, y) = (1 + x^T y)^2 = \phi(x)^T \phi(y)$ computes a dot product in 6-dimensional space without explicitly visiting this space or passing $x \in R^2$ to $\phi(x) : R^2 \rightarrow R^6$.

Linear Regression using the Kernel Trick

How do we fit the data?

Example:

Data: $(x, y), x \in R^n, y \in R$.

Consider $\phi(x) : R^n \rightarrow R^m$ i.e., $(x_1, x_2, ..., x_n) \rightarrow (\phi(x_1), \phi(x_2), ..., \phi(x_n))$.

We want $y \sim f(x) = w^T \phi(x)$, which is m dimensional.

First think of an analogy:

Suppose you wanted a linear model, so $y \sim f(x) = \alpha^T x$ for each x. So the space of functions you can fit from is $F = \{f(x) = \alpha^T x\}$, which is a 3 dimensional space.

We would want to $\min_{\alpha \in R^n} ||y - \beta\alpha||^2$, where β is the model matrix.

$$\beta = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \\ x_1^{(N)} & x_2^{(N)} \end{pmatrix}$$

We know how to solve this minimzation problem as it is just a quadratic: $\alpha = (\beta^T \beta)^{-1} \beta^T y$.

F is a small dimensional space (especially when compared to samples N, which we can assume is more than 2) so there would be no need to penalize but if desired our minimization problem and solution would be:

$\min\limits_{\alpha \in R^n} (||y - \beta\alpha||^2 + \lambda||\alpha||^2)$ (just a ridge regression), with solution $\alpha = (\beta^T \beta + \lambda I)^{-1} \beta^T y$.

Now, going back to a kernel machine fit. We are dealing with a much bigger space in terms of dimensions, $F = \{f(x) = w^T \phi(x)\}$.

We would initially want $\min\limits_{w \in R^n} ||y - \beta w||^2$, where β is a $N \times m$ matrix

$$\beta = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{pmatrix}$$

Take the case when $m >> N$, then the minimzation problem we are trying to solve doesn't have a unique solution since we have a fat matrix. And therefore, we have to penalize: $\min\limits_{w \in R^m} (||y - \beta w||^2 + \lambda||w||^2)$ (again, ridge regression). And the solution is $w = (\beta^T \beta + \lambda I)^{-1} \beta^T y$.

** **Note**, using ridge regression because of quadratic optimization. We can use lasso regression and would be dealing with convex optimization.

However, we have a **problem** from a computational point of view:

β is a $N \times m$ matrix $\implies \beta^T beta$ is a $m \times m$ matrix. And so computing the inverse, which requires gaussian elimination would require at least $\frac{n^3}{3}$ multiplications. If m is "big," this may be computationally infeasible. So as F grows in dimension (m grows) we require more computation to fit the data.

This is when the kernel trick (i.e., a realization, method) comes into play.

Kernel Trick

Trick #1:

Consider $\beta^T = (\phi(x^{(1)})\phi(x^{(2)}) \cdots \phi(x^{(N)}))$.

We write $w = \beta^T a + z$. We are taking w and expressing it as a linear combination of the feature vectors, $\phi(x)$, and some z. I.e., we are projecting w onto the span of the $\phi(x^{(i)})$, and adding z, which is orthogonal to each of the $\phi(x^{(i)})$.

It turns out that $w \in \text{span}(\phi(x^{(1)}), \phi(x^{(2)}), \cdots, \phi(x^{(N)}))$ os that $z = 0$. Let us show why this is true.

We want to $\min\limits_{w \in R^m} (||y - \beta w||^2 + \lambda||w||^2)$. Now substitute for w, $\beta^T a + z$. Note $a \in R^N, z \in R^m$.

So we have

$$\min_{w \in R^m} ||y - \beta w||^2 + \lambda||w||^2$$

$$= \min_{a,z} ||y - \beta(\beta^T a + z)||^2 + \lambda||\beta^T a + z||^2$$

$$= \min_{a,z} ||y - (\beta\beta^T a + \beta z)||^2 + \lambda(\beta^T a + z)^T(\beta^T a + z)$$

$$= \min_{a,z} ||y - (\beta\beta^T a + \beta z)||^2 + \lambda(\beta^T a + z)^T(\beta^T a + z)$$

But we have that z is orthogonal to each of the $\phi(x^{(i)})$ so $\beta z = 0$. I.e., I dont need to add a z because in terms of the minimization problem the z has not effect in getting a better fit for y:

$$\min_{a,z} ||y - (\beta\beta^T a + \beta z)||^2 + \lambda(\beta^T a + z)^T(\beta^T a + z)$$

$$= \min_{a,z} ||y - (\beta\beta^T a + \beta z)||^2 + \lambda(a^T\beta^T\beta a + 2a^T\beta z + z^T z)$$

$$= \min_{a,z} ||y - \beta\beta^T a||^2 + \lambda(a^T\beta^T\beta a + z^T z)$$

$$= \min_{a,z} ||y - \beta\beta^T a||^2 + \lambda a^T\beta^T\beta a + \lambda z^T z$$

$$= \min_{a} ||y - \beta\beta^T a||^2 + \lambda a^T\beta^T\beta a$$

Because we want to minimize, and thus we should choose $z = 0$.

Summary:

We can make the following substitution $w = \beta^T a$ and the optimization problem becomes $\min_{a \in R^N} ||y - \beta\beta^T a||^2 + \lambda a^T\beta^T\beta a$. So we go from an optimization problem in R^m to an optimization in R^N even though we are trying to fit $y \sim w^T\phi(x)$ and $w \in R^m$. This means that our above problem is no longer a problem.

Now consider the matrix $\beta\beta^T$:

$$\beta\beta^T = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{pmatrix} \begin{pmatrix} \phi(x^{(1)}) & \phi(x^{(2)}) & \cdots & \phi(x^{(N)}) \end{pmatrix}$$

Trick #2:

The above matrix is the kernel matrix, i.e, $K = \beta\beta^T$. Note that the entry $K_{ij} = \phi(x^{(i)})^T\phi(x^{(j)})$. Now we come to another problem, storing β, β^T will be an issue as well because of the dimension of the $\phi(x^{(i)}) \in R^m$.

Now, the second part of the kernel trick is that we do not have to store the matrices $\beta, beta^T$. Note the kernel matrix is $N \times N$. To compute a all we need is the dot products of the $\phi(x^{(i)})$, i.e., the Kernel matrix where $K_{ij} = \phi(x^{(i)})^T\phi(x^{(j)})$.

Summary:

data: $(x_{(i)}, y_i), x_{(i)} \in R^n, i = 1, 2, \cdots, N$.

We use a feature map, $x \in R^n \to \phi(x) \in R^m$.

We want $y \sim f(x) = w^T\phi(x)$ where $f \in F$ and $dim(F)$ "big."

And to find w we want to $\min_{w \in R^m}(||y - \beta w||^2 + \lambda||w||^2)$. Then I let $w = \beta^T a$ and so what I want is $\min_{a \in R^N} ||y - \beta\beta^T a||^2 + \lambda a^T\beta^T\beta a$. So we have a more reasonable optimization. There is a further simplification: we let $K = \beta\beta^T$ and $K_{ij} = \phi(x^{(i)})^T\phi(x^{(j)})$. Thus, $\min_{a \in R^N} ||y - Ka||^2 + \lambda a^T Ka$. Now, this is a quadratic and thus the solution is $a = (K^T K + \lambda K^{-1})K^T y$. This can be further simplified: Note K is symmetric since the dot product is commutative:

$$\begin{aligned}
a &= (K^T K + \lambda K^{-1})K^T y \\
a &= (K(K + \lambda I))^{-1} K y \\
a &= (K + \lambda I)^{-1} K^{-1} K y \\
a &= (K + \lambda I)^{-1} y
\end{aligned}$$

Quick checkpoint: Given new x what is predicted $y = w^T \phi(x)$?

Answer:

$$y = w^T \phi(x) = (\beta^T a)\phi(x) = a^T \beta \phi(x) = a^T \begin{bmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{bmatrix} \phi(x) = a^T \begin{bmatrix} \phi(x^{(1)})^T \phi(x) \\ \phi(x^{(2)})^T \phi(x) \\ \phi(x^{(3)})^T \phi(x) \\ \vdots \\ \phi(x^{(N)})^T \phi(x) \end{bmatrix}$$

Now let us go over two examples:

Example 1:

data: $(x^{(i)}, y_i), x^{(i)} \in R^n, i = 1, 2, \cdots, N$.

We use a feature map, $x \in R^n \rightarrow \phi(x) = (1, x_1, x_2, \cdots, x_n, x_1^2, x_1 x_2, \cdots, x_1 x_n, x_2^2, x_2 x_3, \cdots, x_n^2) \in R^{2n+1+n(n-1)/2}$.

$\phi(x)$ can be very high dimensional, but computing $\phi(x^{(i)}) \cdot \phi(x^{(j)})$ is easy.

We can notice that $\phi(x^{(i)}) \cdot \phi(x^{(j)}) = (1 + x^{(i)} \cdot x^{(j)})^2$. Computationally we prefer to do a dot product in R^n than in $R^{2n+1+n(n-1)/2}$. So no we can write dot products for $\phi(x) \in R^{2n+1+n(n-1)/2}$ based on dot products of corresponding $x \in R^n$.

Now, we want $y \sim f(x) = w^T \phi(x)$. We compute K, with $K_{ij} = \phi(x^{(i)}) \cdot \phi(x^{(j)}) = (1 + x^{(i)} \cdot x^{(j)})^2$

$$K = \begin{pmatrix} (1 + x^{(1)} \cdot x^{(1)})^2 & (1 + x^{(1)} \cdot x^{(2)})^2 & \cdots & (1 + x^{(1)} \cdot x^{(N)})^2 \\ \vdots & \vdots & \vdots & \vdots \\ (1 + x^{(N)} \cdot x^{(1)})^2 & (1 + x^{(N)} \cdot x^{(2)})^2 & \cdots & (1 + x^{(N)} \cdot x^{(N)})^2 \end{pmatrix}$$

Now K will give us $a \in R^N$ used to find $w \in R^{2n+1+n(n-1)/2}$, which is $a = \beta^T w$. I.e., then $a = (K + \lambda I)^{-1} y$. And so $w = \beta^T$. But we **never write** w **explicitly.** I.e., given new x, $y = w^T \phi(x) = a^T(\beta \phi(x))$ and $\beta \phi(x)$ is also handled through $\phi(x^{(i)}) \cdot \phi(x) = (1 + x^{(i)} \cdot x)^2$.

Comments:

(1) Applying kernels to logistic regression (explored in HW 15)

(2) Support Vector Machines (explored in course on Machine Learning)

(3) Radial Basis Kernel (explored in course on Machine Learning)