9. Kernel Methods

The kernelized version of a method can be inserted in many methods seen up to now.

<u>So far</u>:

Objects represented as fixed length feature-vectors $x \in \mathbb{R}^n$ or $\phi(x)$.

<u>Issue</u>:

What about objects with variable length or infinite dimensions? $\rightarrow$ (strings, trees, image features, time-series...)

The idea is to not represent explicitly the instances, but represent a function of the instances.

Replace instances with a kernel function, which measures the similarity of two instances.

SIMILARITY MEASURE (any function) $k(x, x') \geq 0$ between the objects x, x'.
$\downarrow$
kernel function

It maps a pair of instances into the real numbers.

<u>Kernel function</u>: a real-valued function $k(x, x') \in \mathbb{R}$, for $x, x' \in \mathcal{X}$, where $\mathcal{X}$ is some abstract space.

Requirements for k:

- SYMMETRIC $k(x, x') = k(x', x)$
- NON-NEGATIVE $k(x, x') \geq 0$

} In some cases they are not needed

x, x' can belong to my input space, to a transformation $\phi(x)$, in general they can belong to an abstract space.

We can define some kernels:

LINEAR	POLYNOMIAL	RBF Radial Basis function	SIGMOID
(dot product) $K(\vec{x},\vec{x}') = \vec{x} \cdot \vec{x}^T$	$K(\vec{x},\vec{x}') = (\beta \vec{x}^T \vec{x}' + \gamma)^d$, $d \in \{2,3,\dots\}$	$K(\vec{x},\vec{x}') = \exp(-\beta \lvert \vec{x}-\vec{x}' \rvert^2)$	$K(\vec{x},\vec{x}') = \tanh(\beta \vec{x}^T \vec{x}' + \gamma)$

In many of the linear models the input vectors multiply each other $(\dots \vec{x}\,\vec{x}^T \dots)$ so you can replace them with some kernel, nothing change, you have just expressed the method in terms of K (KERNEL TRICK)

Consider a linear model $y(\vec{x},\vec{w}) = \vec{w}^T \vec{x}$ with a dataset D, where $D = \{(\vec{x}_i, t_i)_{i=1}^{N}\}$ We want to minimize $J(\vec{w})$:

$$J(\vec{w}) = (\vec{t} - X\vec{w})^T (\vec{t} - X\vec{w}) + \lambda \|\vec{w}\|^2$$

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix} \text{ design matrix} \qquad \vec{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} \text{ output vector}$$

OPT. SOL. $= \hat{\vec{w}} = (XX^T + \lambda I_N)^{-1} \vec{t}$,

$$\text{then } \hat{\vec{w}} = X^T \vec{\alpha} = \sum_{n=1}^{N} \alpha_n \vec{x}_n$$

Hence we have $y(\vec{x}; \hat{\vec{w}}) = \hat{\vec{w}}^T x = \sum_{n=1}^{N} \alpha_n \vec{x}_n^T \vec{x}$.

If we consider a linear kernel $K(x,x') = \vec{x}^T \vec{x}'$ we can rewrite the model.

$$\vec{\alpha} = \left(K + \lambda I_N\right)^{-1} \vec{t}, \text{ where } \underbrace{K = X X^T}$$

$$y(\vec{x}, \hat{\vec{w}}) = \sum_{n=1}^{N} \alpha_n K(\vec{x}_n, \vec{x})$$

is also called GRAM MATRIX
is formed by evaluation of the kernel for each pair

- Linear model with linear kernel $K(\vec{x}, \vec{x}') = \vec{x}^T \vec{x}'$

$$y(\vec{x}, \vec{\alpha}) = \sum_{n=1}^{N} \alpha_n \vec{x}_n^T \vec{x}$$

Solution : $\vec{\alpha} = (K + \lambda I_N)^{-1} \vec{t}$

Gram Matrix

$$K = \begin{bmatrix} \vec{x}_1^T \vec{x}_1 & \cdots & \vec{x}_1^T \vec{x}_N \\ \vdots & & \\ \vec{x}_N^T \vec{x}_1 & \cdots & \vec{x}_N^T \vec{x}_N \end{bmatrix}$$

- Linear model with any kernel K

$$y(\vec{x}, \vec{\alpha}) = \sum_{n=1}^{N} \alpha_n K(\vec{x}_n, \vec{x})$$

Solution : $\vec{\alpha} = (K + \lambda I_N)^{-1} \vec{t}$

Gram matrix

$$K = \begin{bmatrix} K(\vec{x}_1, \vec{x}_1) & \cdots & K(\vec{x}_1, \vec{x}_N) \\ \vdots & \ddots & \\ K(\vec{x}_N, \vec{x}_1) & \cdots & K(\vec{x}_N, \vec{x}_N) \end{bmatrix}$$

Kernel trick or kernel substitution:

$\boxed{\text{If input vector } \vec{x} \text{ appears in an algorithm only in the form of an INNER PRODUCT, REPLACE THE INNER PRODUCT WITH some kernel.}}$

- Can be applied to any $\vec{x}$ (even infinite size)
- No need to know $\phi(\vec{x})$
- Directly extend many well-known algorithms

This solution is similar to the transformation of input space, $\phi(\vec{x})$, $k(\vec{x}', \vec{x}) = \phi(\vec{x})^T \phi(\vec{x}')$, in general you may not know ϕ, with kernel you avoid this problem. $\boxed{\text{NOTE}}$

It is interesting to consider now SVM. The solution of SVM for classification has the following form:

$$\boxed{\hat{\vec{w}} = \sum_{i=1}^{N} \alpha_n \vec{x}_n}$$

α_i = computed as result of an optimization problem.

Given this solution, the linear model is:

$$y(\vec{x}, \hat{\vec{w}}) = \text{sign}\left(w_0 + \sum_i \alpha_i \underbrace{\vec{x}_i^{\,T} \vec{x}}\right)$$

we can replace this with the kernel

Kernel trick:

$$y(\vec{x}, \hat{\vec{w}}) = \text{sign}\left(w_0 + \sum_{i=1} \alpha_i \, k(\vec{x}_i, \vec{x})\right)$$

In this new model with general kernel function, also the Lagrangian problem change, but it can be solved.

We don't need to change anything else, we have just to replace the kernel, substituting the inner product.

• Kernelized linear regression

Let's consider a generalized model for regression, with an error function based on the sum of the errors:

$$\boxed{E(y_i, t_i) = (y_i - t_i)^2}$$

It is an error function that measure the difference between the prediction y_i and what we have in the dataset t_i

Linear model for regression:

$y = \vec{w}^T \vec{x}$, $D = \{(\vec{x}_i, t_i)_{i=1}^N\}$

minimize the regularized loss function:

$$J(\vec{w}) = \sum_{i=1}^{N} E(y_i, t_i) + \lambda \|\vec{w}\|^2$$

usual regulariz. factor

In this case the solution is:

$$\hat{\vec{w}} = (X^T X + \lambda I_M)^{-1} X^T \vec{t} = \dots = X^T \alpha$$

predictions are made using:

$$y(\vec{x}, \hat{\vec{w}}) = \sum_{i=1}^{N} \hat{\alpha}_i \vec{x}_i^T \vec{x} \xrightarrow[\text{kernelized}]{} \boxed{y(\vec{x}, \hat{\vec{w}}) = \sum_{i=1}^{N} \alpha_i K(\vec{x}_i, x)}$$

$$\vec{\alpha} = (K + \lambda I_N)^{-1} \vec{t}$$

There is a problem! α is NOT SPARSE, most of them are not zero, so this problem may be affected by outliers, overfitting and can be computationally expensive.

The idea of KERNELIZE SVM for REGRESSION is to use another error function, that makes possible to have a sparse solution.

Let's consider:

$$J(\vec{w}) = C \sum_{n=1}^{N} E_\epsilon(y_n, t_n) + \frac{1}{2} \|\vec{w}\|^2$$

C is another constant, not the same place of λ, the meaning is the same, it indicates how much importance you want to give to the error.

C is the inverse of λ and ε-insensitive error function

$$E_\varepsilon(y,t) = \begin{cases} 0 & \text{if } |y-t|<\varepsilon \\ |y-t|-\varepsilon & \text{otherwise} \end{cases}$$

C has the goal of measuring the contribution of the error in the optimization function

the E_ε tells you that any prediction that is within ε will not count as an error.

Introduce slack variables $\xi_i^+, \xi_i^- \geq 0$

$$\left.\begin{aligned} t_i &\leq y_i + \varepsilon + \xi_i^+ \\ t_i &\geq y_i - \varepsilon - \xi_i^- \end{aligned}\right\} \varepsilon\text{-tube}$$

Points inside the ε-tube:

$$\boxed{y_i - \varepsilon \leq t_i \leq y_i + \varepsilon} \Rightarrow \xi_i = 0 \leftarrow \text{the slacks are zero}$$

$$\xi_i^+ > 0 \Rightarrow t_i > y_i + \varepsilon$$
$$\xi_i^- > 0 \Rightarrow t_i < y_i - \varepsilon$$

slack variables are placed for each point in the dataset

Loss function can be rewritten:

$$J(\vec{w}) = C \sum_{i=1}^{N} (\xi_i^+ - \xi_i^-) + \frac{1}{2}\|\vec{w}\|^2$$

subject to this constraints:

$$t_i \leq y(\vec{x}_i, \vec{w}) + \varepsilon + \xi_i^+$$
$$t_i \geq y(\vec{x}_i, \vec{w}) - \varepsilon - \xi_i^-$$
$$\xi_i^+, \xi_i^- \geq 0$$

It is easy to solve it by quadratic programming

Associate to a Lagrange problem, where $\vec{x}$ appears only in the kernel function (details are not important).

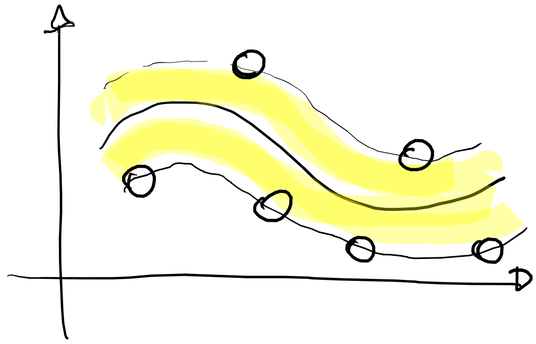
From the Lagrangian we compute $\hat{a}_i$, $\hat{a}_j'$ (sparse values, most of them are zero).

We can extend KKT (Karush - Kuhn Tucker) condition that says that among these values $\hat{a}_i$, they are greater than zero only when:

$\hat{a}_i > 0 \Rightarrow \varepsilon + \xi_i + y_i - t_i = 0$
(data points lies on or above ε-tube upper boundary)

$\hat{a}_i > 0 \Rightarrow \varepsilon + \xi_i - y_i + t_i = 0$
(data points lies on or below the ε-tube lower boundary)

All data points inside the ε-tube have $\hat{a}_i = 0$ and $\hat{a}_i' = 0$ and thus not contribute to the prediction.



The model is computed to a limited set of samples, so it is robust to outliers, better in overcome overfitting (same property of SVM)

(+) Kernel methods overcome difficulties in defining non-linear models

(+) Kernel SVM is one of the most effective ML method for classification and regression

(-) SVM requires model selection and hyper-parameters tuning