Kernel methods

What are kernels?

In the previous chapters we have been dealing with input spaces of known and fixed dimensions such as $X \in R^M$, but in real situation this is not always the case. In face we often have to deal with input spaces of variable length and possibly infinite dimensions like strings, image features etc.

This is when kernel functions come into play. Fundamentally they are functions that given two inputs, return a real value that represents theyr similarity. The following is a formal definition of kernels:

A kernel is a real-valued function $k(x, x') \in \mathbb{R}$ for $x, x' \in \mathcal{X}$ (where $\mathcal{X}$ is some abstract space).

Kernels tipycally satisfy this conditions:

- They are symmetrical: $k(x, x') = k(x', x)$
- They are non-negative: $k(x, x') \geq 0$

The more the two inputs are similiar, the more the output value of the kernel will be close to zero.

The first thing to do when using kernel function is to normalize our dataset. In fact we want our transformations to be independent from the measure units of our inputs. For example the same dataset that uses a measure in centimeters will be scaled differently with respect to the same dataset measured in meters if they are not normalized. Instead, we want to preserve the concept and not care about the scale, thus making our solution independent from the measures.

The most common normalizetion techniques are:

- Min-max: $\bar{x} = \frac{x - min}{max - min}$

- Normalization (or standardization): $\bar{x} = \frac{x-\mu}{\sigma}$, where μ is the mean and σ is the standard deviation of the dataset.

Some common kernel families are:

- Linear: $k(x, x') = x^T x'$
- Polynomial: $k(x, x') = (\beta x^T x' + \gamma)^d \;\; d \in \{1, 2, 3, ...\}$
- Radial basis function: $k(x, x') = e^{-\beta |x - x'|^2}$
- Sigmoid: $k(x, x') = tanh(\beta x^T x' + \gamma)$
- *In general we can define any custom kernel that best fits our data.*

How to use kernels

Let's start by considering the generic definition of a linear model $y(x; w) = W^T X$. Our goal is to minimize the loss function:

$$J(w) = (t - x_w)^T (t - x_w) + \lambda ||W||^2$$

We know that the optimal solution for our weights is:

$$\hat{w} = X^T (X X^T + \lambda I_N)^{-1} t$$

In order to simplify notation, we define $\alpha = (X X^T + \lambda I_N)^{-1} t$. Now the optimal solution can be expressedas a linear combination of terms:

$$\hat{w} = X^T \alpha = \sum_{n=1}^{N} \alpha_n x_n$$

Hence, the solution for our model will be:

$$y(x; \hat{w}) = \hat{W}^T X = \sum_{n=1}^{N} \alpha_n \mathbf{x_n^T x}$$

Notice that we highlited the term $\mathbf{x_n^T x}$ beacuse this can actually be subsituted by the kernel function $k(x, x') = x^T x$. Applying the kernel function, we will obtain the following result:

$$y(x;\hat{w}) = \hat{W}^T X = \sum_{n=1}^{N} \alpha_n k(x_n, x)$$

The huge benefit of using a kernel for our model is that once we computed the solution it will be independent from the dimensions of the input.

The kernel trick

As long as the input vector x appears in our algorithm only in the form of an inner product $x^T x$, we can replace that term with any kernel $k(x, x')$.

Kernelized SVMs

Recalling the definition of Support Vector Machines, we know that our model is formulated as:

$$y(x;\alpha) = sign(w_0 + \sum_{n=1}^{N} \alpha_n \mathbf{x_n^T x})$$

Having our noticeable term, we know that we can apply the kernel trick:

$$y(x;\alpha) = sign(w_0 + \sum_{n=1}^{N} \alpha_n k(x_n, x))$$

Subsequently, we can express our Lagrangian optimization problem as:

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} a_n a_m t_n t_m \mathbf{k}(\mathbf{x_n}, \mathbf{x_m})$$

Hence, the solution will be

$$w_0^* = \frac{1}{|SV|} \sum_{x_k \in SV} (t_k - \sum_{x_j \in SV} a_j^* t_k \mathbf{k}(\mathbf{x_k}, \mathbf{x_j}))$$

Why use kernelized SVMs?

The solution for the classical formulation of the linear model without SVMs implies computing the Graham matrix $K = X^T X$, which grows quadratically with the size of our datasets, since it computes the product for every possible pair of inputs (grows as$|D|^2$). By contrast, we know that SVMs only require the support vectors of the dataset, that will be a small subset of D.

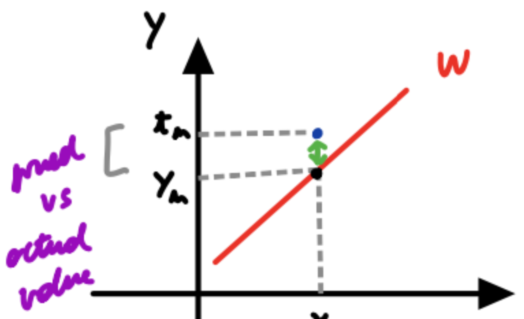
Kernelized regression

Let's start considering the classical formulation of linear regression $y(x) = W^T X$. We define our loss function as:

$$J(w) = \sum_{n=1}^{N} E(y_n, t_n) + \lambda ||W||^2$$

The idea is to achieve a formulation of the model where we can apply the kernel trick.

At the moment the formulation of the error is the quadratic distance between the predicted value and the actual one $E(y_n, t_n) = (y_n - t_n)^2$:



The solution for the current formulation is:

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

Our model will be $y(x; \hat{w}) = \sum_{n=1}^{N} \alpha_n \mathbf{x_n^T} \mathbf{x}$

Notice that this form admits the application of the kernel trick. Hence, the kernelized linear model is

$$y(x; \hat{w}) = \sum_{n=1}^{N} \alpha_n \mathbf{k}(\mathbf{x_n}, \mathbf{x})$$

Notice that the actual formulation of the problem still suffers the fact that it's computationally heavy due to the Graham matrix computation.

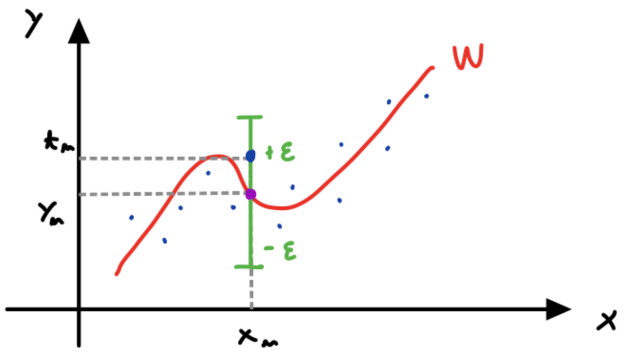
Kernelized SVMs for regression

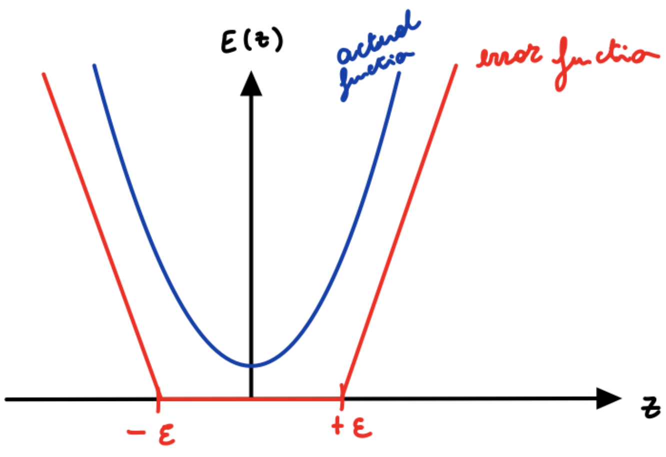
A less computationally expensive approach would be applyin the concept of SVMs to regression.

Our first step will be defining the error function as an ϵ-insensitive linear function:

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

The idea here is that we define an interval $(y-\epsilon, y+\epsilon)$ such that if the real value falls inside this interval, the error will be zero. Substantially, we are making the error less sensitive to noise.



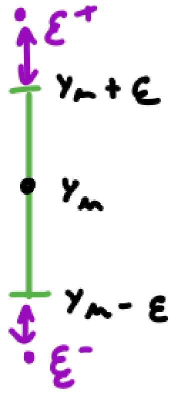


The new formulation for our loss function will be:

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

We still have a major problem given by the current formulation of the error. In fact it’s easy to notice that the error function is not differentiable, meaning that computing the result would be very difficoult. We have to think about a manipulation aimed at solving this problem.

We introduce two slack variables $\xi_n^+, \xi_n^- \geq 0$. They will measure by how much a point falls outside the admitted interval. In particular their value will be 0 if the points falls inside the interval, otherwise they will return the distance between the point and the edge of the interval.



By using this formulation, we can express the loss function by just taking in account the slack variables:

$$J(w) = C\sum_{n=1}^{N}(\xi^+ + \xi^-) + \frac{1}{2}||W||^2$$

With the constraints:

$$\begin{gathered} t_n \leq y(x_n; w) + \epsilon + \xi^+ \\ t_n \geq y(x_n; w) - \epsilon - \xi^- \\ \xi^+ \geq 0 \\ \xi^- \leq 0 \end{gathered}$$

We got to the form of a standard quadratic programming problem.

❗ It is important to notice that the choice for the value of ϵ really important. In fact it will regulate how much noise is acceptable. A good choice is to choose it based on the measure of the precision for our dataset.

Finally, the resulting model after the computation of optimal weights is:

$$y(x) = \sum_{n=1}^{N}(\hat{\alpha}_n - \hat{\alpha}'_n)k(x, x_n) + \hat{w}_0$$

An important property is that the KKT condition is still valid, meaning that the zero terms (every point that falls inside the admitted intervall) will not contribute to the computation.