Machine Learning — Statistical Methods for Machine Learning

Kernel functions

Instructor: *Nicolò Cesa-Bianchi* version of June 16, 2019

In practice, linear predictors may exhibit a high test error. In most cases this is due to underfitting rather than overfitting, which is an indication that the Bayes optimal predictor is hardly linear and therefore linear predictors suffer from high bias.

A popular technique to reduce this bias is feature expansion, which constructs new features by nonlinear combinations of the base features. Formally, this can be viewed in terms of a function $\boldsymbol{\phi} : \mathbb{R}^d \to \mathcal{V}$ mapping data points $\boldsymbol{x} \in \mathbb{R}^d$ to a higher-dimensional space $\mathcal{V}$. This mapping helps reduce the bias because linear classifiers in $\mathcal{V}$ correspond to nonlinear classifiers in $\mathbb{R}^d$.

For example, consider the quadratic feature-expansion map $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^6$ defined by $\boldsymbol{\phi}(x_1, x_2) = \big(1, x_1^2, x_2^2, x_1, x_2, x_1x_2\big)$. Recall that a homogeneous hyperplane in $\mathbb{R}^6$ with coefficients given by $\boldsymbol{w} = (w_1, \dots, w_6)$ is the set of points $\big\{\boldsymbol{z} \in \mathbb{R}^6 \,:\, \boldsymbol{w}^\top \boldsymbol{z} = 0\big\}$. Now note that $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) = w_1 + w_2x_1^2 + w_3x_2^2 + w_4x_1 + w_5x_2 + w_6x_1x_2$. The sets $\big\{\boldsymbol{x} \in \mathbb{R}^2 \,:\, w_1 + w_2x_1^2 + w_3x_2^2 + w_4x_1 + w_5x_2 + w_6x_1x_2 = 0\big\}$ describe second-degree surfaces in the Euclidean bidimensional space. These surfaces include ellipses, parabolas, and hyperbolas.

In general, we may consider polynomial feature expansion maps $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^N$ that use features of the form $\prod_{s=1}^k x_{v_s}$ for all $\boldsymbol{v} \in \{1, \dots, d\}^k$ and for all $k = 0, 1, \dots, n$ (the previous example is a special case for $d = 2$ and $n = 2$). Fix such a $\boldsymbol{\phi}$ and consider the classifier $h : \mathbb{R}^d \to \{-1, 1\}$ defined by

$$h(\boldsymbol{x}) = \operatorname{sgn}\big(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})\big) \qquad \text{where} \quad \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^N w_i \, \phi(\boldsymbol{x})_i$$

This nonlinear classifier in $\mathbb{R}^d$ corresponds to a linear classifier in the space

$$\boldsymbol{\phi}\big(\mathbb{R}^d\big) \equiv \Big\{\boldsymbol{z} \in \mathbb{R}^N \,:\, (\exists \boldsymbol{x} \in \mathbb{R}^d)\, \boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{z}\Big\} \ .$$

Note, however, that

$$N = \sum_{k=0}^n \big|\{1, \dots, d\}^k\big| = \sum_{k=0}^n d^k = \frac{d^{n+1} - 1}{d - 1}$$

This implies that $N = \Theta(d^n)$ is exponential in the degree n, and computing $\boldsymbol{\phi}$ becomes infeasible even for moderately large n.

This computational barrier can be fully sidestepped using the so-called **kernel trick**, which can be applied to many algorithms for learning linear predictors. For example, recall the Perceptron update rule $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t\boldsymbol{x}_t$ where $\boldsymbol{w}_1 = \boldsymbol{0}$. Then, the linear classifiers learned through the Perceptron are of the form

$$h(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{s \in S} y_s \boldsymbol{x}_s^\top \boldsymbol{x}\right)$$

where S is the set of indices s of training examples $(\boldsymbol{x}_s, y_s)$ on which the Perceptron made an update. If we run the Perceptron in the space $\boldsymbol{\phi}(\mathbb{R}^d)$, the linear classifier h becomes

$$h_{\boldsymbol{\phi}}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{s \in S} y_s\, \boldsymbol{\phi}(\boldsymbol{x}_s)^\top \boldsymbol{\phi}(\boldsymbol{x})\right) .$$

Hence, in order to compute $h_{\boldsymbol{\phi}}(\boldsymbol{x})$ quickly we need a way of computing quickly each term $\boldsymbol{\phi}(\boldsymbol{x}_s)^\top \boldsymbol{\phi}(\boldsymbol{x})$. The kernel trick helps us find an efficiently computable kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}') \qquad \text{per ogni } \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d. \tag{1}$$

For example, the quadratic kernel, corresponding to the quadratic feature-expansion map

$$\boldsymbol{\phi}(x_1, x_2) = \big(1, x_1^2, x_2^2, \sqrt{2}\, x_1, \sqrt{2}\, x_2, \sqrt{2}\, x_1 x_2\big)$$

is $K(\boldsymbol{x}, \boldsymbol{x}') = \big(1 + \boldsymbol{x}^\top \boldsymbol{x}'\big)^2$, as one can easily verify (the presence of the $\sqrt{2}$ coefficients, which is needed for the math, does not change the class of linear predictors that are representable using the mapping).

Given a kernel K, we can then write the linear classifier generated by the Perceptron as

$$h_K(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{s \in S} y_s K(\boldsymbol{x}_s, \boldsymbol{x})\right) . \tag{2}$$

Below here, we give the pseudo-code of the Kernel Perceptron algorithm.

```
Algorithm: **Kernel Perceptron**
Let $S$ be the empty set.
Per ogni $t = 1, 2, \dots$
    1. Get next example $(\boldsymbol{x}_t, y_t)$
    2. Compute $\widehat{y}_t = \operatorname{sgn}\left(\sum_{s \in S} y_s\, K(\boldsymbol{x}_s, \boldsymbol{x}_t)\right)$
    3. If $\widehat{y}_t \neq y_t$ add $t$ to $S$
```

The polynomial kernel $K_n(\boldsymbol{x}, \boldsymbol{x}') = \big(1 + \boldsymbol{x}^\top \boldsymbol{x}'\big)^n$ for all $n \in \mathbb{N}$ generalizes the quadratic kernel defined earlier. Using Newton's Binomial Theorem, we can explicitely compute the map $\boldsymbol{\phi}_n$ tale che $K_n(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}_n(\boldsymbol{x})^\top \boldsymbol{\phi}_n(\boldsymbol{x}')$,

$$\big(1 + \boldsymbol{x}^\top \boldsymbol{x}'\big)^n = \sum_{k=0}^{n} \binom{n}{k} \big(\boldsymbol{x}^\top \boldsymbol{x}'\big)^k . \tag{3}$$

Now observe that

$$\big(\boldsymbol{x}^\top \boldsymbol{x}'\big)^k = \left(\sum_{i=1}^{d} x_i x_i'\right)^k = \sum_{\boldsymbol{v} \in \{1, \dots, d\}^k} \left(\prod_{s=1}^{k} x_{v_s} x_{v_s}'\right) .$$

Therefore,

$$\boldsymbol{\phi}_n(\boldsymbol{x}) = \left(\sqrt{\binom{n}{k}} \prod_{s=1}^{k} x_{v_s} \right)_{k=0,\dots,n,\ \boldsymbol{v}\in\{1,\dots,d\}^k} . \tag{4}$$

In other words, the feature map $\boldsymbol{\phi}_n : \mathbb{R}^d \to \mathbb{R}^N$ associated with the polynomial kernel K_n sends each $\boldsymbol{x} \in \mathbb{R}^d$ to a vector whose components are indexed by all monomials $\prod_{s=1}^{k} x_{v_s}$ of degree at most n and weighted by the roots of the binomial coefficients.

Another type of kernel is the Gaussian kernel,

$$K_\gamma(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\gamma} \left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2 \right) \qquad \gamma > 0 \ .$$

In order to derive the map $\boldsymbol{\phi}_\gamma$ associated with K_γ we proceeed as follows,

$$\begin{aligned}
\exp\left(-\frac{1}{2\gamma} \left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2 \right) &= \exp\left(-\frac{1}{2\gamma} \left(\|\boldsymbol{x}\|^2 + \left\|\boldsymbol{x}'\right\|^2 \right) \right) \exp\left(\frac{1}{\gamma} \left(\boldsymbol{x}^\top \boldsymbol{x}' \right) \right) \\
&= \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2\gamma} \right) \exp\left(-\frac{\|\boldsymbol{x}'\|^2}{2\gamma} \right) \sum_{n=0}^{\infty} \frac{1}{n!} \frac{\left(\boldsymbol{x}^\top \boldsymbol{x}' \right)^n}{\gamma^n}
\end{aligned} \tag{5}$$

where we used the Taylor series expansion $e^x = 1 + x + \frac{x^2}{2!} + \cdots$. A closer look at (5) reveals that the Gaussian kernel is a linear combination of infinitely many polynomial kernels (3) of increasing degree, each weighted by the reciprocal of the factorial of its degree. The parameter γ is a scaling factor for the products $\boldsymbol{x}^\top \boldsymbol{x}'$. Finally, the factors $e^{-\|\boldsymbol{x}\|^2/(2\gamma)} e^{-\|\boldsymbol{x}'\|^2/(2\gamma)}$ normalize with respect to $\boldsymbol{x}$ and $\boldsymbol{x}'$ giving $K_\gamma(\boldsymbol{x}, \boldsymbol{x}) = 1$ for each $\boldsymbol{x} \in \mathbb{R}^d$.

Note that predictors of the form (2) that utilize Gaussian kernels predict any point $\boldsymbol{x}$ using a linear combination (with coefficients y_t) of Gaussians $e^{-z^2/(2\gamma)}$ centered on $\boldsymbol{x}_s$ for $s \in S$ and evaluated at $\boldsymbol{x}$.

The Gaussian kernel is *universal* in the following sense: for each $\gamma > 0$, for each continuous function $f : \mathbb{R}^d \to \mathbb{R}$, and for each $\varepsilon > 0$, there exists $g \in \mathcal{H}_\gamma$, with

$$\mathcal{H}_\gamma = \left\{ \sum_{i=1}^{N} \alpha_i \, K_\gamma(\boldsymbol{x}_i, \cdot) \ : \ \boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{R}^d,\ \alpha_1, \dots, \alpha_N \in \mathbb{R},\ N \in \mathbb{N} \right\} \tag{6}$$

such that h approximates f with error bounded by ε. An important consequence of this fact is that learning algorithms that use Gaussian kernels are potentially consistent (that is, the expected risk of their predictors converges to the Bayes risk as the sample size grows to infinity).

A learning algorithm producing classifiers of the form $h(\boldsymbol{x}) = \operatorname{sgn}\big(g(\boldsymbol{x})\big)$, with $g \in \mathcal{H}_\gamma$ defined in (6), is nonparametric. Despite h being a linear classifier, the Gaussian kernel is a linear combination of infinitely many polynomial kernels, and therefore $\mathcal{H}_\gamma$ is infinite-dimensional.

Given a data space $\mathcal{X}$ (not necessarily $\mathbb{R}^d$) and a symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ how can we check whether K is a kernel? In other words, we want to know whether there exist a feature map $\phi_K : \mathcal{X} \to \mathcal{H}_K$ and an inner product $\langle \cdot, \cdot \rangle_K$ in $\mathcal{H}_K$ such that $\big\langle \phi_K(\boldsymbol{x}), \phi_K(\boldsymbol{x}') \big\rangle_K = K(\boldsymbol{x}, \boldsymbol{x}')$. Luckily, there is a very simple way of checking that: K is a kernel if and only if for all $m \in \mathbb{N}$ and for all $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m \in \mathcal{X}$, the $m \times m$ matrix $\boldsymbol{K}$ such that $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive semidefinite. That is, $\boldsymbol{z}^\top \boldsymbol{K} \boldsymbol{z} \geq 0$ for all $\boldsymbol{z} \in \mathbb{R}^m$.

The above result tells us the conditions under which ϕ_K exists for a given K. On the other hand, it does not tell us how ϕ_K looks like. In fact, there is no unique representation of the pair $\phi_K, \mathcal{H}_K$ for a given kernel K. However, it can be shown that if K is a kernel, then we can always represent ϕ_K as $\phi_K(\boldsymbol{x}) = K(\boldsymbol{x}, \cdot)$ e $\mathcal{H}_K$ as

$$\mathcal{H}_K \equiv \left\{ \sum_{i=1}^{N} \alpha_i \, K(\boldsymbol{x}_i, \cdot) \, : \, \boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathcal{X}, \, \alpha_1, \dots, \alpha_N \in \mathbb{R}, \, N \in \mathbb{N} \right\}$$

which is a generalization of the space (6) for Gaussian kernels.

Since ϕ_K is the feature map for K, we must have $\langle \phi_K(\boldsymbol{x}), \phi_K(\boldsymbol{x}') \rangle_K = K(\boldsymbol{x}, \boldsymbol{x}')$. Therefore, we also know that $\langle \phi_K(\boldsymbol{x}), \phi_K(\boldsymbol{x}') \rangle_K = K(\boldsymbol{x}, \boldsymbol{x}')$. Now, recalling that the inner product is a bilinear operator, we can compute the inner product between $\phi_K(\boldsymbol{x}) = K(\boldsymbol{x}, \cdot)$ and any $f \in \mathcal{H}_K$,

$$\langle f, K(\boldsymbol{x}, \cdot) \rangle_K = \sum_{i=1}^{N} \alpha_i \langle K(\boldsymbol{x}_i, \cdot), K(\boldsymbol{x}, \cdot) \rangle_K = \sum_{i=1}^{N} \alpha_i \, K(\boldsymbol{x}_i, \boldsymbol{x}) = f(\boldsymbol{x})$$

The equality $\langle f, K(\boldsymbol{x}, \cdot) \rangle_K = f(\boldsymbol{x})$ is important and is known as *reproducing property*. For this reason, $\mathcal{H}_K$ is also known as *reproducing kernel Hilbert space* (RKHS). Note that, whereas a linear predictor $\boldsymbol{w} \in \mathbb{R}^d$ predicts using $\boldsymbol{w}^\top \boldsymbol{x}$, a linear predictor $f \in \mathcal{H}_K$ predicts using $\langle f, \phi_K(\boldsymbol{x}) \rangle_K = f(\boldsymbol{x})$.

Next, we see how the inner product $\langle f, g \rangle_K$ between two arbitrary $f, g \in \mathcal{H}_K$ is computed, where

$$f = \sum_{i=1}^{N} \alpha_i \, K(\boldsymbol{x}_i, \cdot) \qquad \text{and} \qquad g = \sum_{j=1}^{M} \alpha_j \, K(\boldsymbol{x}'_j, \cdot)$$

Using once more the bilinearity of inner products,

$$\begin{aligned}
\langle f, g \rangle_K &= \left\langle \sum_{i=1}^{N} \alpha_i \, K(\boldsymbol{x}_i, \cdot), \sum_{j=1}^{M} \beta_j \, K(\boldsymbol{x}'_j, \cdot) \right\rangle_K \\
&= \sum_{i=1}^{N} \alpha_i \left\langle K(\boldsymbol{x}_i, \cdot), \sum_{j=1}^{M} \beta_j \, K(\boldsymbol{x}'_j, \cdot) \right\rangle_K \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \beta_j \langle K(\boldsymbol{x}_i, \cdot), K(\boldsymbol{x}'_j, \cdot) \rangle_K \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \beta_j K(\boldsymbol{x}_i, \boldsymbol{x}'_j)
\end{aligned}$$

We can lift to any RKHS the bounds we derived for learning algorithms that can be run with kernels. For instance recall the bound on the number of mistakes provided by the Perceptron convergence theorem,

$$\|\boldsymbol{u}\|^2 \left(\max_t \|\boldsymbol{x}\|^2 \right)$$

which holds for any $\boldsymbol{u} \in \mathbb{R}^d$ such that $y_t \boldsymbol{u}^\top \boldsymbol{x}_t \geq 1$ for $t = 1, \dots, m$.

In a generic RKHS $\mathcal{H}_K$, the linear separator $\boldsymbol{u}$ is some $g \in \mathcal{H}_K$ such that $y_t\, f(\boldsymbol{x}_t) \geq 1$ for $t = 1, \dots, m$. The squared norm $\|\boldsymbol{x}_t\|^2 = \boldsymbol{x}_t^\top \boldsymbol{x}_t$ becomes $\|\phi_K(\boldsymbol{x})\|_K^2 = \big\langle K(\boldsymbol{x}, \cdot), K(\boldsymbol{x}, \cdot) \big\rangle_K = K(\boldsymbol{x}, \boldsymbol{x})$. Finally $\|\boldsymbol{u}\|^2$ is replaced by

$$\|f\|_K^2 = \left\| \sum_{i=1}^{N} \alpha_i\, K(\boldsymbol{x}_i, \cdot) \right\|_K^2 = \left\langle \sum_{i=1}^{N} \alpha_i\, K(\boldsymbol{x}_i, \cdot), \sum_{j=1}^{N} \alpha_j\, K(\boldsymbol{x}_j, \cdot) \right\rangle_K = \sum_{i,j=1}^{N} \alpha_i \alpha_j\, K(\boldsymbol{x}_i, \boldsymbol{x}_j)\ .$$

Linear predictors in RKHS can incur overfitting. For example, by increasing the degree n of a polynomial kernel K_n we reduce the training error because higher-degree curves can be used to separate the training points. If the degree is too high, the predictor will overfit. A similar reasoning applies to Gaussian kernels K_γ. The γ parameter corresponds to the width of the Gaussians centered on training points $\boldsymbol{x}_s$. If γ is small relatively to the typical squared distances $\|\boldsymbol{x}_s - \boldsymbol{x}\|^2$ between training and test points, then the classification of a test point $\boldsymbol{x}$ is essentially determined by the training point closest to it. This implies a training error equal or close to zero, because —similarly to 1-NN classifiers— the training points are almost never misclassified. Once again, the resulting predictor is likely to overfit. On the other hand, for values of γ that are large with respect to the squared distances $\|\boldsymbol{x}_s - \boldsymbol{x}\|^2$, the Gaussians centered on training points are very wide, and the resulting predictors are similar to k-NN classifiers when k is chosen close to the training set size, which is likely to cause underfitting.

We established that any symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if the kernel matrix $\boldsymbol{K}$ is positive semidefinite. This is an important result, because it holds irrespective to the choice of the data space $\mathcal{X}$. We can therefore define kernels on any set $\mathcal{X}$: be it a set of matrices, sequences, trees, graphs, and so on. Kernels can be viewed as a way to encapsulate the data space, offering a uniform interface to a learning algorithm that can be efficiently run in the corresponding RKHS. In order to guide the intuition when designing a kernel function on a given data space $\mathcal{X}$, one should recall that $K(x, x')$ implements an inner product between $K(x, \cdot)$ and $K(x', \cdot)$ in some RKHS. We can then start by defining a notion of similarity on $\mathcal{X}$, and then refine it until arbitrary kernel matrices are proven to be positive semidefinite.