9.2.3 Kernel Methods

We have all the tools together now to make an exciting step. Let us summarize our findings. We are interested in regularized estimation problems of the form (9.5), where $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) + b$ is linear, examples include the soft margin SVM and MAP for logistic regression. Here is a mad idea. Suppose we use a huge number of features p, maybe even infinitely many. Before figuring out how this could be done, let us first see whether this makes any sense in principle. After all, we have to store $\boldsymbol{w} \in \mathbb{R}^p$ and evaluate $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^p$. Do we? In the previous section, we learned that we can always *represent* $\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} \in \mathbb{R}^n$, and our dataset is finite. Moreover, the error function in (9.5) depends on

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w} + b = \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha} + b$$

only, and $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ is just an $\mathbb{R}^{n\times n}$ matrix. Finally, the Tikhonov regularizer is given by

$$\frac{\nu}{2}\|\boldsymbol{w}\|^2 = \frac{\nu}{2}\left\|\boldsymbol{\Phi}^T\boldsymbol{\alpha}\right\|^2 = \frac{\nu}{2}\boldsymbol{\alpha}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha},$$

it also only depends on $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$. Finally, once we are done and found $(\boldsymbol{\alpha}_*, b_*)$, where $\boldsymbol{w}_* = \boldsymbol{\Phi}^T\boldsymbol{\alpha}_*$, we can predict on new inputs $\boldsymbol{x}$ with

$$y^*(\boldsymbol{x}) = \boldsymbol{w}_*^T\boldsymbol{\phi}(\boldsymbol{x}) + b_* = \boldsymbol{\alpha}_*^T\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x}) + b_*.$$

We need finite quantities only in order to make our idea work, namely the matrix $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ during training, and the mapping $\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})$ for predictions later on, to be evaluated at finitely many $\boldsymbol{x}$. This is the basic observation which makes *kernelization* work.

The entries of $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ are $\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)$, while $[\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})] = [\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x})]$. We can write

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}'),$$

a *kernel function*. It is now clear that given the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$, we never need to access the underlying $\boldsymbol{\phi}(\boldsymbol{x})$. In fact, we can forget about the dimensionality p and vectors of this size altogether. What makes $K(\boldsymbol{x}, \boldsymbol{x}')$ a kernel function? It must be the inner product in some feature space, but what does that imply? Let us work out some properties. First, a kernel function is obviously symmetric: $K(\boldsymbol{x}', \boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}')$. Second, consider some arbitrary set $\{\boldsymbol{x}_i\}$ of n input points and construct the *kernel matrix* $\boldsymbol{K} = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)] \in \mathbb{R}^{n\times n}$. Also, denote $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_n)]^T \in \mathbb{R}^{n\times p}$. Then,

$$\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} = \boldsymbol{\alpha}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha} = \left\|\boldsymbol{\Phi}^T\boldsymbol{\alpha}\right\|^2 \geq 0.$$

In other words, the kernel matrix $\boldsymbol{K}$ is positive semidefinite (see Section 6.3). This property defines kernel functions. *$K(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel function if the kernel matrix $\boldsymbol{K} = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$ for any finite set of points $\{\boldsymbol{x}_i\}$ is symmetric positive semidefinite.* An important subfamily are the *infinite-dimensional* or *positive definite kernel functions*. A member $K(\boldsymbol{x}, \boldsymbol{x}')$ of this subfamily is defined by all its kernel matrices $\boldsymbol{K} = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$ being positive definite for any set $\{\boldsymbol{x}_i\}$ of any size. In particular, all kernel matrices are invertible. As we will see shortly, it is positive definite kernel functions which give rise to infinite-dimensional feature spaces, therefore to nonlinear kernel methods.

Problem: Can I use astronomically many features p? How about $p = \infty$?
Approach: No problem! As long as you can efficiently compute the *kernel function* $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$, the representer theorem saves the day.

Hilbert Spaces and All That (*)

Before we give examples of kernel functions, a comment for meticulous readers (all others can safely skip this paragraph and move to the examples). How can we even talk about $\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x})$ if $p = \infty$? Even worse, what is $\boldsymbol{\Phi} \in \mathbb{R}^{n \times p}$ in this case? In the best case, all this involves infinite sums, which may not converge. Rest assured that all this can be made rigorous within the framework of Hilbert function and functional spaces. In short, infinite dimensional vectors become functions, their transposes become functionals, and matrices become linear operators. A key result is Mercer's theorem for positive semidefinite kernel functions, which provides a construction for a feature map. However, with the exception of certain learning-theoretical questions, the importance of all this function space mathematics for down-to-earth machine learning is very limited. Historically, the point about the efforts of mathematicians like Hilbert, Schmidt and Riesz was to find conditions under which function spaces could be treated in the same simple way as finite-dimensional vector spaces, working out analogies for positive definite matrices, quadratic functions, eigendecomposition, and so on. Moreover, function spaces governing kernel methods are of the particularly simple reproducing kernel Hilbert type, where common pathologies like "delta functions" do not even arise. You may read about all that in [39] or other kernel literature, it will not play a role in this course. Just one warning which you will not find spelled out much in the SVM literature. The "geometry" in huge or infinite-dimensional spaces is dramatically different from anything we can draw or imagine. For example, in Mercer's construction of $K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j \geq 1} \phi_j(\boldsymbol{x}) \phi_j(\boldsymbol{x}')$, the different feature dimensions $j = 1, 2, \ldots$ are by no means on equal terms, as far as concepts like distance or volume are concerned. For most commonly used infinite-dimensional kernel functions, the contributions $\phi_j(\boldsymbol{x}) \phi_j(\boldsymbol{x}')$ rapidly become extremely small, and only a small number of initial features determine most of the predictions. A good intuition about kernel methods is that they behave like (easy to use) linear methods of flexible dimensionality. As the number of data points n grows, a larger (but finite) number of the feature space dimensions will effectively be used.

Examples of Kernel Functions

Let us look at some examples. Maybe the simplest kernel function is $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$, the standard inner product. Moreover, for any finite-dimensional feature map $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^p$ ($p < \infty$), $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$ is a kernel function. Since any kernel matrix of this type can at most have rank p, such kernel functions are positive semidefinite, but not positive definite. However, even for finite-dimensional kernels, it can be much simpler to work with $K(\boldsymbol{x}, \boldsymbol{x}')$ directly than to evaluate $\boldsymbol{\phi}(\boldsymbol{x})$. For example, recall polynomial regression estimation from Section 4.1, giving rise to a polynomial feature map $\boldsymbol{\phi}(x) = [1, x, \ldots, x^r]^T$ for $x \in \mathbb{R}$. Now, if $\boldsymbol{x} \in \mathbb{R}^d$ is multivariate, a corresponding polynomial feature

map would consists of very many features. Is there a way around their explicit representation? Consider the *polynomial kernel*

$$K(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}'\right)^r = \sum_{j_1, \ldots, j_r} \left(x_{j_1} \ldots x_{j_r}\right) \left(x'_{j_1} \ldots x'_{j_r}\right).$$

For example, if $d = 3$ and $r = 2$, then

$$K(\boldsymbol{x}, \boldsymbol{x}') = (x_1 x'_1 + x_2 x'_2 + x_3 x'_3)^2 = x_1^2 (x'_1)^2 + x_2^2 (x'_2)^2 + x_3^2 (x'_3)^2 + 2(x_1 x_2)(x'_1 x'_2) + 2(x_1 x_3)(x'_1 x'_3) + 2(x_2 x_3)(x'_2 x'_3),$$

a feature map of which is

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ \sqrt{2} x_1 x_2 \\ \sqrt{2} x_1 x_3 \\ \sqrt{2} x_2 x_3 \end{bmatrix}.$$

If $\boldsymbol{x} \in \mathbb{R}^d$, $K(\boldsymbol{x}, \boldsymbol{x}')$ is evaluated in $O(d)$, independent of r. Yet it is based on a feature map $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{d^r}$, whose dimensionality[6] scales exponentially in r. A variant is given by

$$K(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}' + \varepsilon\right)^r, \quad \varepsilon > 0,$$

which can be obtained by replacing $\boldsymbol{x}$ by $[\boldsymbol{x}^T, \sqrt{\varepsilon}]^T$ above. The feature map now runs over all subranges $1 \le j_1 \le \cdots \le j_k \le d$, $0 \le k \le r$.

A frequently used infinite-dimensional (positive definite) kernel is the *Gaussian* (or radial basis function, or squared exponential) kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\tau}{2} \|\boldsymbol{x} - \boldsymbol{x}'\|^2}, \quad \tau > 0. \tag{9.6}$$

We establish it as a kernel function in Section 9.2.4. The Gaussian is an example of a stationary kernel, these depend on $\boldsymbol{x} - \boldsymbol{x}'$ only. We can weight each dimension differently:

$$K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{1}{2} \sum_{j=1}^d \tau_j (x_j - x'_j)^2}, \quad \tau_1, \ldots \tau_d > 0.$$

Free parameters in kernels are hyperparameters, much like C in the soft margin SVM or the noise variance σ^2 in Gaussian linear regression, choosing them is a model selection problem (Chapter 10).

Choosing the right kernel is much like choosing the right model. In order to do it well, you need to know your options. Kernels can be combined from others in many ways, [6, ch. 6.2] gives a good overview. It is also important to understand statistical properties implied by a kernel. For example, the Gaussian kernel produces extremely smooth solutions, while other kernels from the Matérn family are more flexible. Most books on kernel methods will provide some overview, see also [41].

One highly successful application domain for kernel methods concerns problems where input points $\boldsymbol{x}$ have combinatorial structure, such as chains, trees, or

[6] More economically, we can run over all $1 \le j_1 \le \cdots \le j_r \le d$.

graphs. Applications range from bioinformatics over computational chemistry to structured objects in computer vision. The rationale is that it is often simpler and much more computationally efficient to devise a kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ than a feature map $\boldsymbol{\phi}(\boldsymbol{x})$. This field was seeded by independent work of David Haussler [22] and Chris Watkins.

A final remark concerns normalization. As noted in Chapter 2 and above in Section 9.1, it is often advantageous to use normalized feature maps $\|\boldsymbol{\phi}(\boldsymbol{x})\| = 1$. What does this mean for a kernel?

$$K(\boldsymbol{x}, \boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}) = 1.$$

Therefore, a kernel function gives rise to a normalized feature map if its diagonal entries $K(\boldsymbol{x}, \boldsymbol{x})$ are all 1. For example, the Gaussian kernel (9.6) is normalized. Moreover, if $K(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel, then so is

$$\frac{K(\boldsymbol{x}, \boldsymbol{x}')}{\sqrt{K(\boldsymbol{x}, \boldsymbol{x}) K(\boldsymbol{x}', \boldsymbol{x}')}}$$

(see Section 9.2.4), and the latter is normalized. It is a good idea to use normalized kernels in practice.

9.2.4 Techniques: Properties of Kernels (*)

In this section, we review a few properties of kernel functions and look at some more examples. The class of kernel functions has formidable closedness properties. If $K_1(\boldsymbol{x}, \boldsymbol{x}')$ and $K_2(\boldsymbol{x}, \boldsymbol{x}')$ are kernels, so are cK_1 for $c > 0$, $K_1 + K_2$ and $K_1 K_2$. You will have no problem confirming the the first two. The third is shown at the end of this section. Moreover, $f(\boldsymbol{x}) K_1(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}')$ is a kernel function as well, for any $f(\boldsymbol{x})$. This justifies kernel normalization, as discussed at the end of Section 9.2.3. If $K_r(\boldsymbol{x}, \boldsymbol{x}')$ is a sequence of kernel functions converging pointwise to $K(\boldsymbol{x}, \boldsymbol{x}') = \lim_{r \to \infty} K_r(\boldsymbol{x}, \boldsymbol{x}')$, then $K(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel function as well. Finally, if $K(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel and $\boldsymbol{\psi}(\boldsymbol{y})$ is some mapping into $\mathbb{R}^d$, then $(\boldsymbol{y}, \boldsymbol{y}') \mapsto K(\boldsymbol{\psi}(\boldsymbol{y}), \boldsymbol{\psi}(\boldsymbol{y}'))$ is a kernel as well.

Let us show that the Gaussian kernel (9.6) is a valid kernel function. First, $(\boldsymbol{x}^T \boldsymbol{x}')^r$ is a kernel for every $r = 0, 1, 2, \ldots$, namely the polynomial kernel from Section 9.2.3. By the way, $K(\boldsymbol{x}, \boldsymbol{x}') = 1$ is a kernel function, since its kernel matrices $\mathbf{1}\mathbf{1}^T$ are positive semidefinite. Therefore,

$$K_r(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=0}^{r} \frac{1}{j!} (\boldsymbol{x}^T \boldsymbol{x}')^j$$

are all kernels, and so is the limit $e^{\boldsymbol{x}^T \boldsymbol{x}'} = \lim_{r \to \infty} K_r(\boldsymbol{x}, \boldsymbol{x}')$. More general, if $K(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel, so is $e^{K(\boldsymbol{x}, \boldsymbol{x}')}$. Now,

$$e^{-\frac{\tau}{2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2} = e^{-\frac{\tau}{2}\|\boldsymbol{x}\|^2} e^{\tau \boldsymbol{x}^T \boldsymbol{x}'} e^{-\frac{\tau}{2}\|\boldsymbol{x}'\|^2}.$$

The middle is a kernel, and we apply our normalization rule with $f(\boldsymbol{x}) = e^{-\frac{\tau}{2}\|\boldsymbol{x}\|^2}$. The Gaussian kernel is infinite-dimensional (positive definite), although we will not show this here.

Another way to think about kernels is in terms of *covariance functions*. A random process is a set of random variables $a(\boldsymbol{x})$, one for each $\boldsymbol{x} \in \mathbb{R}^d$. Its covariance function is

$$K(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}[a(\boldsymbol{x}), a(\boldsymbol{x}')] = \mathrm{E}\left[(a(\boldsymbol{x}) - \mathrm{E}[a(\boldsymbol{x})])(a(\boldsymbol{x}') - \mathrm{E}[a(\boldsymbol{x}')])\right].$$

Covariance functions are kernel functions. For some set $\{\boldsymbol{x}_i\}$, let $\boldsymbol{a} = [a(\boldsymbol{x}_i) - \mathrm{E}[a(\boldsymbol{x}_i)]] \in \mathbb{R}^n$ be a random vector. Then, for any $\boldsymbol{v} \in \mathbb{R}^n$:

$$\boldsymbol{v}^T \boldsymbol{K} \boldsymbol{v} = \boldsymbol{v}^T \mathrm{E}\left[\boldsymbol{a}\boldsymbol{a}^T\right] \boldsymbol{v} = \mathrm{E}\left[(\boldsymbol{v}^T \boldsymbol{a})^2\right] \geq 0.$$

Finally, there are some symmetric functions which are not kernels. One example is

$$K(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left(\alpha \boldsymbol{x}^T \boldsymbol{x}' + \beta\right).$$

In an attempt to make SVMs look like multi-layer perceptrons, this non-kernel was suggested and is shipped to this day in many SVM toolboxes[7]. Running the SVM with "kernels" like this spells trouble. Soft margin SVM is a convex optimization problem only if kernel matrices are positive semidefinite, codes will typically crash if that is not the case. A valid "neural networks" kernel is found in [47], derived from the covariance function perspective.

Finally, why is $K_1(\boldsymbol{x}, \boldsymbol{x}')K_2(\boldsymbol{x}, \boldsymbol{x}')$ a kernel? This argument is for interested readers only, it can be skipped at no loss. We have to show that for two positive semidefinite kernel matrices $\boldsymbol{K}_1, \boldsymbol{K}_2 \in \mathbb{R}^{n \times n}$ the Schur (or Hadamard) product $\boldsymbol{K}_1 \circ \boldsymbol{K}_2 = [K_1(\boldsymbol{x}_i, \boldsymbol{x}_j)K_2(\boldsymbol{x}_i, \boldsymbol{x}_j)]$ (Section 2.4.3) is positive semidefinite as well. To this end, we consider the Kronecker product $\boldsymbol{K}_1 \otimes \boldsymbol{K}_2 = [K_1(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{K}_2] \in \mathbb{R}^{n^2 \times n^2}$. This is positive semidefinite as well. Namely, we can write $\boldsymbol{K}_1 = \boldsymbol{V}_1 \boldsymbol{V}_1^T$, $\boldsymbol{K}_2 = \boldsymbol{V}_2 \boldsymbol{V}_2^T$, then

$$\boldsymbol{K}_1 \otimes \boldsymbol{K}_2 = (\boldsymbol{V}_1 \otimes \boldsymbol{V}_2)(\boldsymbol{V}_1 \otimes \boldsymbol{V}_2)^T.$$

But the Schur product is a square submatrix of $\boldsymbol{K}_1 \otimes \boldsymbol{K}_2$, a so called minor. In other words, for some index set $J \subset \{1, \ldots, n^2\}$ of size $|J| = n$: $\boldsymbol{K}_1 \circ \boldsymbol{K}_2 = (\boldsymbol{K}_1 \otimes \boldsymbol{K}_2)_J$, so that for any $\boldsymbol{v} \in \mathbb{R}^n$:

$$\boldsymbol{v}^T(\boldsymbol{K}_1 \circ \boldsymbol{K}_2)\boldsymbol{v} = \boldsymbol{z}^T(\boldsymbol{K}_1 \otimes \boldsymbol{K}_2)\boldsymbol{z} \geq 0, \quad \boldsymbol{z} = \boldsymbol{I}_{\cdot,J}\boldsymbol{v} \in \mathbb{R}^{n^2}.$$

The same proof works to show that the positive definiteness of $\boldsymbol{K}_1$, $\boldsymbol{K}_2$ implies the positive definiteness of $\boldsymbol{K}_1 \circ \boldsymbol{K}_2$, a result due to Schur.

9.2.5 Summary

Let us summarize the salient points leading up to support vector machine binary classification. We started with the observation that for a linearly separable dataset, many different separating hyperplanes result in zero training error. Among all those potential solutions, which could arise as outcome of the perceptron algorithm, there is one which exhibits maximum stability against small displacements of patterns $\boldsymbol{x}_i$, by attaining the maximum margin $\gamma_{\mathcal{D}}(\boldsymbol{w}, b)$. Pursuing this lead, we addressed a number of problems:

[7] It is even on Wikipedia (`en.wikipedia.org/wiki/Support_vector_machine`).