Kernel methods

Recall that a binary linear classifier $h : X \rightarrow \{-1, +1\}$ takes the following form:

$$h(x) = \begin{cases} 
+1 & w^T x > 0 \\
-1 & w^T x \leq 0
\end{cases}$$

Just as we saw how the basic version of linear regression can fail when tasked with approximating an inherently nonlinear target function, linear classifiers will not perform well if the true separating boundary is not a hyperplane. A simple illustrative example is the two-class data set in which class +1 is defined to be the set of points contained in a ball of radius $R$ centered about the origin and class −1 surrounds this ball but does not intersect it.

Example: Another example is the XOR function from Boolean logic. In this case, our feature space is $X = \{(x_1, x_2) : x_1, x_2 \in \{0, 1\}\} = \{0, 1\}^2$ and $Y = \{0, 1\}$. The target function is $f(x_1, x_2) = x_1 + x_2 \pmod{2}$. You should verify for yourself that it is impossible to separate the class $y = 0$ from $y = 1$ using a linear decision boundary. To overcome this obstacle we can define the nonlinear transformation $\phi : \{0, 1\}^2 \rightarrow \{0, 1\}^3$ by

$$\phi(x_1, x_2) = (x_1, x_2, x_1x_2).$$

We now check that the weight vector $w = (1, 1, -2)$ along with the redefined classifier $h(x) = w^T \phi(x)$ successfully replicates the target function $f$. There are four cases:

1. $x_1 = x_2 = 0$. Then $w^T \phi(x) = 0 = f(x)$.
2. $x_1 = x_2 = 1$. Then $w^T \phi(x) = 0 = f(x)$.
3. $x_1 = 0$, $x_2 = 1$. Then $w^T \phi(x) = 1 = f(x)$.
4. $x_1 = 1$, $x_2 = 0$. Then $w^T \phi(x) = 1 = f(x)$.
As was the case with linear regression, choosing an appropriate feature transform \( \phi \) can allow us to achieve sophisticated nonlinear decision boundaries using the simple linear machinery of SVMs and perceptrons.

**Example:** The \( D \)th order polynomial transform \( \Phi_D \) maps the input space \( X \) to the set of all monomials of the input features with at most degree \( D \). For example, if \( x = (x_1, x_2, x_3) \) and \( D = 2 \), then

\[
\Phi_2(x) = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3).
\]

If \( L \) is the number of features, it can be shown that the number of monomials in the \( D \)th order polynomial transform is roughly \( \binom{D+L}{L} \). Now recall the digit recognition example, for which the input space is \( X = \mathbb{R}^L \), where \( L = 784 \) for a 28 \( \times \) 28 pixel image. Then even restricting ourselves to degree 3 polynomials will yield on the order of a billion features in the transformed space. This is a bad sign for both generalization (we will need a huge amount of data to avoid overfitting) and computation (learning a billion weights is costly).

**Kernels in support vector machines**

The example above illustrates that mapping to higher dimensional spaces is useful in that it allows us to implement more sophisticated decision boundaries, but that this benefit comes at a high price. We will now see how we can extend support vector machines to implement nonlinear decision boundaries while avoiding the problems above.

Let \( T = \{(x_1, y_1), \ldots, (x_N, y_N)\} \) be a training set where \( x_i \in \mathbb{R}^D \) and \( y_i \in \{-1, +1\} \). Let \( \phi : \mathbb{R}^D \to \mathbb{R}^M \) be a feature transformation. We choose the optimal weight vector \( w^* \) of a binary SVM by minimizing an objective function that incorporates the transform \( \phi \):

\[
w^* = \arg\min_w \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \max\{0, 1 - y_i w^T \phi(x_i)\}.
\]

It can be shown that \( w^* \) can be represented as a linear combination of \( \phi(x_1), \ldots, \phi(x_N) \). Thus, instead of optimizing over all possible weights \( w \in \mathbb{R}^M \), we can solve for coefficients \( a \in \mathbb{R}^N \) satisfying

\[
w = \sum_{i=1}^N a_i \phi(x_i).
\]
Note that we can write
\[ \|w\|^2 = w^T w = \left( \sum_{i=1}^{N} a_i \phi(x_i) \right)^T \left( \sum_{i=1}^{N} a_i \phi(x_i) \right) = \sum_{i=1}^{N} \sum_{j=1}^{N} a_ia_j \phi(x_i)^T \phi(x_j). \]  

**Definition:** We say a function \( k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R} \) is a **kernel** if it is of the form \( k(x, z) = \phi(x)^T \phi(z) \) for some function \( \phi \). Now define the **Gram matrix** \( M \in \mathbb{R}^{N \times N} \) by letting \( M_{ij} = \phi(x_i)^T \phi(x_j) = k(x_i, x_j) \).

Then equation (1) can be rewritten as
\[ \|w\|^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} a_ia_j M_{ij}. \]

The vector of coefficients \( a^* \) that minimizes the loss function is therefore given by
\[ a^* = \arg\min_a \frac{1}{2} \sum_{i,j} a_ia_j M_{ij} + C \sum_{i=1}^{N} \max \left\{ 0, 1 - y_i \sum_{j=1}^{N} a_j M_{ji} \right\}. \]  

At this point it may not be clear what has been gained by converting the problem into an optimization over \( a \) and introducing the Gram matrix \( M \). The most important thing to realize is that the feature transform \( \phi \) only appears in equation (2) via its role in the entries of \( M \). In particular, the loss function in (2) only depends on \( \phi \) via the scalar products \( \phi(x_i)^T \phi(x_j) = M_{ij} \). To understand why this matters, we need to discuss properties of kernel functions.

**What is the kernel \( k \)?**

Let's look at a specific example of a kernel function. The kernel \( k : \mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R} \) defined by
\[ k(x, z) = (x^T z + 1)^D \]  
is known as the **degree-\( D \) polynomial kernel**. To show that \( k \) is a genuine kernel, we need to exhibit some function \( \phi \) satisfying \( k(x, z) = \phi(x)^T \phi(z) \).
Suppose for concreteness that $L = D = 2$. Then

$$k(x, z) = (x^T z + 1)^2$$
$$= (x_1 z_1 + x_2 z_2 + 1)^2$$
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 1 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2$$
$$= (1, \sqrt{2} x_1, \sqrt{2} x_2, \sqrt{2} x_1 x_2, x_1^2, x_2^2) \cdot (1, \sqrt{2} z_1, \sqrt{2} z_2, \sqrt{2} z_1 z_2, z_1^2, z_2^2)$$
$$= \phi(x)^T \phi(z),$$

where

$$\phi(x) = (1, \sqrt{2} x_1, \sqrt{2} x_2, \sqrt{2} x_1 x_2, x_1^2, x_2^2).$$

Thus, $k$ can in fact be represented as

$$k(x, z) = \phi(x)^T \phi(z), \quad (4)$$

satisfying the definition of a kernel function above. Note that $\phi(x)$ is a vector that contains an entry for each monomial of degree $\leq D = 2$.

Thus we have shown the computations (3) and (4) are equivalent (at least for $D = L = 2$, but the result holds more generally). Computing the kernel in (3) only requires work proportional to the dimensionality of the input space; all we need to do is take a scalar product and raise the result to some power. The dimensionality $D$ of the transformed space has no effect on the cost of computing the kernel! Indeed, we need not even compute $\phi(x)$ explicitly at any point in the learning process, since $\phi$ only plays a role via the kernel.

**Putting it all together**

To use the kernel trick in soft-margin SVMs, we follow these steps.

1. Given a training set $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, we construct the $N \times N$ Gram matrix $M$ given by

$$M_{ij} = k(x_i, x_j)$$

for some choice of kernel $k$. As we saw with the polynomial kernel, computing $k$ is easy even if $\phi$ (implicitly) maps to a very high-dimensional space.

2. We then solve for $a^*$ using quadratic programming on the objective function in equation (2). The computational bottleneck is now the size $N$ of the data set.
3. To classify \( x \), compute

\[
w^T \phi(x) = \sum_{i=1}^{N} a_i^* k(x_i, x)
\]

and take the class label to be the sign of this number.

**Radial basis kernel**

The radial basis kernel has the form

\[
k(x, z) = \exp \left\{ -\frac{\|x - z\|^2}{2\sigma^2} \right\},
\]

where \( \sigma \in \mathbb{R} \) is a tunable parameter. (By the way, the parameter for the polynomial kernel was the degree \( D \).) Since \( k \) is a kernel, it can also be written as

\[
k(x, z) = \phi(x)^T \phi(z).
\]

It turns out that in this case, \( \phi : X \to \mathbb{R}^\infty \). In other words, the radial basis kernel involves an implicit feature transform to an infinite dimensional space!

After we find the optimal coefficient vector \( a^* \), we classify a new data point \( x \) by computing

\[
w^T \phi(x) = \sum_{i} a_i k(x, x_i)
\]

\[
= \sum_{i} a_i \exp \left\{ -\frac{\|x - x_i\|^2}{2\sigma^2} \right\}
\]

and taking the sign of the result. From this equation, we can see that \( x \) is classified by examining the value at \( x \) of the sum of many Gaussian-like bumps, each of which is centered at \( x_i \).