Kernels II

Well-defined kernels

Here are the most common kernels:

- *Linear*: $\mathsf{k}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
- *RBF*: $\mathsf{k}(\mathbf{x}, \mathbf{z}) = e^{-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2}}$
- *Polynomial*: $\mathsf{k}(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^d$

Kernels built by recursively combining one or more of the following rules are called *well-defined kernels*:

1. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
2. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = c\mathsf{k}_1(\mathbf{x}, \mathbf{z})$
3. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = \mathsf{k}_1(\mathbf{x}, \mathbf{z}) + \mathsf{k}_2(\mathbf{x}, \mathbf{z})$
4. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = g(\mathsf{k}(\mathbf{x}, \mathbf{z}))$
5. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = \mathsf{k}_1(\mathbf{x}, \mathbf{z})\mathsf{k}_2(\mathbf{x}, \mathbf{z})$
6. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\mathsf{k}_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$
7. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = e^{\mathsf{k}_1(\mathbf{x}, \mathbf{z})}$
8. $\mathsf{k}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{A} \mathbf{z}$

where k_1, k_2 are well-defined kernels, $c \geq 0$, g is a polynomial function with positive coefficients, f is any function and $\mathbf{A} \succeq 0$ is positive semi-definite. Kernel being well-defined is equivalent to the corresponding kernel matrix, K, being *positive semidefinite* (not proved here), which is equivalent to any of the following statement:

1. All eigenvalues of K are non-negative.
2. $\exists$ real matrix P s.t. $K = P^\top P$.
3. $\forall$ real vector $\mathbf{x}, \mathbf{x}^\top K \mathbf{x} \geq 0$.

It is trivial to prove that linear kernel and polynomial kernel with integer d are both well-defined kernel.

Theorem. *The RBF kernel* $k(\mathbf{x}, \mathbf{z}) = e^{\frac{-(\mathbf{x}-\mathbf{z})^2}{\sigma^2}}$ *is a well-defined kernel matrix.*

Quiz1: Prove it!

Proof.

$$\mathsf{k}_1(\mathbf{x},\mathbf{z}) = \mathbf{x}^\top \mathbf{z} \qquad \text{well defined by rule 1}$$
$$\mathsf{k}_2(\mathbf{x},\mathbf{z}) = \frac{2}{\sigma^2}\mathsf{k}_1(\mathbf{x},\mathbf{z}) = \frac{2}{\sigma^2}\mathbf{x}^\top \mathbf{z} \qquad \text{well defined by rule 2}$$
$$\mathsf{k}_3(\mathbf{x},\mathbf{z}) = e^{\mathsf{k}_2(\mathbf{x},\mathbf{z})} = e^{\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2}} \qquad \text{well defined by rule 7}$$
$$\mathsf{k}_4(\mathbf{x},\mathbf{z}) = e^{-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}}\mathsf{k}_3(\mathbf{x},\mathbf{z})e^{-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2}} = e^{-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}} e^{\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2}} e^{-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2}} \qquad \text{well defined by rule 6 with } f(\mathbf{x}) = e^{-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}}$$
$$= e^{\frac{-\mathbf{x}^\top \mathbf{x} + 2\mathbf{x}^\top \mathbf{z} - \mathbf{z}^\top \mathbf{z}}{\sigma^2}} = e^{\frac{-(\mathbf{x}-\mathbf{z})^2}{\sigma^2}} = \mathsf{k}_{RBF}(\mathbf{x},\mathbf{z})$$

■

You can even define kernels of sets, or strings or molecules.

Theorem. *The following kernel is defined on any two sets $S_1, S_2 \subseteq \Omega$,*

$$\mathsf{k}(S_1, S_2) = e^{|S_1 \cap S_2|}.$$

Quiz2: Prove it!

Proof. List out all possible samples Ω and arrange them into a sorted list. We define a vector $\mathbf{x}_S \in \{0,1\}^{|\Omega|}$, where each of its element indicates whether a corresponding sample is included in the set S. It is easy to prove that

$$\mathsf{k}(S_1, S_2) = e^{x_{S_1}^\top x_{S_2}},$$

which is a well-defined kernel by rules 1 and 7. ■

Kernel Machines

(In practice) an algorithm can be kernelized in 2 steps:

1. Prove that the solution lies in the span of the training points (i.e. $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ for some α_i)
2. Rewrite the algorithm and the classifier so that all training or testing inputs $\mathbf{x}_i$ are only accessed in inner-products with other inputs, e.g. $\mathbf{x}_i^\top \mathbf{x}_j$.
3. Define a kernel function and substitute $\mathsf{k}(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i^\top \mathbf{x}_j$.

Kernelized Linear Regression

Recap

Vanilla Ordinary Least Squares Regression (OLS) [also referred to as linear regression] minimizes the following squared loss regression loss function,

$$\min_{\mathbf{w}} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2,$$

to find the hyper-plane $\mathbf{w}$. The prediction at a test-point is simply $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$.
If we let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and $\mathbf{y} = [y_1, \ldots, y_n]^\top$, the solution of OLS can be written in closed form:

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y} \qquad (5)$$

Kernelization

We begin by expressing the solution $\mathbf{w}$ as a linear combination of the training inputs

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i = \mathbf{X}\vec{\alpha}.$$

We derived in the previous lecture that such a vector $\vec{\alpha}$ must always exist by observing the gradient updates that occur if (5) is minimized with gradient descent and the initial vector is set to $\mathbf{w}_0 = \vec{0}$ (because the squared loss is convex the solution is independent of its initialization.)

Similarly, during testing a test point is only accessed through inner-products with training inputs:

$$h(\mathbf{z}) = \mathbf{w}^\top \mathbf{z} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^\top \mathbf{z}.$$

We can now immediately kernelize the algorithm by substituting $k(\mathbf{x}, \mathbf{z})$ for any inner-product $\mathbf{x}^\top \mathbf{z}$. It remains to show that we can also solve for the values of α in closed form. As it turns out, this is straight-forward.

Theorem. *Kernelized ordinary least squares has the solution* $\vec{\alpha} = \mathbf{K}^{-1}\mathbf{y}$.

Proof.

$$\begin{aligned}
\mathbf{X}\vec{\alpha} = \mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y} \quad &| \text{ multiply from left by } \mathbf{X}^\top\mathbf{X}\mathbf{X}^\top \\
(\mathbf{X}^\top\mathbf{X})(\mathbf{X}^\top\mathbf{X})\vec{\alpha} = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1})\mathbf{X}\mathbf{y} \quad &|\text{substitute } \mathbf{K} = \mathbf{X}^\top\mathbf{X} \\
\mathbf{K}^2\vec{\alpha} = \mathbf{K}\mathbf{y} \quad &|\text{multiply from left by } (\mathbf{K}^{-1})^2 \\
\vec{\alpha} = \mathbf{K}^{-1}\mathbf{y}
\end{aligned}$$

■

Kernel regression can be extended to the kernelized version of *ridge regression*. The solution then becomes

$$\vec{\alpha} = (\mathbf{K} + \tau^2\mathbf{I})^{-1}\mathbf{y}.$$

In practice a small value of $\tau^2 > 0$ increases stability, especially if $\mathbf{K}$ is not invertible. If $\tau = 0$ kernel ridge regression, becomes kernelized ordinary least squares. Typically *kernel ridge regression* is also referred to as *kernel regression*.

Testing

Remember that we defined $\mathbf{w} = \mathbf{X}\vec{\alpha}$. The prediction of a test point $\mathbf{z}$ then becomes

$$h(\mathbf{z}) = \mathbf{z}^\top\mathbf{w} = \mathbf{z}^\top\underbrace{\mathbf{X}\vec{\alpha}}_{\mathbf{w}} = \underbrace{\mathbf{k}_*}_{\mathbf{z}^\top\mathbf{X}}\underbrace{(\mathbf{K} + \tau^2\mathbf{I})^{-1}\mathbf{y}}_{\vec{\alpha}} = \mathbf{k}_*\vec{\alpha},$$

or, if everything is in closed form:

$$h(\mathbf{z}) = \mathbf{k}_*(\mathbf{K} + \tau^2\mathbf{I})^{-1}\mathbf{y},$$

where $\mathbf{k}_*$ is the kernel (vector) of the test point with the training points, i.e. the i^{th} dimension corresponds to $[\mathbf{k}_*]_i = \phi(\mathbf{z})^\top\phi(\mathbf{x}_i)$, the inner-product between the test point $\mathbf{z}$ with the training point $\mathbf{x}_i$ after the mapping into feature space through ϕ.

Nearest Neighbors

Quiz 3: Let $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. How can you kernelize nearest neighbors (with Euclidean distances)?

Kernel SVM

The original, **primal** SVM is a quadratic programming problem:

$$\begin{aligned} \min_{\mathbf{w},b} \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^n \xi_i \\ \text{s.t. } \forall i, \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{aligned}$$

has the **dual** form

$$\begin{aligned} \min_{\alpha_1,\cdots,\alpha_n} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad 0 \leq \alpha_i \leq C \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

where $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)$ (although this is never computed) and

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \right).$$

Support Vectors

There is a very nice interpretation of the dual problem in terms of support vectors. For the primal formulation we know (from a previous lecture) that only support vectors satisfy the constraint with equality:

$$y_i(\mathbf{w}^\top \phi(x_i) + b) = 1.$$

In the dual, these same training inputs can be identified as their corresponding dual values satisfy $\alpha_i > 0$ (all other training inputs have $\alpha_i = 0$). For test-time you only need to compute the sum in $h(\mathbf{x})$ over the support vectors and all inputs $\mathbf{x}_i$ with $\alpha_i = 0$ can be discarded after training.

Recovering b

One apparent problem with the dual version is that b is no longer part of the optimization. However, we need it to perform classification. Luckily, we know that the primal solution and the dual solution are identical. In the duel, support vectors are those with $\alpha_i > 0$. We can then solve the following equation for b

$$\begin{aligned} y_i(\mathbf{w}^\top \phi(x_i) + b) = 1 \\ y_i \left(\sum_j y_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) + b \right) = 1 \\ y_i - \sum_j y_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) = b \end{aligned}$$

This allows us to solve for b from the support vectors (in practice it is best to average the b from several support vectors, as there may be numerical precision problems).

Quiz: What is the dual form of the hard-margin SVM?

Kernel SVM - the smart nearest neighbor

Do you remember the k-nearest neighbor algorithm? For binary classification problems ($y_i \in \{+1, -1\}$), we can write the decision function for a test point $\mathbf{z}$ as

$$h(\mathbf{z}) = \operatorname{sign}\left(\sum_{i=1}^{n} y_i \delta^{nn}(\mathbf{x}_i, \mathbf{z})\right),$$

where $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) \in \{0, 1\}$ with $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) = 1$ only if $\mathbf{x}_i$ is one of the k nearest neighbors of test point $\mathbf{z}$. The SVM decision function

$$h(\mathbf{z}) = \operatorname{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{z}) + b\right)$$

is very similar, but instead of limiting the decision to the k nearest neighbors, it considers *all* training points but the kernel function assigns more weight to those that are closer (large $k(\mathbf{z}, \mathbf{x}_i)$). In some sense you can view the RBF kernel as a soft nearest neighbor assignment, as the exponential decay with distance will assign almost no weight to all but the neighboring points of $\mathbf{z}$. The Kernel SVM algorithm also learns a weight $\alpha_i > 0$ for each training point and a bias b and it essentially "removes" useless training points by setting many $\alpha_i = 0$.