Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.S064 Introduction to Machine Learning

Phase 2: classification (lecture 9)

Kernel methods

We have discussed several linear prediction methods, including the perceptron algorithm, support vector machines, and linear (Ridge) regression. All of these methods can be transformed into non-linear methods simply by mapping examples $x \in \mathcal{R}^d$ into feature vectors $\phi(x) \in \mathcal{R}^p$. Typically $p > d$ and $\phi(x)$ is constructed from x by appending polynomial (or other non-linear) terms involving the coordinates of x such as $x_i x_j$, x_i^2, and so on. The resulting predictors

$$\text{Perceptron}: \quad y = \text{sign}\big(\theta \cdot \phi(x) + \theta_0\big) \tag{1}$$

$$\text{SVM}: \quad y = \text{sign}\big(\theta \cdot \phi(x) + \theta_0\big) \tag{2}$$

$$\text{Linear regression}: \quad y = \theta \cdot \phi(x) + \theta_0 \tag{3}$$

differ from each other based on how they are trained in response to (expanded) training examples $S_n = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \ldots, n\}$. In other words, the estimated parameters $\hat{\theta}$ and $\hat{\theta}_0$ will be different in the three cases even if they were all trained based on the same data. Note that, in the regression case, the responses $y^{(t)}$ are typically not binary labels. However, there's no problem applying the linear regression method even if the training labels are all ± 1. The only issue is the mismatch between what the model/method assumes (real valued responses) and what the data look like (e.g., binary labels).

The problem with explicitly mapping examples to feature vectors $\phi(x)$ is that they tend to be high dimensional, making it difficult to estimate and use the corresponding prediction methods. However, we have already seen how support vector machines can be cast entirely in terms of inner products or *kernels* $K(x, x') = \phi(x) \cdot \phi(x')$. These inner products can be sometimes evaluated substantially more efficiently than the associated feature vectors $\phi(x)$. In fact, we often focus precisely on kernels that can be evaluated efficiently. In such cases, we can *implicitly* work with very high (or even infinite) dimensional feature vectors. For example, the feature vectors that specify the radial basis kernel

$$K(x, x') = \phi(x) \cdot \phi(x') = \exp(-\|x - x'\|^2/2) \tag{4}$$

are indeed infinite dimensional (see below). It is quite easy to specify powerful classifiers or regression methods via kernels. The only requirement is that the prediction method can be cast entirely in terms of inner products.

We begin by developing kernel versions of the perceptron and linear regression methods, and subsequently discuss ways of constructing kernels directly instead of building

feature vectors. All the kernels we consider can be readily used as part of any of the three methods.

Kernel perceptron

We can run the perceptron algorithm (until convergence) when the training examples are linearly separable in the given feature representation. Recall that the algorithm is given by

(0) Initialize: $\theta = 0$ (vector), $\theta_0 = 0$

(1) Cycle through the training examples $t = 1, \ldots, n$

If $y^{(t)}(\theta \cdot \phi(x^{(t)}) + \theta_0) \leq 0$ (mistake)
then $\theta \leftarrow \theta + y^{(t)}\phi(x^{(t)})$ and $\theta_0 \leftarrow \theta_0 + y^{(t)}$

It is clear from this description that the parameters θ and θ_0 at any point in the algorithm can be written as

$$\theta = \sum_{i=1}^{n} \alpha_i y^{(i)} \phi(x^{(i)}) \tag{5}$$

$$\theta_0 = \sum_{i=1}^{n} \alpha_i y^{(i)} \tag{6}$$

where α_i is the number of times that we have made a mistake on the corresponding training example $(\phi(x^{(i)}), y^{(i)})$. Our goal here is to rewrite the algorithm so that we just update α_i's, never explicitly constructing θ which may be high dimensional. To this end, note that

$$\theta \cdot \phi(x) = \sum_{i=1}^{n} \alpha_i y^{(i)} (\phi(x^{(i)}) \cdot \phi(x)) = \sum_{i=1}^{n} \alpha_i y^{(i)} K(x^{(i)}, x) \tag{7}$$

So, the discriminant function $\theta \cdot \phi(x) + \theta_0$ can be written solely in terms of the kernel function and α's

$$\theta \cdot \phi(x) + \theta_0 = \sum_{i=1}^{n} \alpha_i y^{(i)} K(x^{(i)}, x) + \sum_{i=1}^{n} \alpha_i y^{(i)} = \sum_{i=1}^{n} \alpha_i y^{(i)} [K(x^{(i)}, x) + 1] \tag{8}$$

This is all that we need for prediction or for assessing whether there was a mistake on a particular training example $(\phi(x^{(t)}), y^{(t)})$. We can therefore write the algorithm just in terms of α's, updating them in response to each mistake. The resulting *kernel perceptron* algorithm is given by

Initialize: $\alpha_t = 0$, $t = 1, \ldots, n$

Cycle through training examples $t = 1, \ldots, n$

If $y^{(t)}(\sum_{i=1}^{n} \alpha_i y^{(i)}[K(x^{(i)}, x^{(t)}) + 1]) \leq 0$ (mistake)

then $\alpha_t \leftarrow \alpha_t + 1$

Note that the algorithm can be run with any valid kernel function $K(x, x')$. Also, typically only a few of the counts α_i will be non-zero. This means that only a few of the training examples are relevant for finding a separating solution, the rest of the counts α_i remain exactly at zero. So, just as with support vector machines, the solution can be quite *sparse*. Figure 1 below shows the decision boundary

$$\sum_{i=1}^{n} \alpha_i y^{(i)}[K(x^{(i)}, x) + 1] = 0 \tag{9}$$

resulting from running the kernel perceptron algorithm with the radial basis kernel (see later for why this is a valid kernel)

$$K(x, x') = \exp(-\|x - x'\|^2/2) \tag{10}$$

The algorithm makes only four mistakes until a separating solution is found!

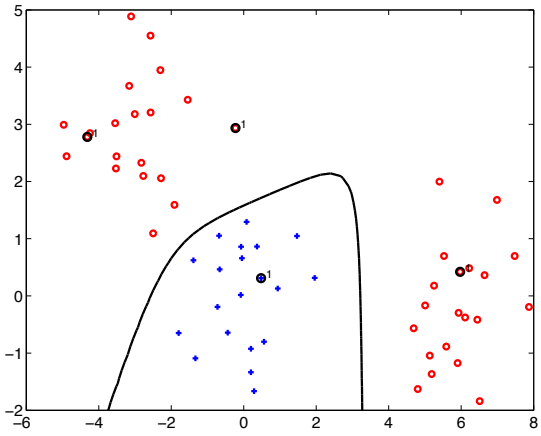

Figure 1: Kernel perceptron example. We have circled the four training examples that the algorithm makes a mistake on.

Kernel linear regression

For simplicity, let's consider here the case where $\theta_0 = 0$, i.e., that the regression function we wish to estimate is given by $\theta \cdot \phi(x)$. With this change, the estimation criterion for

parameters θ was given by

$$J(\theta) = \frac{1}{n}\sum_{t=1}^{n}(y^{(t)} - \theta \cdot \phi(x^{(t)}))^2/2 + \frac{\lambda}{2}\|\theta\|^2 \tag{11}$$

This is a convex function (bowl-shaped) and thus the minimum is obtained at the point where the gradient is zero. To this end,

$$\begin{aligned} \frac{\partial}{\partial\theta}J(\theta) &= -\frac{1}{n}\sum_{t=1}^{n}\underbrace{(y^{(t)} - \theta \cdot \phi(x^{(t)}))}_{=n\lambda\,\alpha_t}\phi(x^{(t)}) + \lambda\theta & (12)\\ &= -\lambda\sum_{t=1}^{n}\alpha_t\phi(x^{(t)}) + \lambda\theta = 0 & (13)\end{aligned}$$

or, equivalently, that $\theta = \sum_{t=1}^{n}\alpha_t\phi(x^{(t)})$, where α_t are proportional to prediction errors (recall that α's were related to errors also in SVM and perceptron algorithms). The above equation holds only if α_t and θ relate to each other in a specific way, i.e., only if

$$\begin{aligned} n\lambda\,\alpha_t &= y^{(t)} - \theta \cdot \phi(x^{(t)}) & (14)\\ &= y^{(t)} - \left(\sum_{i=1}^{n}\alpha_i\phi(x^{(i)})\right) \cdot \phi(x^{(t)}) & (15)\\ &= y^{(t)} - \sum_{i=1}^{n}\alpha_i K(x^{(i)}, x^{(t)}) & (16)\end{aligned}$$

which should hold for all $t = 1, \ldots, n$. Let's write the equation in a vector form. $\vec{\alpha} = [\alpha_1, \ldots, \alpha_n]^T$, $\vec{y} = [y^{(1)}, \ldots, y^{(n)}]^T$, and because $K(x^{(i)}, x^{(t)}) = K(x^{(t)}, x^{(i)})$,

$$\sum_{i=1}^{n}\alpha_i K(x^{(i)}, x^{(t)}) = \sum_{i=1}^{n}K(x^{(t)}, x^{(i)})\alpha_i = [K\vec{\alpha}]_t \tag{17}$$

where K is a $n \times n$ matrix whose i, j element is $K(x^{(i)}, x^{(j)})$ (a.k.a. the Gram matrix). Now, in a vector form, we have

$$n\lambda\,\vec{\alpha} = \vec{y} - K\vec{\alpha} \quad \text{or} \quad (n\lambda\,I + K)\vec{\alpha} = \vec{y} \tag{18}$$

The solution is simply $\vec{\alpha} = (n\lambda\,I + K)^{-1}\vec{y}$ (always invertible for $\lambda > 0$). In other words, estimated coefficients $\hat{\alpha}_t$ can be computed only in terms of the kernel function and the target responses, never needing to explicitly construct feature vectors $\phi(x)$. Once we have the coefficients, prediction for a new point x is similarly easy

$$\hat{\theta} \cdot \phi(x) = \sum_{i=1}^{n}\hat{\alpha}_i\phi(x^{(i)}) \cdot \phi(x) = \sum_{i=1}^{n}\hat{\alpha}_i K(x^{(i)}, x) \tag{19}$$

Figure 2 below shows examples of one dimensional regression problems with higher order polynomial kernels. Which of these kernels should we use? (the right answer is linear; this is how the data was generated, with noise). The general problem selecting the kernel (or the corresponding feature representation) is a *model selection* problem. We can always try to use cross-validation as a model selection criterion.

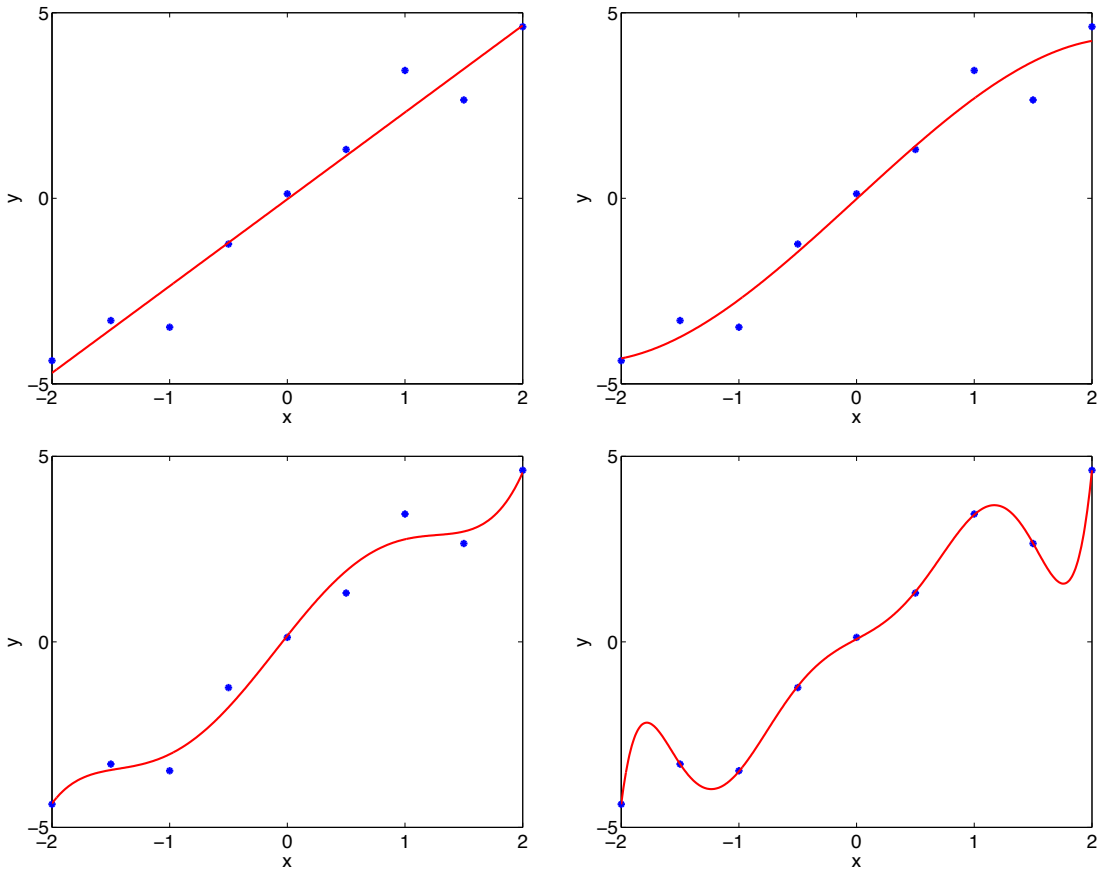

Figure 2: Kernel regression with a linear kernel (top left), 3rd order polynomial kernel (top right), 5th order polynomial kernel (bottow left), and a 7th order polynomial kernel (bottom right).

Kernel functions

All of the methods discussed above can be run with any valid kernel function $K(x, x')$. A kernel function is valid if and only if there exists some feature mapping $\phi(x)$ such that $K(x, x') = \phi(x) \cdot \phi(x')$. We don't need to know what $\phi(x)$ is (necessarily), only that one exists. We can build many common kernel functions based only on the following four rules

(1) $K(x, x') = 1$ is a kernel function.

(2) Let $f : \mathcal{R}^d \to \mathcal{R}$ be any real valued function of x. Then, if $K(x, x')$ is a kernel function, then so is $\tilde{K}(x, x') = f(x)K(x, x')f(x')$

(3) If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then so is their sum. In other words, $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel.

(4) If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then so is their product $K(x, x') = K_1(x, x')K_2(x, x')$

To understand these composition rules, let's figure out how they relate to the underlying feature mappings. For example, a constant kernel $K(x, x') = 1$ simply corresponds to $\phi(x) = 1$ for all $x \in \mathcal{R}^d$. Similarly, if $\phi(x)$ is the feature mapping for kernel $K(x, x')$, then $\tilde{K}(x, x') = f(x)K(x, x')f(x')$ (rule 2) corresponds to $\tilde{\phi}(x) = f(x)\phi(x)$. Adding kernels means appending feature vectors. For example, let's say that $K_1(x, x')$ and $K_2(x, x')$ correspond to feature mappings $\phi^{(1)}(x)$ and $\phi^{(2)}(x)$, respectively. Then (see rule 3)

$$\begin{aligned} K(x, x') &= \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix} \cdot \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix} && (20) \\ &= \phi^{(1)}(x) \cdot \phi^{(1)}(x') + \phi^{(2)}(x) \cdot \phi^{(2)}(x') && (21) \\ &= K_1(x, x') + K_2(x, x') && (22) \end{aligned}$$

Can you figure out what the feature mapping is for $K(x, x')$ in rule 4, expressed in terms of the feature mappings for $K_1(x, x')$ and $K_2(x, x')$?

Many typical kernels can be constructed on the basis of these rules. For example, $K(x, x') = x \cdot x'$ is a kernel based on rules (1), (2), and (3). To see this, let $f_i(x) = x_i$ (i^{th} coordinate mapping), then

$$x \cdot x' = x_1 x'_1 + \ldots + x_d x'_d = f_1(x)\, 1\, f_1(x') + \cdots + f_d(x)\, 1\, f_d(x') \tag{23}$$

where each term uses rules (1) and (2), and the addition follows from rule (3). Similarly, the 2nd order polynomial kernel

$$K(x, x') = (x \cdot x') + (x \cdot x')^2 \tag{24}$$

can be built from assuming that $(x \cdot x')$ is a kernel, using the product rule to realize the 2nd term, i.e., $(x \cdot x')^2 = (x \cdot x')(x \cdot x')$, and finally adding the two. More interestingly,

$$K(x, x') = \exp(x \cdot x') = 1 + (x \cdot x') + \frac{1}{2!}(x \cdot x')^2 + \cdots \tag{25}$$

is also a kernel by the same rules. But, since the expansion is an infinite sum, the resulting feature representation for $K(x, x')$ is infinite dimensional! This is also why the radial basis kernel has an infinite dimensional feature representation. Specifically,

$$\begin{aligned} K(x, x') &= \exp(-\|x - x'\|^2/2) && (26) \\ &= \exp(-\|x\|^2/2) \exp(x \cdot x') \exp(-\|x'\|^2/2) && (27) \\ &= f(x) \exp(x \cdot x') f(x') && (28) \end{aligned}$$

where $f(x) = \exp(-\|x\|^2/2)$. The radial basis kernel is special in many ways. For example, any distinct set of training examples are always perfectly separable with the radial basis kernel (follows from a Michelli theorem about monotone functions of distance and the invertibility of the corresponding Gram matrices; theorem not shown). In other words, running the perceptron algorithm with the radial basis kernel will always converge to a separable solution provided that the training examples are all distinct.