SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.007 Machine Learning, Summer 2022
Lecture Notes for Week 4

SVM: Kernel Methods

Last update: Wednesday 1st June, 2022 06:41

1 Non-linear Support Vector Machines – Kernels

Let's start by considering how we can use linear classifiers to make non-linear predictions. The easiest way is to first map all the examples $x \in \mathcal{R}^d$ into a different feature representation $\phi(x) \in \mathcal{R}^p$ where typically p is much larger than d. We would then simply use a linear classifier on the new (higher dimensional) feature vectors, pretending that they were the original input vectors. There are many ways to create such feature vectors. For example, we can build $\phi(x)$ by concatenating polynomial terms of the original coordinates. For example, in two dimensions, we could map $x = [x_1, x_2]^T$ to a five dimensional feature vector

$$\phi(x) = [x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T \tag{1}$$

We can then train a "linear" classifier (linear in the new ϕ-coordinates)

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

by mapping each training example to the corresponding feature vector. In other words, our training set is now $S_n^\phi = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \ldots, n\}$. The resulting parameter estimates $\hat{\theta}$, $\hat{\theta}_0$ define a linear decision boundary in the ϕ-coordinates but a non-linear boundary in the original x-coordinates

$$\hat{\theta} \cdot \phi(x) + \hat{\theta}_0 \Leftrightarrow \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 \sqrt{2} x_1 x_2 + \hat{\theta}_4 x_1^2 + \hat{\theta}_5 x_2^2 + \hat{\theta}_0 = 0 \tag{3}$$

The non-linear boundary can represent, e.g., an ellipse in the original two dimensional space.

The main problem with the above procedure is that the feature vectors $\phi(x)$ can become quite high dimensional. For example, if we start with $x \in \mathcal{R}^d$, where $d = 1,000$, then compiling $\phi(x)$ by concatenating polynomial terms up to the 2^{nd} order would have dimension $d + d(d + 1)/2$ or about 500,000. Using higher order polynomial terms in such situations becomes quickly infeasible. However, it may still be possible to *implicitly* use such feature vectors. If training and prediction problems can be formulated only in terms of inner products between examples, then the relevant computation for us is $\phi(x) \cdot \phi(x')$. Depending on how we define $\phi(x)$, this computation can be carried out efficiently even if using $\phi(x)$ explicitly is not. For example, when $\phi(x) = [x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T$, we see that (check!)

$$\phi(x) \cdot \phi(x') = (x \cdot x') + (x \cdot x')^2 \tag{4}$$

So the inner product is obtained easily from the original input vectors. One of the main advantages from considering the dual form of support vector machines is that it can be expressed entirely in terms of inner products. For example, in solving for the Lagrange multipliers $\hat{\alpha}_t$, the dual problem only requires us to evaluate inner products between the training examples, i.e., $(x^{(t)} \cdot x^{(t')})$. Similarly, the parameter $\hat{\theta}_0$ can be reconstructed from the margin constraints based on inner products. Finally, we can predict labels for new examples x with access only to the inner products between the training examples $x^{(t)}$ and the new points:

$$y = \text{sign}(\hat{\theta} \cdot x + \hat{\theta}_0) = \text{sign}\left(\sum_{t=1}^{n} \hat{\alpha}_t y^{(t)} (x^{(t)} \cdot x) + \hat{\theta}_0 \right) \tag{5}$$

When we map examples into feature vectors, we replace $(x^{(t)} \cdot x^{(t')})$ with $\phi(x^{(t)}) \cdot \phi(x^{(t')})$. Instead of explicitly using feature vectors, we will focus on evaluating *kernel* functions $K(x^{(t)}, x^{(t')}) = \phi(x^{(t)}) \cdot \phi(x^{(t')})$.

2 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

$$\text{Perceptron :} \qquad y = \text{sign}(\theta \cdot \phi(x) + \theta_0) \tag{6}$$

$$\text{SVM :} \qquad y = \text{sign}(\theta \cdot \phi(x) + \theta_0) \tag{7}$$

$$\text{Linear Regression :} \qquad y = \theta \cdot \phi(x) + \theta_0 \tag{8}$$

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, as mentioned above, 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 function kernel* (or *RBF kernel*):

$$K(x, x') = \phi(x) \cdot \phi(x') = \exp(-\|x - x'\|/2) \tag{9}$$

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.

Properties of Kernel Functions

Both the kernel SVM and kernel perceptron 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)

$$K(x, x') = \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix} \cdot \begin{bmatrix} \phi^{(1)}(x') \\ \phi^{(2)}(x') \end{bmatrix} \tag{10}$$

$$= \phi^{(1)}(x) \cdot \phi^{(1)}(x') + \phi^{(2)}(x) \cdot \phi^{(2)}(x') \tag{11}$$

$$= K_1(x, x') + K_2(x, x') \tag{12}$$

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

$$x \cdot x' = x_1 x'_1 + \ldots + x_d x'_d = f_1(x)1f_1(x') + \ldots + f_d(x)1f_d(x') \tag{13}$$

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

$$K(x, x') = (x \cdot x') + (x \cdot x')^2 \tag{14}$$

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,

$$K(x, x') = \exp(x \cdot x') = 1 + (x \cdot x') + \frac{1}{2!}(x \cdot x')^2 + \dots \tag{15}$$

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 function kernel has an infinite dimensional feature representation. Specifically,

$$K(x, x') = \exp(-\left\|x - x'\right\|^2/2) \tag{16}$$

$$= \exp(-\left\|x\right\|^2/2)\exp(x \cdot x')\exp(-\left\|x'\right\|^2/2) \tag{17}$$

$$= f(x)\exp(x \cdot x')f(x') \tag{18}$$

where $f(x) = \exp(-\left\|x\right\|^2/2)$. The radial basis function kernel is special in many ways. For example, running the SVM with such a kernel function will always be able to return you a separable solution provided the training examples are all distinct.

Learning Objective

You need to know:

1. What is a kernel function and how it is used in the context of SVM.
2. What are the essential properties associated with a kernel function.
3. How to identify whether a function is a valid (or an invalid) kernel function.
4. What is the definition of a RBF kernel.