

Non-linear predictions, kernels

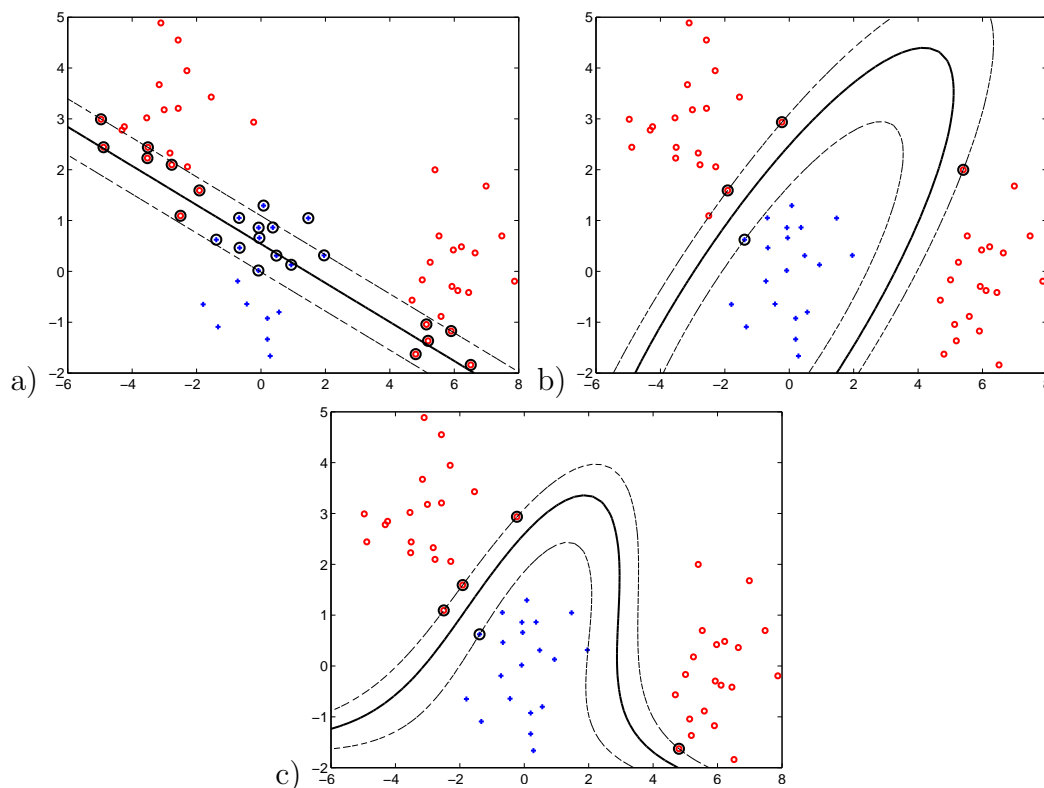


Figure 1: Example solutions with different feature representations. a) only linear features; b) up to second order polynomial features; c) up to third order polynomial features

Linear classifiers such as perceptron and SVM can be extended to non-linear classifiers. The classifiers remain linear in the parameters but perform non-linear operations in the original input space. This is achieved by mapping the input examples to a (higher dimensional) feature space where the dimensions in the new feature vectors include non-linear functions of the inputs. In other words, we consider classifiers of the form

$$f(x; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta} \cdot \underline{\phi}(x) + \theta_0) \quad (1)$$

The simplest setting for demonstrating this is when the input examples are just real numbers $x \in \mathcal{R}$. We can obtain a quadratic classifier by simply mapping the input x to a longer feature vector that includes a term quadratic in x . A third order model can be constructed

by including all terms up to degree three, and so on. Explicitly, we would make linear predictions using feature vectors

$$x \xrightarrow{\phi} [1, \sqrt{2}x, x^2]^T = \underline{\phi}(x) \quad (2)$$

$$x \xrightarrow{\phi} [1, \sqrt{3}x, \sqrt{3}x^2, x^3]^T \quad (3)$$

$$\dots \quad (4)$$

The role of $\sqrt{2}$ and other constants will become clear shortly. The dimensionality of $\underline{\phi}(x)$ (and therefore also $\underline{\theta}$) depends on the order of the polynomial expansion.

The polynomial expansion of input vectors works the same in higher dimensions, e.g.,

$$\underline{x} = [x_1, x_2]^T \xrightarrow{\phi} [1, x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T = \underline{\phi}(\underline{x}) \quad (5)$$

One limitation of explicating the feature vectors is that the dimensionality can increase rapidly with the degree of polynomial expansion, especially when the dimension of the input vector is already high. The table below gives some indication of this though the effect is more dramatic with higher input dimensions.

$dim(\underline{x}) = 2$		$dim(\underline{x}) = 3$	
degree p	# of features	degree p	# of features
2	6	2	10
3	10	3	20
4	15	4	35
5	21	5	56

Can we somehow avoid explicating the dimensions of the feature vectors? In the context of classifiers we have discussed thus far, yes, we can. We can define the feature vectors implicitly by focusing on specifying the values of their inner products or *kernel* instead. To get a sense of the power of this approach, let's evaluate the inner product between two feature vectors corresponding to the specific cubic expansions of 1-dimensional inputs shown before:

$$\underline{\phi}(x) = [1, \sqrt{3}x, \sqrt{3}x^2, x^3]^T, \quad (6)$$

$$\underline{\phi}(x') = [1, \sqrt{3}x', \sqrt{3}x'^2, x'^3]^T, \quad (7)$$

$$\underline{\phi}(x)^T \underline{\phi}(x') = 1 + 3xx' + 3(xx')^2 + (xx')^3 = (1 + xx')^3 \quad (8)$$

So it seems we can compactly evaluate the inner products between polynomial feature vectors. The effect is more striking with higher dimensional inputs and higher polynomial

degrees. (We did have to specify the constants appropriately in the feature vectors to make this work). To shift the modeling from explicit feature vectors to inner products (kernels) we obviously have to first turn the estimation problem into a form that involves only inner products between feature vectors.

Kernels

In lectures, we have discussed how to turn a perceptron algorithm into a kernel perceptron that makes use of the feature vectors only through their inner product with others, during training and prediction. This means that we can simply substitute different kernel functions $K(\underline{x}, \underline{x}')$ into the estimation/prediction equations. This gives us an easy access to a wide range of possible regression functions. Here are a couple of standard examples of kernels:

- *Polynomial kernel*

$$K(\underline{x}, \underline{x}') = (1 + \underline{x}^T \underline{x}')^p, \quad p = 1, 2, \dots \quad (9)$$

- *Radial basis kernel*

$$K(\underline{x}, \underline{x}') = \exp\left(-\frac{\beta}{2}\|\underline{x} - \underline{x}'\|^2\right), \quad \beta > 0 \quad (10)$$

We have already discussed the feature vectors corresponding to the polynomial kernel. The components of these feature vectors were polynomial terms up to degree p with specifically chosen coefficients. The restricted choice of coefficients was necessary in order to collapse the inner product calculations.

The feature “vectors” corresponding to the radial basis kernel are infinite dimensional! The components of these “vectors” are indexed by $\underline{z} \in \mathcal{R}^d$ where d is the dimension of the original input \underline{x} . More precisely, the feature vectors are functions:

$$\underline{\phi}(\underline{z}; \underline{x}) = c(\beta, d) N(\underline{z}; \underline{x}, 1/2\beta) \quad (11)$$

where $N(\underline{z}; \underline{x}, (1/\beta))$ is a normal pdf over \underline{z} and $c(\beta, d)$ is a constant. Roughly speaking, the radial basis kernel measures the probability that you would get the same sample \underline{z} (in the same small region) from two normal distributions with means \underline{x} and \underline{x}' and a common variance $1/2\beta$. This is a reasonable measure of “similarity” between \underline{x} and \underline{x}' and kernels

are often defined from this perspective. The inner product giving rise to the radial basis kernel is defined through integration

$$K(\underline{x}, \underline{x}') = \int \underline{\phi}(z; \underline{x}) \underline{\phi}(z; \underline{x}') dz \quad (12)$$

We can also construct various types of kernels from simpler ones. Here are a few rules to guide us. Assume $K_1(\underline{x}, \underline{x}')$ and $K_2(\underline{x}, \underline{x}')$ are valid kernels (correspond to inner products of some feature vectors), then

1. $K(\underline{x}, \underline{x}') = f(\underline{x})K_1(\underline{x}, \underline{x}')f(\underline{x}')$ for any real valued function $f(\underline{x})$,
2. $K(\underline{x}, \underline{x}') = K_1(\underline{x}, \underline{x}') + K_2(\underline{x}, \underline{x}')$,
3. $K(\underline{x}, \underline{x}') = K_1(\underline{x}, \underline{x}')K_2(\underline{x}, \underline{x}')$

are all valid kernels. While simple, these rules are quite powerful. Let's first understand these rules from the point of view of the implicit feature vectors. For each rule, let $\underline{\phi}(\underline{x})$ be the feature vector corresponding to K and $\underline{\phi}^{(1)}(\underline{x})$ and $\underline{\phi}^{(2)}(\underline{x})$ the feature vectors associated with K_1 and K_2 , respectively. The feature mapping for the first rule is given simply by multiplying with the scalar function $f(\underline{x})$:

$$\underline{\phi}(\underline{x}) = f(\underline{x})\underline{\phi}^{(1)}(\underline{x}) \quad (13)$$

so that $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}') = f(\underline{x})\underline{\phi}^{(1)}(\underline{x})^T \underline{\phi}^{(1)}(\underline{x}')f(\underline{x}') = f(\underline{x})K_1(\underline{x}, \underline{x}')f(\underline{x}')$. The second rule, adding kernels, corresponds to just concatenating the feature vectors

$$\underline{\phi}(\underline{x}) = \begin{bmatrix} \underline{\phi}^{(1)}(\underline{x}) \\ \underline{\phi}^{(2)}(\underline{x}) \end{bmatrix} \quad (14)$$

The third and the last rule is a little more complicated but not much. Suppose we use a double index i, j to index the components of $\underline{\phi}(\underline{x})$ where i ranges over the components of $\underline{\phi}^{(1)}(\underline{x})$ and j refers to the components of $\underline{\phi}^{(2)}(\underline{x})$. Then

$$\phi_{i,j}(\underline{x}) = \phi_i^{(1)}(\underline{x})\phi_j^{(2)}(\underline{x}) \quad (15)$$

It is now easy to see that

$$K(\underline{x}, \underline{x}') = \underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}') \quad (16)$$

$$= \sum_{i,j} \phi_{i,j}(\underline{x}) \phi_{i,j}(\underline{x}') \quad (17)$$

$$= \sum_{i,j} \phi_i^{(1)}(\underline{x}) \phi_j^{(2)}(\underline{x}) \phi_i^{(1)}(\underline{x}') \phi_j^{(2)}(\underline{x}') \quad (18)$$

$$= \left[\sum_i \phi_i^{(1)}(\underline{x}) \phi_i^{(1)}(\underline{x}') \right] \left[\sum_j \phi_j^{(2)}(\underline{x}) \phi_j^{(2)}(\underline{x}') \right] \quad (19)$$

$$= [\underline{\phi}^{(1)}(\underline{x})^T \underline{\phi}^{(1)}(\underline{x}')] [\underline{\phi}^{(2)}(\underline{x})^T \underline{\phi}^{(2)}(\underline{x}')] \quad (20)$$

$$= K_1(\underline{x}, \underline{x}') K_2(\underline{x}, \underline{x}') \quad (21)$$

These construction rules can also be used to verify that something is a valid kernel. As an example, let's figure out why a radial basis kernel

$$K(\underline{x}, \underline{x}') = \exp\left\{-\frac{1}{2}\|\underline{x} - \underline{x}'\|^2\right\} \quad (22)$$

is a valid kernel.

$$\exp\left\{-\frac{1}{2}\|\underline{x} - \underline{x}'\|^2\right\} = \exp\left\{-\frac{1}{2}\underline{x}^T \underline{x} + \underline{x}^T \underline{x}' - \frac{1}{2}\underline{x}'^T \underline{x}'\right\} \quad (23)$$

$$= \underbrace{\exp\left\{-\frac{1}{2}\underline{x}^T \underline{x}\right\}}_{f(\underline{x})} \cdot \exp\{\underline{x}^T \underline{x}'\} \cdot \underbrace{\exp\left\{-\frac{1}{2}\underline{x}'^T \underline{x}'\right\}}_{f(\underline{x}')} \quad (24)$$

Here $\exp\{\underline{x}^T \underline{x}'\}$ is a sum of simple products $\underline{x}^T \underline{x}'$ and is therefore a kernel based on the second and third rules; the first rule allows us to incorporate $f(\underline{x})$ and $f(\underline{x}')$.

String kernels. It is often necessary to make predictions (classify, assess risk, determine user ratings) on the basis of more complex objects such as variable length sequences or graphs that do not necessarily permit a simple description as points in \mathcal{R}^d . The idea of kernels extends to such objects as well. Consider, for example, the case where the inputs \underline{x} are variable length sequences (e.g., documents or biosequences) with elements from some common alphabet \mathcal{A} (e.g., letters or protein residues). One way to compare such sequences is to consider subsequences that they may share. Let $\mathbf{u} \in \mathcal{A}^k$ denote a length k sequence from this alphabet and \mathbf{i} a sequence of k indexes. So, for example, we can say that $\mathbf{u} = \underline{x}[\mathbf{i}]$ if $u_1 = x_{i_1}$, $u_2 = x_{i_2}$, \dots , $u_k = x_{i_k}$. In other words, \underline{x} contains the elements of \mathbf{u} in positions $i_1 < i_2 < \dots < i_k$. If the elements of \mathbf{u} are found in successive positions in \mathbf{x} ,

then $i_k - i_1 = k - 1$. A simple string kernel corresponds to feature vectors with counts of occurrences of length k subsequences:

$$\underline{\phi}_{\mathbf{u}}(\underline{x}) = \sum_{\mathbf{i}: \mathbf{u}=\mathbf{x}[\mathbf{i}]} \delta(i_k - i_1, k - 1) \quad (25)$$

In other words, the components are indexed by subsequences \mathbf{u} and the value of \mathbf{u} -component is the number of times \mathbf{x} contains \mathbf{u} as a contiguous subsequence. For example,

$$\underline{\phi}_{\text{on}}(\text{the common construct}) = 2 \quad (26)$$

The number of components in such feature vectors is very large (exponential in k). Yet, the inner product

$$\sum_{\mathbf{u} \in \mathcal{A}^k} \underline{\phi}_{\mathbf{u}}(\underline{x}) \underline{\phi}_{\mathbf{u}}(\underline{x}') \quad (27)$$

can be computed efficiently (there are only a limited number of possible contiguous subsequences in \underline{x} and \underline{x}'). The reason for this difference, and the argument in favor of kernels more generally, is that the feature vectors have to aggregate the information necessary to compare any two sequences while the inner product is evaluated for two *specific* sequences.

We can also relax the requirement that matches must be contiguous. To this end, we define the length of the window of \underline{x} where \mathbf{u} appears as $l(\mathbf{i}) = i_k - i_1$. The feature vectors in a *weighted gapped substring kernel* are given by

$$\underline{\phi}_{\mathbf{u}}(\underline{x}) = \sum_{\mathbf{i}: \mathbf{u}=\mathbf{x}[\mathbf{i}]} \lambda^{l(\mathbf{i})} \quad (28)$$

where the parameter $\lambda \in (0, 1)$ specifies the penalty for non-contiguous matches to \mathbf{u} . The resulting kernel

$$K(\underline{x}, \underline{x}') = \sum_{\mathbf{u} \in \mathcal{A}^k} \underline{\phi}_{\mathbf{u}}(\underline{x}) \underline{\phi}_{\mathbf{u}}(\underline{x}') = \sum_{\mathbf{u} \in \mathcal{A}^k} \left(\sum_{\mathbf{i}: \mathbf{u}=\mathbf{x}[\mathbf{i}]} \lambda^{l(\mathbf{i})} \right) \left(\sum_{\mathbf{i}: \mathbf{u}=\mathbf{x}'[\mathbf{i}]} \lambda^{l(\mathbf{i})} \right) \quad (29)$$

can be computed recursively. It is often useful to normalize such a kernel so as to remove any immediate effect from the sequence length:

$$\tilde{K}(\underline{x}, \underline{x}') = \frac{K(\underline{x}, \underline{x}')}{\sqrt{K(\underline{x}, \underline{x})} \sqrt{K(\underline{x}', \underline{x}')}} \quad (30)$$