17 Kernel Methods *

In this chapter, we consider **nonparametric methods** for regression and classification. Such methods do not assume a fixed parametric form for the prediction function, but instead try to estimate the function itself (rather than the parameters) directly from data. The key idea is that we observe the function value at a fixed set of N points, namely $y_n = f(\boldsymbol{x}_n)$ for $n = 1 : N$, where f is the unknown function, so to predict the function value at a new point, say $\boldsymbol{x}_*$, we just have to compare how "similar" $\boldsymbol{x}_*$ is to each of the N training points, $\{\boldsymbol{x}_n\}$, and then we can predict that $f(\boldsymbol{x}_*)$ is some weighted combination of the $\{f(\boldsymbol{x}_n)\}$ values. Thus we may need to "remember" the entire training set, $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$, in order to make predictions at test time — we cannot "compress" $\mathcal{D}$ into a fixed-sized parameter vector.

The weights that are used for prediction are determined by the similarity between $\boldsymbol{x}_*$ and each $\boldsymbol{x}_n$, which is computed using a special kind of function known as kernel function, $\mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_*) \geq 0$, which we explain in Section 17.1. This approach is similar to RBF networks (Section 13.6.1), except we use the datapoints $\{\boldsymbol{x}_n\}$ themselves as the "anchors", rather than learning the RBF centroids $\{\boldsymbol{\mu}_k\}$.

In Section 17.2, we discuss an approach called Gaussian processes, which allows us to use the kernel to define a *prior over functions*, which we can update given data to get a *posterior over functions*. Alternatively we can use the same kernel with a method called Support Vector Machines to compute a MAP estimate of the function, as we explain in Section 17.3.

17.1 Mercer kernels

The key to nonparametric methods is that we need a way to encode prior knowledge about the similarity of two input vectors. If we know that $\boldsymbol{x}_i$ is similar to $\boldsymbol{x}_j$, then we can encourage the model to make the predicted output at both locations (i.e., $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$) to be similar.

To define similarity, we introduce the notion of a **kernel function**. The word "kernel" has many different meanings in mathematics, including density kernels (Section 16.3.1), transition kernels of a Markov chain (Section 3.6.1.2), and convolutional kernels (Section 14.1). Here we consider a **Mercer kernel**, also called a **positive definite kernel**. This is any symmetric function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ such that

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) c_i c_j \geq 0 \tag{17.1}$$

for any set of N (unique) points $\boldsymbol{x}_i \in \mathcal{X}$, and any choice of numbers $c_i \in \mathbb{R}$. (We assume $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) > 0$, so that we can only achieve equality in the above equation if $c_i = 0$ for all i.)

Another way to understand this condition is the following. Given a set of N datapoints, let us define the **Gram matrix** as the following $N \times N$ similarity matrix:

$$\mathbf{K} = \begin{pmatrix} \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ & \vdots & \\ \mathcal{K}(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & \mathcal{K}(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix} \tag{17.2}$$

We say that $\mathcal{K}$ is a Mercer kernel iff the Gram matrix is positive definite for any set of (distinct) inputs $\{\boldsymbol{x}_i\}_{i=1}^N$.

The most widely used kernel for real-valued inputs is the **squared exponential kernel** (SE), also called the **exponentiated quadratic kernel** (EQ), **Gaussian kernel**, or **RBF kernel**. It is defined by

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\ell^2}\right) \tag{17.3}$$

Here ℓ corresponds to the length scale of the kernel, i.e., the distance over which we expect differences to matter. This is known as the **bandwidth** parameter. The RBF kernel measures similarity between two vectors in $\mathbb{R}^D$ using (scaled) Euclidean distance. In Section 17.1.2, we will discuss several other kinds of kernel.

In Section 17.2, we show how to use kernels to define priors and posteriors over functions. The basic idea is this: if $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$ is large, meaning the inputs are similar, then we expect the output of the function to be similar as well, so $f(\boldsymbol{x}) \approx f(\boldsymbol{x}')$. More precisely, information we learn about $f(\boldsymbol{x})$ will help us predict $f(\boldsymbol{x}')$ for all $\boldsymbol{x}'$ which are correlated with $\boldsymbol{x}$, and hence for which $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$ is large.

In Section 17.3, we show how to use kernels to generalize from Euclidean distance to a more general notion of distance, so that we can use geometric methods such as linear discriminant analysis in an implicit feature space instead of input space.

17.1.1 Mercer's theorem

Recall from Section 7.4 that any positive definite matrix $\mathbf{K}$ can be represented using an eigendecomposition of the form $\mathbf{K} = \mathbf{U}^\mathsf{T}\boldsymbol{\Lambda}\mathbf{U}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$, and $\mathbf{U}$ is a matrix containing the eigenvectors. Now consider element (i, j) of $\mathbf{K}$:

$$k_{ij} = (\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:i})^\mathsf{T}(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:j}) \tag{17.4}$$

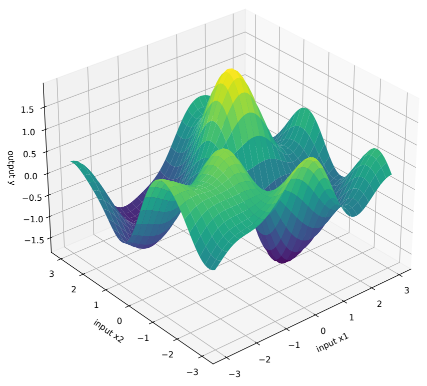
where $\mathbf{U}_{:i}$ is the i'th column of $\mathbf{U}$. If we define $\boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:i}$, then we can write

$$k_{ij} = \boldsymbol{\phi}(\boldsymbol{x}_i)^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x}_j) = \sum_m \phi_m(\boldsymbol{x}_i)\phi_m(\boldsymbol{x}_j) \tag{17.5}$$

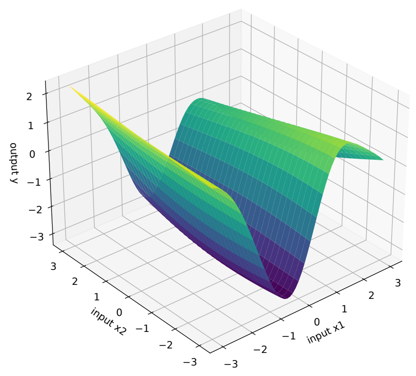
Thus we see that the entries in the kernel matrix can be computed by performing an inner product of some feature vectors that are implicitly defined by the eigenvectors of the kernel matrix. This idea can be generalized to apply to kernel functions, not just kernel matrices; this result is known as **Mercer's theorem**.

For example, consider the **quadratic kernel** $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2$. In 2d, we have

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = (x_1 x_1' + x_2 x_2')^2 = x_1^2 (x_1')^2 + 2x_1 x_2 x_1' x_2' + x_2^2 (x_2')^2 \tag{17.6}$$



(a)



(b)

Figure 17.1: Function samples from a GP with an ARD kernel. (a) $\ell_1 = \ell_2 = 1$. Both dimensions contribute to the response. (b) $\ell_1 = 1$, $\ell_2 = 5$. The second dimension is essentially ignored. Adapted from Figure 5.1 of [RW06]. Generated by gprDemoArd.ipynb.

We can write this as $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{x})$ if we define $\boldsymbol{\phi}(x_1, x_2) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3$. So we embed the 2d inputs $\boldsymbol{x}$ into a 3d feature space $\boldsymbol{\phi}(\boldsymbol{x})$.

Now consider the RBF kernel. In this case, the corresponding feature representation is infinite dimensional (see Section 17.2.9.3 for details). However, by working with kernel functions, we can avoid having to deal with infinite dimensional vectors.

17.1.2 Some popular Mercer kernels

In the sections below, we describe some popular Mercer kernels. More details can be found at [Wil14] and `https://www.cs.toronto.edu/~duvenaud/cookbook/`.

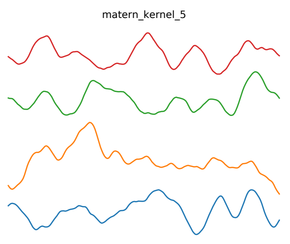
17.1.2.1 Stationary kernels for real-valued vectors

For real-valued inputs, $\mathcal{X} = \mathbb{R}^D$, it is common to use **stationary kernels**, which are functions of the form $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \mathcal{K}(||\boldsymbol{x} - \boldsymbol{x}'||)$; thus the value only depends on the elementwise difference between the inputs. The RBF kernel is a stationary kernel. We give some other examples below.

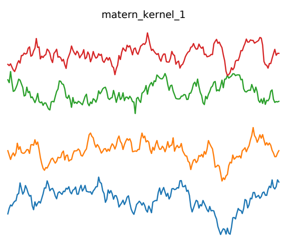
ARD kernel

We can generalize the RBF kernel by replacing Euclidean distance with Mahalanobis distance, as follows:

$$\mathcal{K}(\boldsymbol{r}) = \sigma^2 \exp\left(-\frac{1}{2}\boldsymbol{r}^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{r}\right) \tag{17.7}$$



(a)



(b)

Figure 17.2: Functions sampled from a GP with a Matern kernel. (a) $\nu = 5/2$. (b) $\nu = 1/2$. Generated by gpKernelPlot.ipynb.

where $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{x}'$. If $\boldsymbol{\Sigma}$ is diagonal, this can be written as

$$\mathcal{K}(\boldsymbol{r}; \boldsymbol{\ell}, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{1}{\ell_d^2}r_d^2\right) = \prod_{d=1}^{D}\mathcal{K}(r_d; \ell_d, \sigma^{2/d}) \tag{17.8}$$

where

$$\mathcal{K}(r; \ell, \tau^2) = \tau^2 \exp\left(-\frac{1}{2}\frac{1}{\ell^2}r^2\right) \tag{17.9}$$

We can interpret σ^2 as the overall variance, and ℓ_d as defining the **characteristic length scale** of dimension d. If d is an irrelevant input dimension, we can set $\ell_d = \infty$, so the corresponding dimension will be ignored. This is known as **automatic relevancy determination** or **ARD** (Section 11.7.7). Hence the corresponding kernel is called the **ARD kernel**. See Figure 17.1 for an illustration of some 2d functions sampled from a GP using this prior.

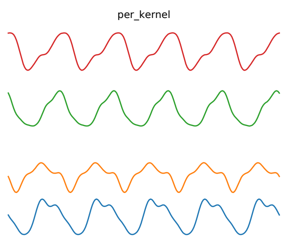
Matern kernels

The SE kernel gives rise to functions that are infinitely differentiable, and therefore are very smooth. For many applications, it is better to use the **Matern kernel**, which gives rise to "rougher" functions, which can better model local "wiggles" without having to make the overall length scale very small.

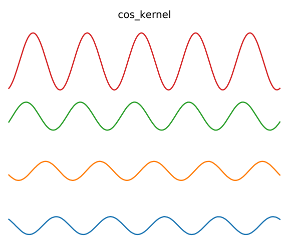
The Matern kernel has the following form:

$$\mathcal{K}(r; \nu, \ell) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}r}{\ell}\right) \tag{17.10}$$

where K_ν is a modified Bessel function and ℓ is the length scale. Functions sampled from this GP are k-times differentiable iff $\nu > k$. As $\nu \to \infty$, this approaches the SE kernel.



(a) Periodic kernel.



(b) Cosine kernel.

Figure 17.3: Functions sampled from a GP using various stationary periodic kernels. Generated by gpKernelPlot.ipynb.

For values $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$, the function simplifies as follows:

$$\mathcal{K}(r; \frac{1}{2}, \ell) = \exp(-\frac{r}{\ell}) \tag{17.11}$$

$$\mathcal{K}(r; \frac{3}{2}, \ell) = \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right) \tag{17.12}$$

$$\mathcal{K}(r; \frac{5}{2}, \ell) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right) \tag{17.13}$$

The value $\nu = \frac{1}{2}$ corresponds to the **Ornstein-Uhlenbeck process**, which describes the velocity of a particle undergoing Brownian motion. The corresponding function is continuous but not differentiable, and hence is very "jagged". See Figure 17.2b for an illustration.

Periodic kernels

The **periodic kernel** captures repeating structure, and has the form

$$\mathcal{K}_{\text{per}}(r; \ell, p) = \exp\left(-\frac{2}{\ell^2}\sin^2(\pi\frac{r}{p})\right) \tag{17.14}$$

where p is the period. See Figure 17.3a for an illustration.

A related kernel is the **cosine kernel**:

$$\mathcal{K}(r; p) = \cos\left(2\pi\frac{r}{p}\right) \tag{17.15}$$

See Figure 17.3b for an illustration.

17.1.2.2 Making new kernels from old

Given two valid kernels $\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')$ and $\mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}')$, we can create a new kernel using any of the following methods:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = c\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}'), \text{ for any constant } c > 0 \tag{17.16}$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}'), \text{ for any function } f \tag{17.17}$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = q(\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')) \text{ for any function polynomial } q \text{ with nonneg. coef.} \tag{17.18}$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp(\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')) \tag{17.19}$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\mathsf{T}\mathbf{A}\boldsymbol{x}', \text{ for any psd matrix } \mathbf{A} \tag{17.20}$$

For example, suppose we start with the linear kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\mathsf{T}\boldsymbol{x}'$. We know this is a valid Mercer kernel, since the corresponding Gram matrix is just the (scaled) covariance matrix of the data. From the above rules, we can see that the polynomial kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\mathsf{T}\boldsymbol{x}')^M$ is a valid Mercer kernel. This contains all monomials of order M. For example, if $M = 2$ and the inputs are 2d, we have

$$(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}')^2 = (x_1x_1' + x_2x_2')^2 = (x_1x_1')^2 + (x_2x_2)^2 + 2(x_1x_1')(x_2x_2') \tag{17.21}$$

We can generalize this to contain all terms up to degree M by using the kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\mathsf{T}\boldsymbol{x}' + c)^M$. For example, if $M = 2$ and the inputs are 2d, we have

$$\begin{aligned}(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}' + 1)^2 = {} & (x_1x_1')^2 + (x_1x_1')(x_2x_2') + (x_1x_1') \\ & + (x_2x_2)(x_1x_1') + (x_2x_2')^2 + (x_2x_2') \\ & + (x_1x_1') + (x_2x_2') + 1\end{aligned} \tag{17.22}$$

We can also use the above rules to establish that the Gaussian kernel is a valid kernel. To see this, note that

$$||\boldsymbol{x} - \boldsymbol{x}'||^2 = \boldsymbol{x}^\mathsf{T}\boldsymbol{x} + (\boldsymbol{x}')^\mathsf{T}\boldsymbol{x}' - 2\boldsymbol{x}^\mathsf{T}\boldsymbol{x}' \tag{17.23}$$

and hence

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\sigma^2) = \exp(-\boldsymbol{x}^\mathsf{T}\boldsymbol{x}/2\sigma^2)\exp(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}'/\sigma^2)\exp(-(\boldsymbol{x}')^\mathsf{T}\boldsymbol{x}'/2\sigma^2) \tag{17.24}$$

is a valid kernel.

17.1.2.3 Combining kernels by addition and multiplication

We can also combine kernels using addition or multiplication:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}') + \mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}') \tag{17.25}$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}') \times \mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}') \tag{17.26}$$

Multiplying two positive-definite kernels together always results in another positive definite kernel. This is a way to get a conjunction of the individual properties of each kernel, as illustrated in Figure 17.4.

In addition, adding two positive-definite kernels together always results in another positive definite kernel. This is a way to get a disjunction of the individual properties of each kernel, as illustrated in Figure 17.5.

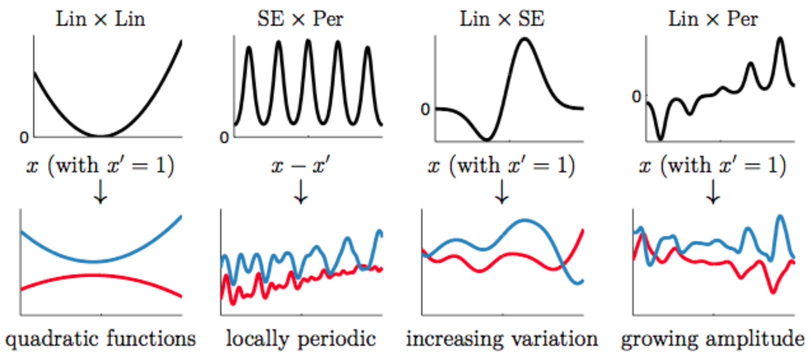


Figure 17.4: Examples of 1d structures obtained by multiplying elementary kernels. Top row shows $\mathcal{K}(x, x' = 1)$. Bottom row shows some functions sampled from $GP(f|0, \mathcal{K})$. From Figure 2.2 of [Duv14]. Used with kind permission of David Duvenaud.

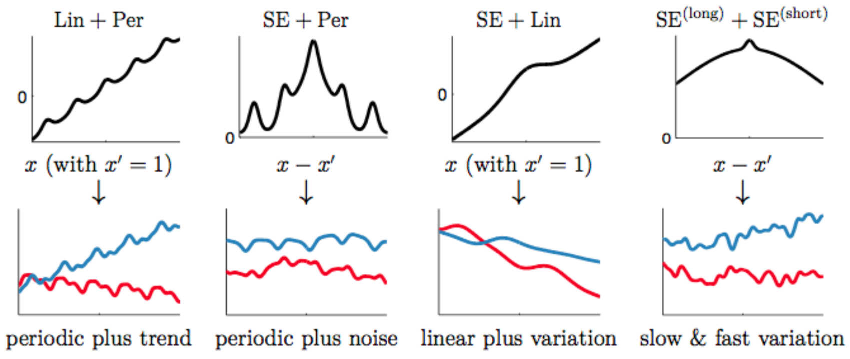


Figure 17.5: Examples of 1d structures obtained by adding elementary kernels. Here $\mathrm{SE}^{(\mathrm{short})}$ and $\mathrm{SE}^{(\mathrm{long})}$ are two SE kernels with different length scales. From Figure 2.4 of [Duv14]. Used with kind permission of David Duvenaud.

17.1.2.4 Kernels for structured inputs

Kernels are particularly useful when the inputs are structured objects, such as strings and graphs, since it is often hard to “featurize” variable-sized inputs. For example, we can define a **string kernel** which compares strings in terms of the number of n-grams they have in common [Lod+02; BC17].

We can also define kernels on graphs [KJM19]. For example, the **random walk kernel** conceptually performs random walks on two graphs simultaneously, and then counts the number of paths that were produced by both walks. This can be computed efficiently as discussed in [Vis+10]. For more details on graph kernels, see [KJM19].

For a review of kernels on structured objects, see e.g., [Gär03].

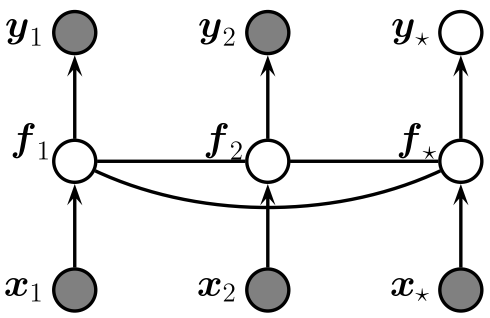


Figure 17.6: A Gaussian process for 2 training points, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and 1 testing point, $\boldsymbol{x}_*$, represented as a graphical model representing $p(\boldsymbol{y}, \boldsymbol{f}_X|\mathbf{X}) = \mathcal{N}(\boldsymbol{f}_X|m(\mathbf{X}), \mathcal{K}(\mathbf{X}))\prod_i p(y_i|f_i)$. The hidden nodes $f_i = f(\boldsymbol{x}_i)$ represent the value of the function at each of the data points. These hidden nodes are fully interconnected by undirected edges, forming a Gaussian graphical model; the edge strengths represent the covariance terms $\Sigma_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. If the test point $\boldsymbol{x}_*$ is similar to the training points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, then the value of the hidden function f_* will be similar to f_1 and f_2, and hence the predicted output y_* will be similar to the training values y_1 and y_2.

17.2 Gaussian processes

In this section, we discuss **Gaussian processes**, which is a way to define distributions over functions of the form $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is any domain. The key assumption is that the function values at a set of $M > 0$ inputs, $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_M)]$, is jointly Gaussian, with mean $(\boldsymbol{\mu} = m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_M))$ and covariance $\boldsymbol{\Sigma}_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where m is a mean function and $\mathcal{K}$ is a positive definite (Mercer) kernel. Since we assume this holds for any $M > 0$, this includes the case where $M = N + 1$, containing N training points $\boldsymbol{x}_n$ and 1 test point $\boldsymbol{x}_*$. Thus we can infer $f(\boldsymbol{x}_*)$ from knowledge of $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)$ by manipulating the joint Gaussian distribution $p(f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N), f(\boldsymbol{x}_*))$, as we explain below. We can also extend this to work with the case where we observe noisy functions of $f(\boldsymbol{x}_n)$, such as in regression or classification problems.

17.2.1 Noise-free observations

Suppose we observe a training set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n) : n = 1 : N\}$, where $y_n = f(\boldsymbol{x}_n)$ is the noise-free observation of the function evaluated at $\boldsymbol{x}_n$. If we ask the GP to predict $f(\boldsymbol{x})$ for a value of $\boldsymbol{x}$ that it has already seen, we want the GP to return the answer $f(\boldsymbol{x})$ with no uncertainty. In other words, it should act as an **interpolator** of the training data.

Now we consider the case of predicting the outputs for new inputs that may not be in $\mathcal{D}$. Specifically, given a test set $\mathbf{X}_*$ of size $N_* \times D$, we want to predict the function outputs $\boldsymbol{f}_* = [f(\boldsymbol{x}_{*1}), \ldots, f(\boldsymbol{x}_{*,N_*})]$. By definition of the GP, the joint distribution $p(\boldsymbol{f}_X, \boldsymbol{f}_*|\mathbf{X}, \mathbf{X}_*)$ has the following form

$$\begin{pmatrix} \boldsymbol{f}_X \\ \boldsymbol{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\mathsf{T} & \mathbf{K}_{*,*} \end{pmatrix}\right) \tag{17.27}$$