Chapter 1

Expressing Structure with Kernels

In this chapter, we'll show how to use kernels (also called *covariance functions*) to build many different kinds of models of functions. By combining a few simple kernels through addition and multiplication, we'll be able to express many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, and many types of invariances. Combining kernels in a few simple ways gives us a rich language with which to build appropriate models.

1.1 Definition

Since we'll be discussing covariance functions at length, we now give a precise definition. A kernel $k(x, x') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is any positive-definite function between two points x, x' in some space $\mathcal{X}$. In this chapter, $\mathcal{X}$ is usually taken to be Euclidian space, but it doesn't have to be. As we shall see, $\mathcal{X}$ can also be the space of images, documents, categories or points on a sphere, for example.

Gaussian process models use a kernel to define the prior covariance between any two function values:

$$\mathrm{Cov}\left(f(x), f(x')\right) = k(x, x') \tag{1.1}$$

Colloquially, kernels are often said to specify the similarity between two objects. This is slightly misleading in the context of GP, since what is actually being specified is the similarity between two values of a *function* of an object. The kernel specifies which functions are likely under the GP prior, which in turn determines the generalization properties of the model.

1.2 Basic Kernels

Commonly used kernels families include the squared exponential (SE), periodic (Per), linear (Lin), and rational quadratic (RQ). These kernels are defined in figure 1.1. There is nothing special about these kernels in particular, except that they represent a

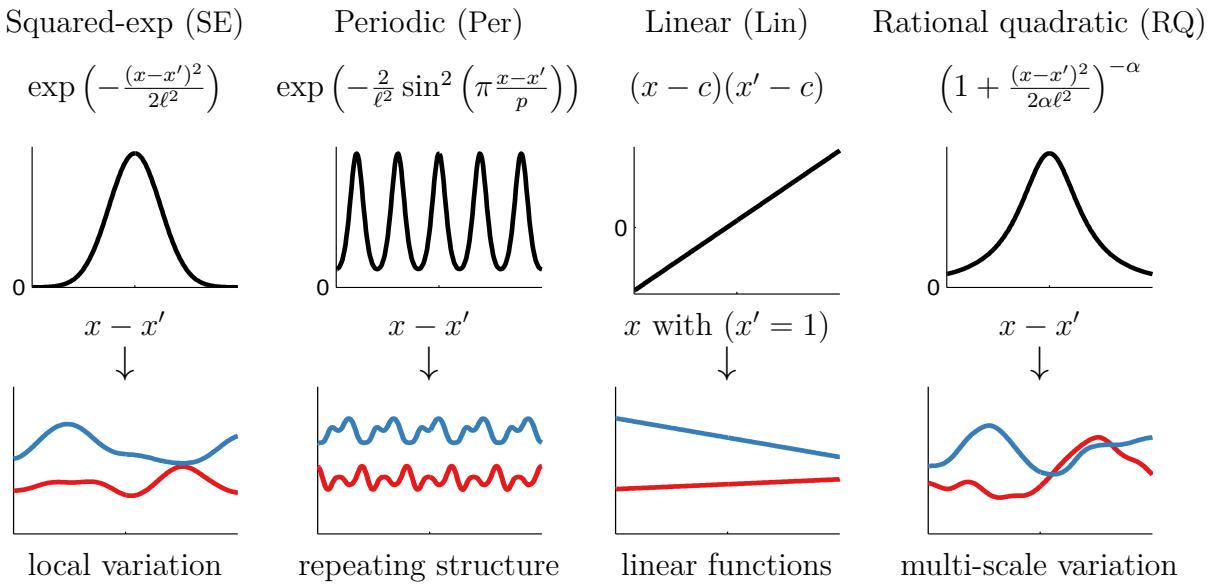


Fig. 1.1 Examples of structures expressible by base kernels. Left and third columns: base kernels $k(\cdot, 0)$. Second and fourth columns: draws from a GP with each repective kernel. The x-axis has the same range on all plots.

diverse set.

Each covariance function corresponds to a different set of assumptions made about the function we wish to model. For example, using a squared-exp (SE) kernel implies that the function we are modeling has infinitely many derivatives.

1.3 Combining Kernels

The rest of this chapter will explore ways in which kernels can be combined to create new ones with different properties. Using a few simple base kernels and operations for combining them will give us a rich language with which to build models with diverse sorts of properties.

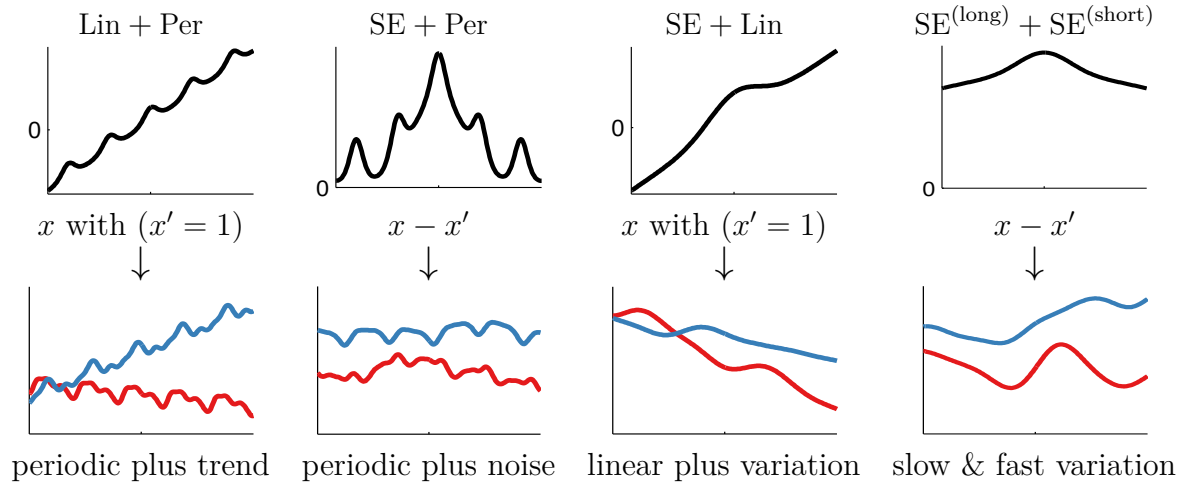


Fig. 1.2 Examples of one-dimensional structures expressible by adding kernels. Plots have same meaning as in figure 1.1.

1.3.1 Notation

In this chapter, we'll focus on two ways of combining kernels: addition and multiplication, which we'll often write in shorthand without arguments:

$$k_1 + k_2 = k_1(x, x') + k_2(x, x') \tag{1.2}$$

$$k_1 \times k_2 = k_1(x, x') \times k_2(x, x') \tag{1.3}$$

All of the base kernels we'll consider in this chapter are one-dimensional. Kernels over multidimensional inputs are constructed by adding and multiplying kernels over individual dimensions. These dimensions are represented using subscripts, e.g. SE_2 represents an SE kernel over the second dimension of x.

Each kernel has a number of parameters (sometimes called *hyperparameters*) which specify the precise shape of the covariance function. To remove clutter, we'll usually refer to a kernel without specifying these parameters.

1.4 Modeling Additive Functions

Additivity is a very useful modeling assumption in a wide variety of contexts. It often aids in building interpretable models, as well as enabling extrapolation in high-dimensional settings. Fortunately, this assumption is easy to encode in GP models.

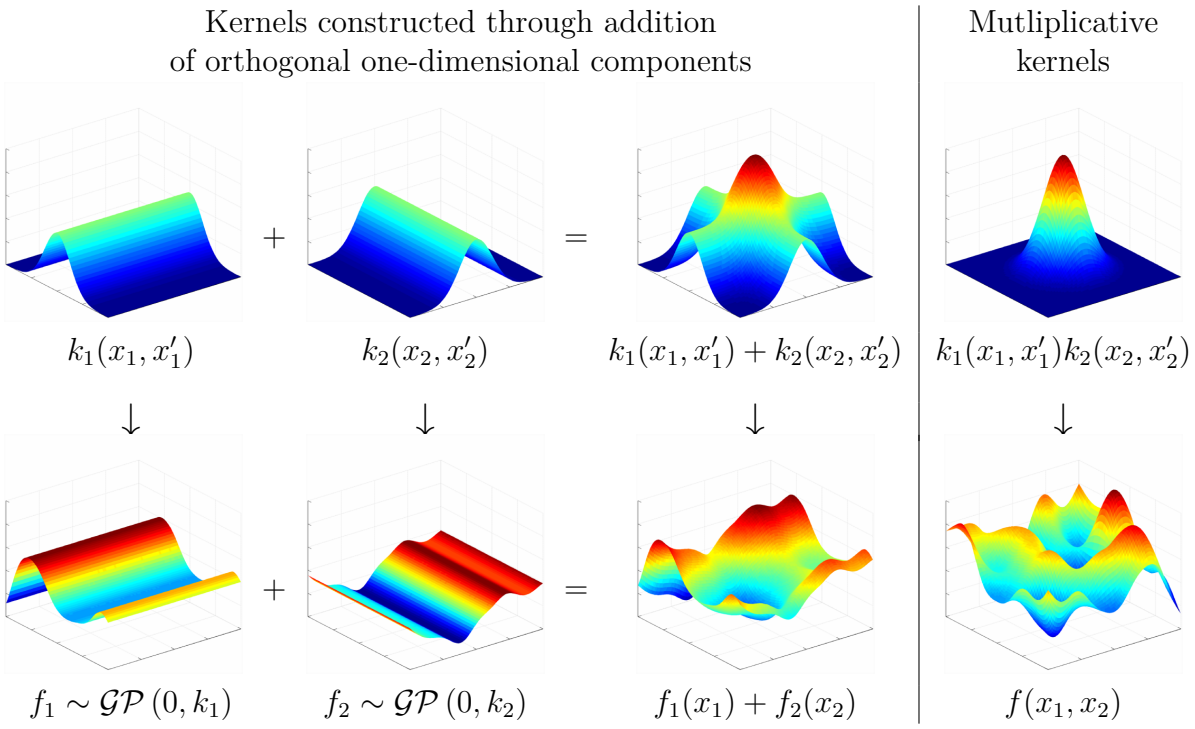


Fig. 1.3 Top left: An additive kernel is a sum of kernels. Bottom left: A draw from an additive kernel corresponds to a sum of draws from independent GP priors with the corresponding kernels. Top right: A product kernel. Bottom right: A GP prior with a product of kernels does not correspond to a product of draws from GPs.

By summing kernels, we can model the data as a sum of independent functions, each possibly representing a different type of structure. Suppose functions f_1, f_2 are drawn independently from GP priors:

$$f_1 \sim \mathcal{GP}(\mu_1, k_1) \tag{1.4}$$

$$f_2 \sim \mathcal{GP}(\mu_2, k_2) \tag{1.5}$$

Then we can model the sum of those functions through another GP:

$$f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2). \tag{1.6}$$

Kernels k_1 and k_2 can be kernels of any type, and we can sum up any number of kernels this way.

1.4.1 Additivity Across Multiple Dimensions

When modeling functions of multiple dimensions, summing kernels can give rise to additive structure across different dimensions. To be more precise, if the kernels being added together are functions only of a subset of input dimensions, then the implied prior over functions decomposes in the same way. For example, if

$$f(x_1, x_2) \sim \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1') + k_2(x_2, x_2')) \tag{1.7}$$

Then this is equivalent to the model

$$f_1(x_1) \sim \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1')) \tag{1.8}$$
$$f_2(x_2) \sim \mathcal{GP}(\mathbf{0}, k_2(x_2, x_2')) \tag{1.9}$$
$$f(x_1, x_2) = f_1(x_1) + f_2(x_2) \tag{1.10}$$

Figure 1.3 illustrates this decomposition. Note that the product of two kernels does not have an analogous interpretation as the product of two functions.

1.4.2 Example: Additive model of Concrete Compressive Strength

We now give an example of how additive kernels give rise to interpetable models. We model measurements of the compressive strength of concrete, as a function of the concentration of 7 ingredients, plus the age of the concrete (Yeh, 1998).

We build an additive model

$$\begin{aligned} f(\mathbf{x}) = f_1(\text{cement}) + f_2(\text{slag}) + f_3(\text{fly ash}) + f_4(\text{water}) \\ + f_5(\text{plasticizer}) + f_6(\text{coarse}) + f_7(\text{fine}) + f_8(\text{age}) + \epsilon \end{aligned} \tag{1.11}$$

where $\epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_n)$. After learning the hyperparameters of the model by maximizing the marginal likelihood conditioned on the data, we can visualize the posterior distribution of each component of the model.

Figure 1.4 shows the marginal posterior distribution of each of the 8 components in Equation (1.11). We can see that the hyperparameters controlling the variance of two of the components, Coarse and Fine, were set to zero, meaning that the marginal likelihood preferred a parsimonious model which did not depend on these dimensions. This is an example of the automatic sparsity that arises by maximizing marginal likelihood in GP models.

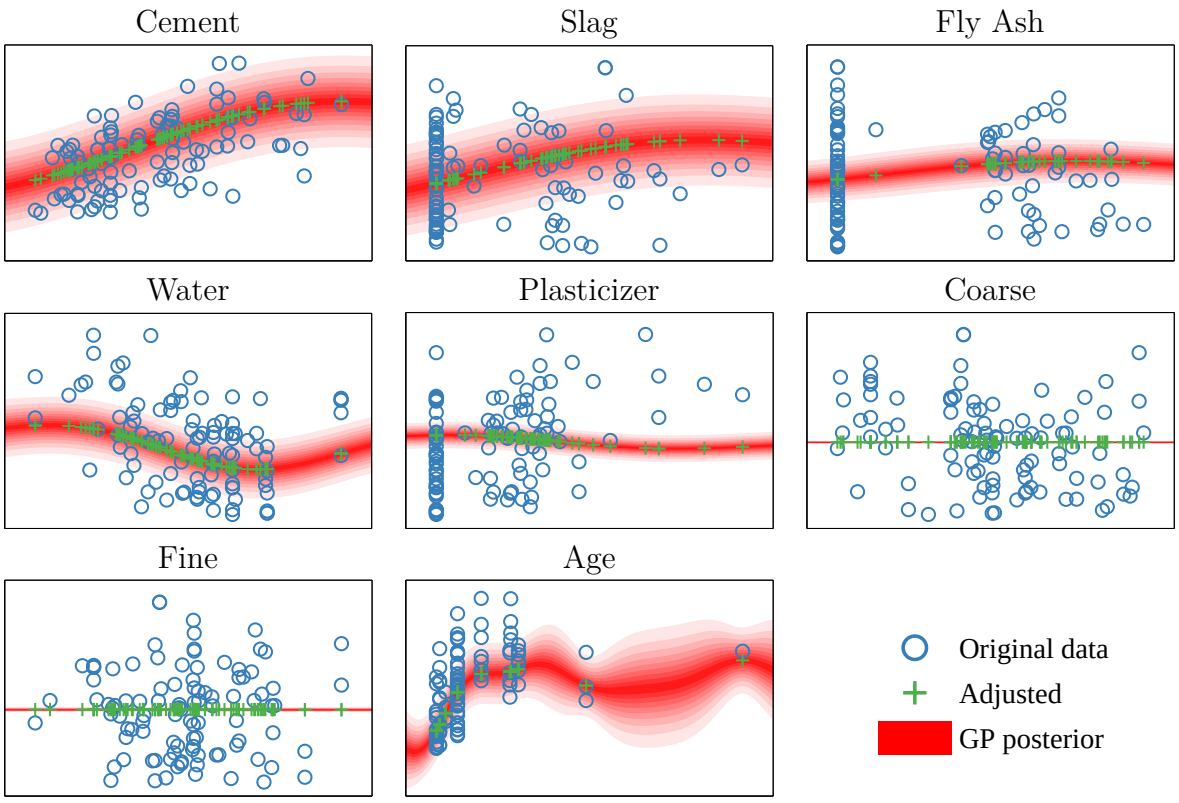


Fig. 1.4 By considering only one-dimensional terms of the additive kernel, we recover a form of Generalized Additive Model, and can plot the corresponding 1-dimensional functions. Blue points indicate the original data, green points are data after the contribution from all other terms has been subtracted. The vertical axis is the same for all plots.

1.4.3 Marginal Variance of Components

How to compute the posterior distributions shown in Figure 1.4? Here we derive the posterior marginal variance and covariance of each of the additive components of a GP.

First, we'll write down the joint prior over the sum of two functions. We'll distinguish between $\boldsymbol{f}(\mathbf{X})$ (the function values at the training locations) and $\boldsymbol{f}(\mathbf{X}^\star)$ (the function values at some set of query locations).

If f_1 and f_2 are *a priori* independent, and $f_1 \sim \mathrm{GP}(\mu_1, k_1)$ and $f_2 \sim \mathrm{GP}(\mu_2, k_2)$, then

$$\begin{bmatrix} \boldsymbol{f}_1(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) \\ \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_2(\mathbf{x}^\star) \\ \boldsymbol{f}_1(\mathbf{x}) + \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) + \boldsymbol{f}_2(\mathbf{x}^\star) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_1^\star \\ \mu_2 \\ \mu_2^\star \\ \mu_1 + \mu_2 \\ \mu_1^\star + \mu_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_1^\star & 0 & 0 & \mathbf{K}_1 & \mathbf{K}_1^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & 0 & 0 & \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} \\ 0 & 0 & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_2 & \mathbf{K}_2^\star \\ 0 & 0 & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} \\ \mathbf{K}_1 & \mathbf{K}_1^\star & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_1^\star + \mathbf{K}_2^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_1^\star + \mathbf{K}_2^\star & \mathbf{K}_1^{\star\star} + \mathbf{K}_2^{\star\star} \end{bmatrix} \right) \tag{1.12}$$

where we represent the Gram matrices, evaluated at all pairs of vectors in bold capitals as $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. So

$$\mathbf{K}_1 = k_1(\mathbf{X}, \mathbf{X}) \tag{1.13}$$

$$\mathbf{K}_1^\star = k_1(\mathbf{X}^\star, \mathbf{X}) \tag{1.14}$$

$$\mathbf{K}_1^{\star\star} = k_1(\mathbf{X}^\star, \mathbf{X}^\star) \tag{1.15}$$

By the formula for Gaussian conditionals:

$$\mathbf{x}_A | \mathbf{x}_B \sim \mathcal{N}(\mu_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\left(\mathbf{x}_B - \mu_B\right), \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA}), \tag{1.16}$$

we get that the conditional distribution of a GP-distributed function conditioned on its sum with another GP-distributed function is given by

$$\begin{aligned} \boldsymbol{f}_1^\star | \boldsymbol{f} \sim \mathcal{N}\Big(& \mu_1^\star + \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\left(\boldsymbol{f} - \mu_1 - \mu_2\right) \\ & \mathbf{K}_1^{\star\star} - \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\mathbf{K}_1^\star \Big) \end{aligned} \tag{1.17}$$

These formulae express the posterior model uncertainty about different components of the signal, integrating over the possible configurations of the other components. In the case where we condition on the sum of more than two functions, the term $\mathbf{K}_1 + \mathbf{K}_2$ can simply be replaced by $\sum_i \mathbf{K}_i$ everywhere.

1.4.4 Marginal Covariance of Components

One of the advantages of using a generative, model-based approach is that we can examine any aspect of the model we wish to. For instance, we can compute the posterior

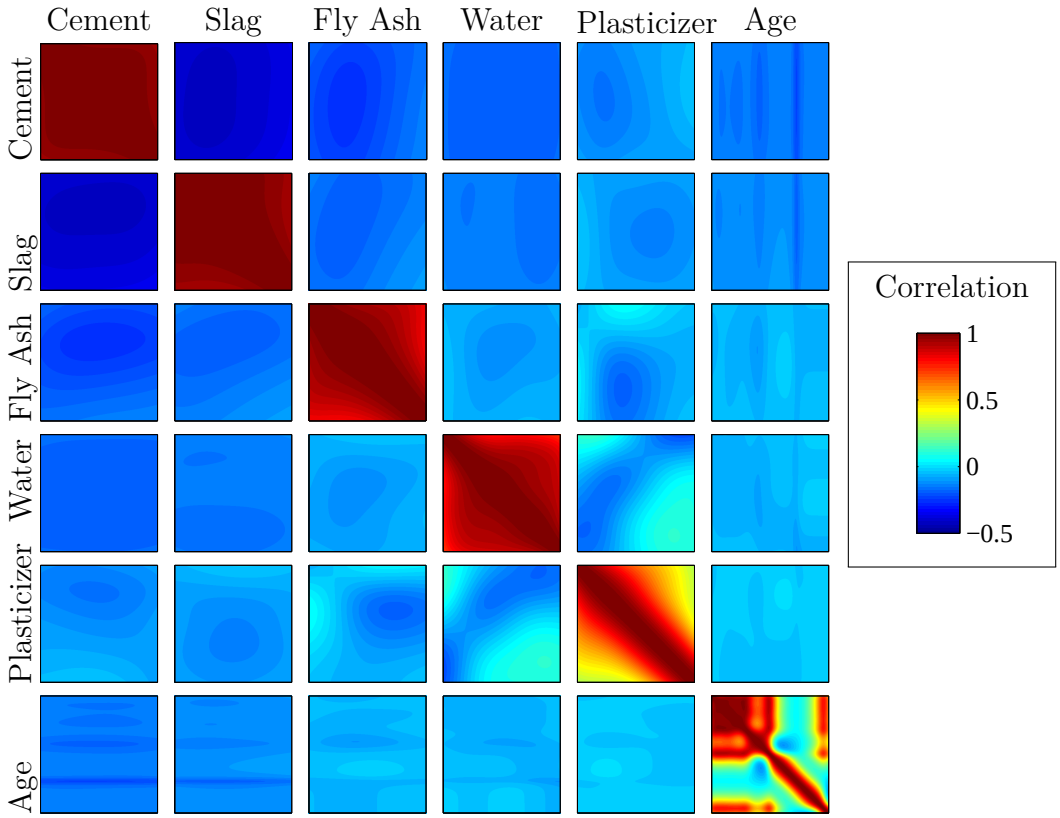


Fig. 1.5 Visualizing posterior correlations between the components explaining the concrete dataset. Each plot shows the additive model's posterior correlation between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. There is negative correlation between the "Cement" and "Slag" variables, meaning that - one of these functions is high and the other low, but which one is uncertain under the model. Dimensions 'Coarse' and 'Fine' are not shown, because their variance was zero.

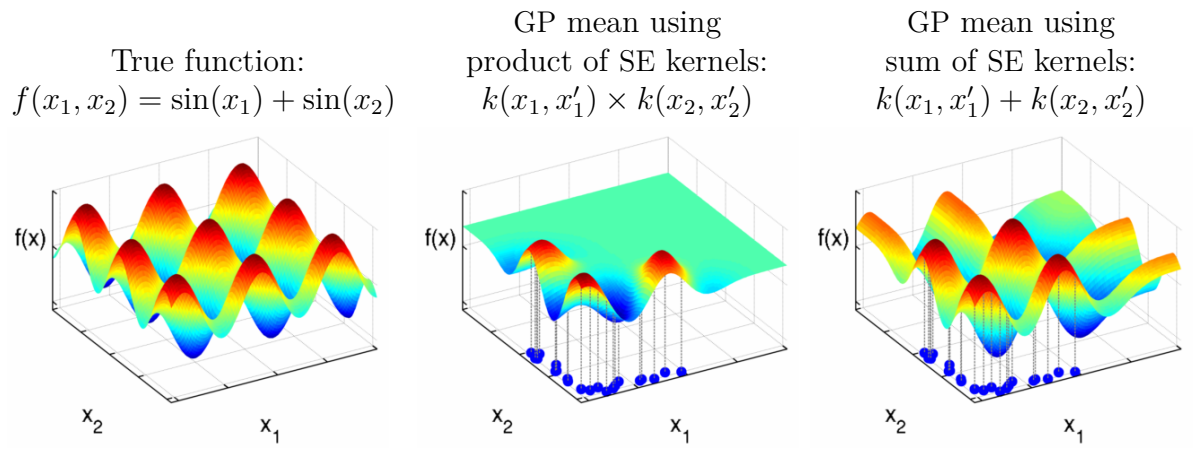


Fig. 1.6 Inference in functions with additive structure. When the function being modeled has additive structure, we can exploit this fact to extrapolate far from the training data. The product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data.

covariance between any two components, conditioned on their sum:

$$\text{cov}\left[\boldsymbol{f}_1^\star, \boldsymbol{f}_2^\star | \boldsymbol{f}\right] = -\mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\mathbf{K}_2^\star \tag{1.18}$$

If this quantity is negative, it means that there is ambiguity about which of the two components explains the data at that location.

For example, Figure 1.5 plots the posterior correlation between all non-zero components of the concrete model. We can see that there is negative correlation between the "Cement" and "Slag" variables. This reflects an ambiguity in the model about which one of these functions is high and the other low.

1.4.5 Long-range Extrapolation Through Additivity

Finding additive structure is useful, because it allows us to make predictions far from the training data. Figure 1.6 compares the extrapolations made by additive versus non-additive GP models, conditioned on data from a sum of two axis-aligned sine functions, evaluated in a small, L-shaped area. In this example, the additive model is able to correctly predict the value of the function at unseen combinations of inputs. The product-kernel model is more flexible, and so remains uncertain about the function value at unseen combinations of inputs.

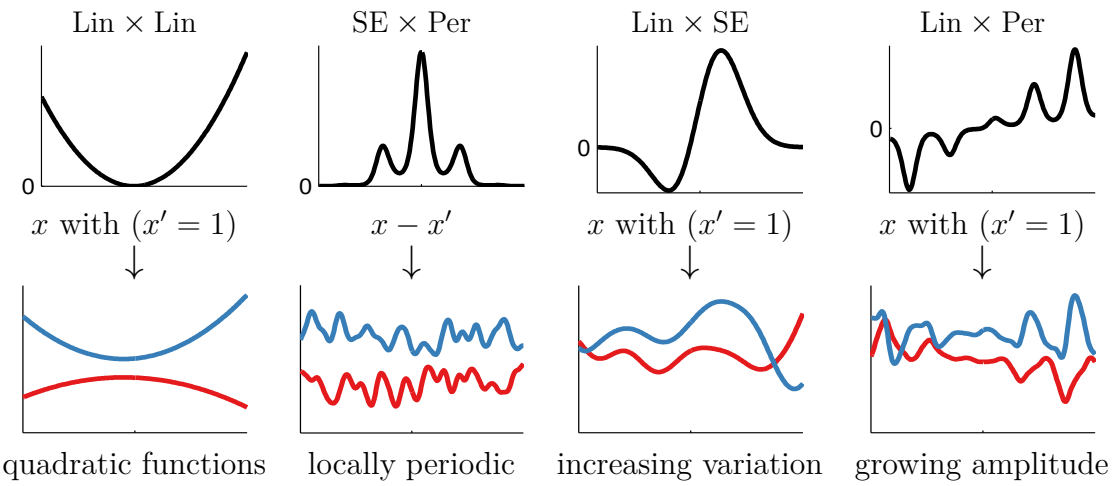


Fig. 1.7 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 1.1.

These types of additive models have been well-explored in the statistics literature. For example, generalized additive models (Hastie and Tibshirani, 1990) have seen wide adoption. In high dimensions, we can also consider sums of functions of more than one dimension. Chapter **??** considers this model class in more detail.

1.5 Modeling Interactions

Multiplying kernels allows us to account for interactions between different input dimensions or different notions of similarity. For instance, in multidimensional data, the multiplicative kernel $\mathrm{SE}_1 \times \mathrm{SE}_3$ represents a smoothly varying function of dimensions 1 and 3 which is not constrained to be additive. In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to globally periodic structure, whereas Per × SE corresponds to locally periodic structure, as shown in row 1 of figure 1.3.

Many architectures for learning complex functions, such as convolutional networks LeCun et al. (1989) and sum-product networks Poon and Domingos (2011), include units which compute AND-like and OR-like operations. Composite kernels can be viewed in this way too. A sum of kernels can be understood as an OR-like operation: two points are considered similar if either kernel has a high value. Similarly, multiplying kernels is an AND-like operation, since two points are considered similar only if both kernels have high values. Since we are applying these operations to the similarity functions rather

than the regression functions themselves, compositions of even a few base kernels are able to capture complex relationships in data which do not have a simple parametric form.

1.5.1 Expressing Multiplication with a Known Function

If we wish to model a function that's been multiplied by some fixed, known function $a(x)$, this can be easily achieved by multiplying the kernel by $a(\mathbf{x})a(\mathbf{x}')$.

This is why it is common practice to assume zero mean, since marginalizing over an unknown mean function can be equivalently expressed as a zero-mean GP with a new kernel.

1.6 Feature Representation

By Mercer's theorem (Mercer, 1909), any positive-definite kernel can be represented as the inner product between a fixed set of features, evaluated at x and at x':

$$k(x, x') = \mathbf{h}(x)^\mathsf{T}\mathbf{h}(x') \tag{1.19}$$

As a simple example, the squared-exponential kernel (SE) on the real line has a representation in terms of infinitely many radial-basis functions. Any stationary kernel (one which only depends on the distance between its inputs) on the real line can be represented by a set of sines and cosines - a Fourier representation (Bochner, 1959). In general, the feature representation of a kernel is not unique, and depends on which space $\mathcal{X}$ is being considered (Minh et al., 2006).

In some cases, $\mathcal{X}$ can even be the infinite-dimensional feature mapping of another kernel. Composing feature maps in this way leads to *deep kernels*, a topic explored in chapter **??**.

1.6.1 Relation to Linear Regression

Suprisingly, GP regression is equivalent to Bayesian linear regression on $\mathbf{h}(x)$:

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{h}(\mathbf{x}), \quad \mathbf{w} \sim \mathcal{N}(\mathbf{w}|\mathbf{0}, \Sigma) \quad \iff \quad f \sim \mathcal{GP}\Big(f \,\Big|\, \mathbf{0}, \mathbf{h}(\mathbf{x})^\mathsf{T}\Sigma\mathbf{h}(\mathbf{x})\Big) \tag{1.20}$$

The link between Gaussian processes, linear regression, and neural networks is explored further in chapter **??**.

1.6.2 Feature-space view of combining kernels

We can also view kernel addition and multiplication as a combination of the features of the original kernels.

Viewing kernel addition from this point of view, if

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}') \tag{1.21}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{1.22}$$

then

$$k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}) = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') + \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{1.23}$$

$$= \begin{bmatrix} \mathbf{a}(\mathbf{x}) \\ \mathbf{b}(\mathbf{x}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{a}(\mathbf{x}') \\ \mathbf{b}(\mathbf{x}') \end{bmatrix} \tag{1.24}$$

meaning that the features of $k_a + k_b$ are the concatenation of the features of each kernel.

We can examine kernel multiplication in a similar way:

$$k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') = \left[\mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}')\right] \times \left[\mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}')\right] \tag{1.25}$$

$$= \sum_i a_i(\mathbf{x})a_i(\mathbf{x}') \times \sum_j b_j(\mathbf{x})b_j(\mathbf{x}') \tag{1.26}$$

$$= \sum_i \sum_j a_i(\mathbf{x})a_i(\mathbf{x}')b_j(\mathbf{x})b_j(\mathbf{x}') \tag{1.27}$$

$$= \sum_{i,j} \left[a_i(\mathbf{x})b_j(\mathbf{x})\right]\left[a_i(\mathbf{x}')b_j(\mathbf{x}')\right] \tag{1.28}$$

$$= \mathrm{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right)^\mathsf{T} \mathrm{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right) \tag{1.29}$$

In other words, the features of $k_a \times k_b$ are the cartesian product (all possible combinations) of the original sets of features. For example, one set of features $\mathbf{h}(\mathbf{x})$ corresponding to a one-dimensional SE kernel is a set of infinitely many Gaussian bumps spread along the real line. Multiplied by an SE kernel in another dimension, the resulting features are a set of two-dimensional Gaussian bumps, covering the plane.

1.7 Expressing Symmetries

When modeling functions, encoding known symmetries improves both learning and prediction. Many types of symmetry can be enforced through operations on the covariance

function. In this section, we'll look at the different ways in which we can encode a given symmetry in a our prior on functions.

We'll demonstrate the properties of the resulting models by sampling functions from their priors. By using these priors to define warpings from $\mathbb{R}^2 \to \mathbb{R}^3$, we'll also show how to build a nonparametric prior on an open-ended family of topological manifolds, such as cylinders, torii, and Möbius strips.

Ginsbourger et al. (2012) and Ginsbourger et al. (2013) characterized the set of GP priors on functions invariant to given transformations. They showed that the only way to construct a prior on functions which respect a given invariance, is to construct a kernel which is invariant to the same invariance. Formally, given a symmetry Φ, and $f \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$, f is invariant under the action of a finite group Φ if and only if k is argumentwise invariant:

$$\forall \mathbf{x}, \forall g \in G, \quad k(g(\mathbf{x}), x') = k(\mathbf{x}, \mathbf{x}') \tag{1.30}$$

Knowing the orbit of a symmetry group and an existing kernel, there are a few simple ways to transform that kernel into one which is argumentwise invariant, and thus obeys the group's symmetries:

- **Sum over the Orbit** Ginsbourger et al. (2012) and Kondor (2008) suggest simply summing over the orbit:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \sum_{g \in G} k(\mathbf{x}, T(\mathbf{x}')) \tag{1.31}$$

- **Product over the Orbit** (Ryan P. Adams, personal communication) suggests a similar construction using a product over the orbit:

$$k_{\text{prod}}(\mathbf{x}, \mathbf{x}') = \prod_{g \in G} k(\mathbf{x}, T(\mathbf{x}')) \tag{1.32}$$

- **Project to a Representer** Ginsbourger et al. (2013) also explore the possibility of projecting each datapoint into a representer A of the group:

$$k_{\text{rep}}(\mathbf{x}, \mathbf{x}') = k(A_G(\mathbf{x}), A_G(\mathbf{x}')) \tag{1.33}$$

There are presumably many possible ways to achive a given symmetry, but we must be careful to do so without compromising other qualities of the model we are construct-

ing. For example, simply setting $k(\mathbf{x}, \mathbf{x}') = 0$ gives rise to a GP prior which obeys *all possible* symmetries, but this is presumably not a model we wish to use.

1.7.1 Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

$$f(x) = f(x + \tau) \tag{1.34}$$

where τ is the period.

The most popular method for building a periodic kernel is due to **?**, who used the representer method in combination with a squared-exp kernel.

The representer transformation for periodicity is simply $A(x) = [\sin(x), \cos(x)]$:

$$k_{\text{periodic}}(x, x') = \text{SE}([\sin(x), \cos(x)], [\sin(x'), \cos(x')]) \tag{1.35}$$

1.7.2 Reflective Symmetry

Reflective Symmetry Along an Axis

W e can enforce the symmetry

$$f(x) = f(-x) \tag{1.36}$$

by the kernel transform

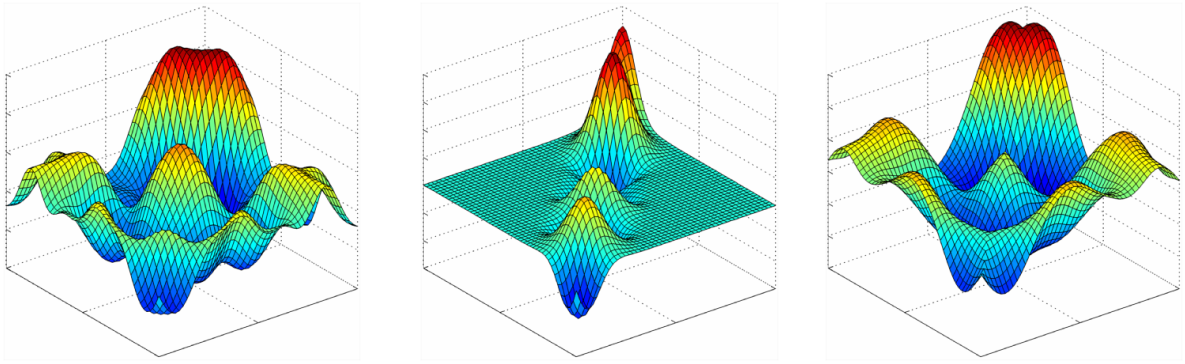
$$k_{\text{symm arg1}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{1.37}$$

Note that, because $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}')$ for any PSD kernel, so we don't need to include the corresponding terms $k(-x, x')$ and $k(-x, -x')$.

Reflective Symmetry Along a Diagonal

We can enforce that a function is invariant to the order of two of its arguments:

$$f(x, y) = f(y, x) \tag{1.38}$$



Additive method

$k(x, y, x', y') + k(x, y, y', x')$

Product method

$k(x, y, x', y') \times k(x, y, y', x')$

Representer method

$k(\min(x, y), \max(x, y), \min(x', y'), \max(x', y'))$

Fig. 1.8 An illustration of three methods of introducing symmetry. Left: The additive method. Center: The product method. Right: The representer method. All three methods introduce a different type of nonstationarity.

by two methods: In the additive method, we transform the kernel by:

$$k_{\text{reflect add}}(x, y, x', y') = k(x, y, x', y') + k(x, y, y', x') + k(y, x, x', y') + k(y, x, y', x') \tag{1.39}$$

or by

$$k_{\text{reflect min}}(x, y, x', y') = k(\min(x, y), \max(x, y), \min(x', y'), \max(x', y')) \tag{1.40}$$

however, the second method will in general lead to non-differentiability along $x = y$. Figure 1.8 shows the difference.

1.7.3 Example: Computing Molecular Energies

Figure 1.9 gives one example of a function which obeys the same symmetries as a Möbius strip, in some subsets of its arguments.

1.7.4 Example: Translation Invariance in Images

Most models of images are invariant to spatial translations [cite convolution nets]. Similarily, most models of sounds are also invariant to translation through time.

Note that this sort of translational invariance is completely distinct from the station-

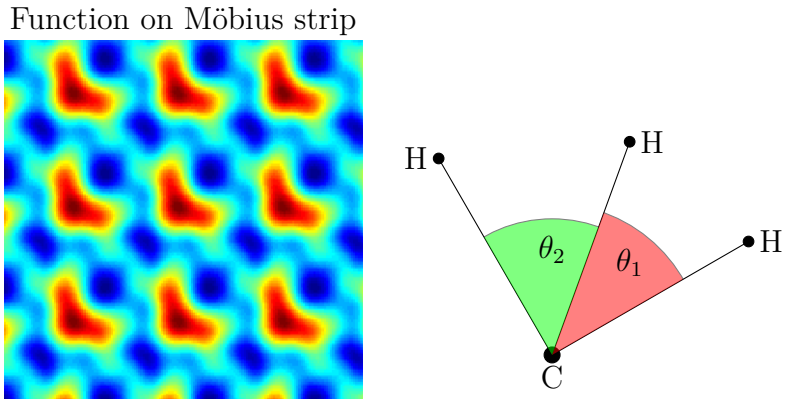


Fig. 1.9 An example of a function expressing the same symmetries as a Möbius strip in two of its arguments. The energy of a molecular configuration $f(\theta_1, \theta_2)$ depends only on the relative angles between atoms, and because each atom is indistiguishable, is invariant to permuting the atoms.

arity properties of kernels used in Gaussian process priors. A stationary kernel implies that the prior is invariant to translations of the entire training and test set.

We are discussing here a discretized input space (into pixels or the audio equivalent), where the input vectors have one dimension for every pixel. We are interested in creating priors on functions that are invariant to shifting a signal along its pixels:

$$f\left(\quad\right) = f\left(\quad\right) \tag{1.41}$$

Translational invariance in this setting is equivalent to symmetries between dimensions in the input space.

This prior can be achieved in one dimension by using the following kernel transformation:

$$k\left((x_1, x_2, \ldots, x_D), (x'_1, x'_2, \ldots, x'_D)\right) = \sum_{i=1}^{D} \prod_{j=1}^{D} k(x_j, x'_{i+j \bmod D}) \tag{1.42}$$

Edge effects can be handled either by wrapping the image around, or by padding it with zeros.

Convolution The resulting kernel could be called a *discrete convolution kernel*. For an image with R, C rows and columns, it can also be written as:

$$k_{\text{conv}}\left((x_{11}, x_{12}, \ldots, x_{RC}), (x'_{11}, x'_{12}, \ldots, x'_{RC})\right) = \sum_{i=-L}^{L} \sum_{j=-L}^{L} k\left(\mathbf{x}, T_{ij}(\mathbf{x}')\right) \tag{1.43}$$

where $T_{ij}(\mathbf{x})$ is the operator which replaces each x_{mn} with $x_{m+i,n+j}$. Thus we are simply defining the covariance between two images to be the sum of all covariances between all relative translations of the two images.

1.8 Generating Topological Manifolds

In this section, we give a geometric illustration of the symmetries encoded by GPs with different sorts of kernels. The language of models of functions exhibiting symmetries, can be used to create a prior on surfaces, by warping a latent surface $\mathbf{x}$ to an observed surface $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$. The distribution on $\boldsymbol{f}$ allows us to put mass on

First create a mesh in 2d. Then draw 3 independent functions from a GP prior with the relevant symmetries encoded in the kernel. Then, map the 2d points making up the mesh through those 3 functions to get the 3D coordinates of each point on the mesh.

This is similar in spirit to the GP-LVM model Lawrence (2005), which learns an embedding of the data into a low-dimensional space, and constructs a fixed kernel structure over that space.

1.8.1 Surfaces, Cylinders and Torii

Figure 1.10 shows...

1.8.2 Möbius Strips

A prior on functions on Möbius strips can be achieved by enforcing the symmetries:

$$f(x, y) = f(x, y + \tau_y) \tag{1.44}$$

$$f(x, y) = f(x + \tau_x, y) \tag{1.45}$$

$$f(x, y) = f(y, x) \tag{1.46}$$

If we imagine moving along the edge of a Möbius strip, that is equivalent to moving along a diagonal in the function generated. Figure 1.11 shows this. The second example

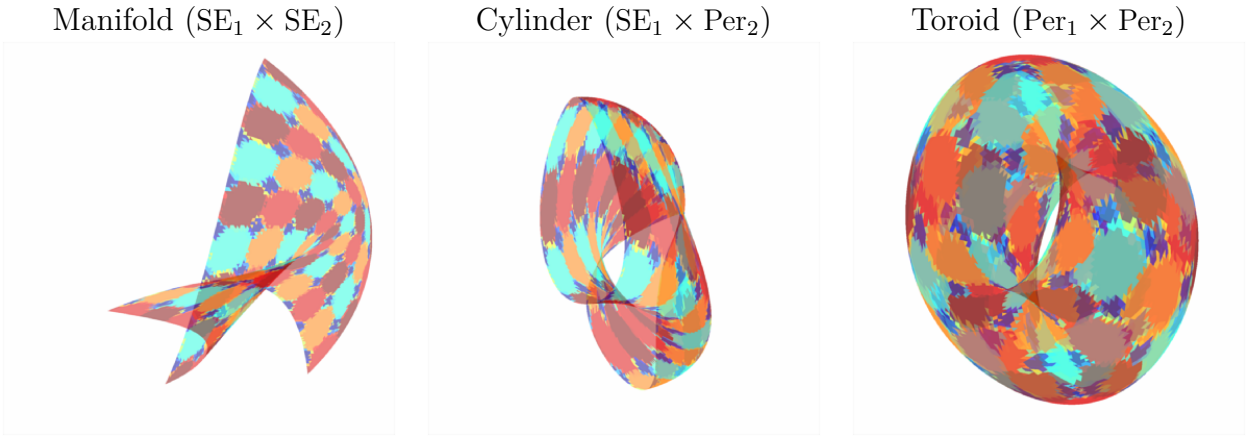


Fig. 1.10 Generating 2D manifolds with different topological structures. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have the corresponding topologies, ignoring self-intersections.

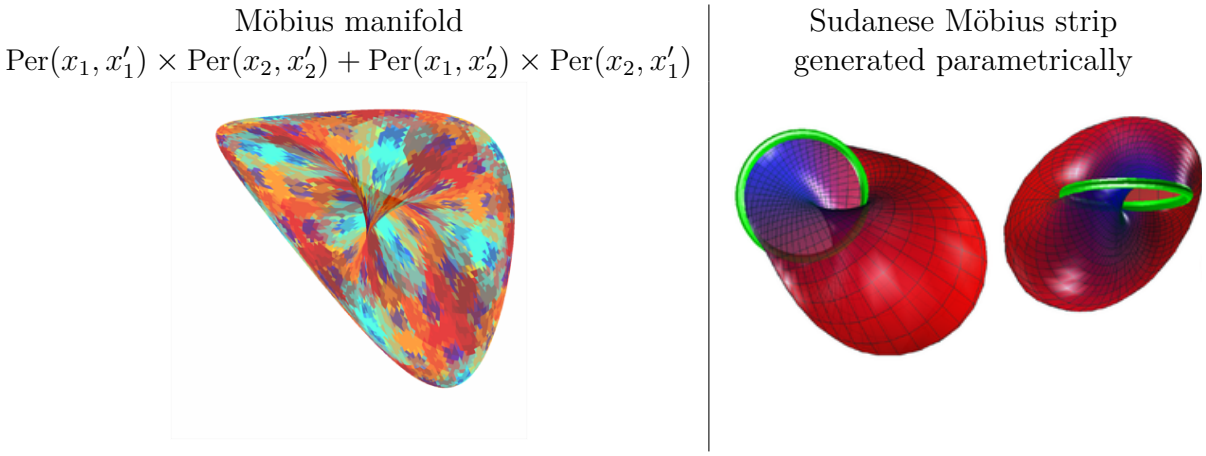


Fig. 1.11 Generating Möbius strips. Left: By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, surfaces sampled from the prior have topology corresponding to a Möbius strip. Möbius strips generated this way do not have the familiar shape of a circular flat surface with a half-twist; rather they tend to look like *Sudanese* Möbius strips (Lerner and Asimov, 1984), whose edge has a circular shape. Right: A Sudanese projection of a Möbius strip. Image adapted from (Commons, 2005).

is doesn't resemble a typical Möbius strip because the edge of the mobius strip is in a geometric circle. This kind of embedding is resembles the Sudanese Möbius strip [cite].

Another classic example of a function living on a Mobius strip is the auditory quality of 2-note intervals. The harmony of a pair of notes is periodic (over octaves) for each note, and the

1.9 Kernels on Discrete Data

Kernels can be defined over all types of data structures: Text, images, matrices, and even kernels . Coming up with a kernel on a new type of data used to be an easy way to get a NIPS paper.

1.9.1 Categorical Variables

There is a simple way to do GP regression over categorical variables. Simply represent your categorical variable as a by a one-of-k encoding. This means that if your number ranges from 1 to 5, represent that as 5 different data dimensions, only one of which is on at a time.

Then, simply put a product of SE kernels on those dimensions. This is the same as putting one SE ARD kernel on all of them. The lengthscale hyperparameter will now encode whether, when that coding is active, the rest of the function changes. If you notice that the estimated lengthscales for your categorical variables is short, your model is saying that it's not sharing any information between data of different categories.

1.10 Why Assume Zero Mean?

In literature, as well as in practice, it is common to construct GP priors with a zero mean function. This might seem strange, since it is presumably a good place to put prior information, or if we are comparing models, to express since marginalizing over an unknown mean function can be equivalently expressed as a different GP with zero-mean, and another term added to the kernel.

A known mean function can be moved into the covariance function Specifically, if we wish to model an unknown function $f(\mathbf{x})$ with known mean $m(\mathbf{x})$, (with unknown magnitude $c \sim \mathcal{N}(0, \sigma_c^2)$), we can equivalently express this model using another

GP with zero mean:

$$f \sim \mathcal{GP}(cm(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad c \sim \mathcal{N}\left(0, \sigma_c^2\right) \iff f \sim \mathcal{GP}\left(\mathbf{0}, c^2 m(\mathbf{x}) m(\mathbf{x}') + k(\mathbf{x}, \mathbf{x}')\right) \tag{1.47}$$

By moving the mean function into the covariance function, we get the same model, but we can integrate over the magnitude of the mean function at no additional cost. This is one advantage of moving as much structure as possible into the covariance function.

An unknown mean function can be moved into the covariance function If we wish to express our ignorance about the mean function, one way would be by putting a GP prior on it.

$$m \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}')), \quad f \sim \mathcal{GP}(m(\mathbf{x}), k_2(\mathbf{x}, \mathbf{x}')) \iff f \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')) \tag{1.48}$$

1.11 Why Not Learn the Mean Function Instead?

One might ask: besides integrating over the magnitude, what is the advantage of moving the mean function into the covariance function? After all, mean functions are certainly more interpretable than a posterior distribution over functions.

Instead of searching over a large class of covariance functions, which seems strange and unnatural, we might consider simply searching over a large class of structured mean functions, assuming a simple i.i.d. noise model. This is the approach taken by practically every other regression technique: neural networks, decision trees, boosting, etc. . If we could integrate over a wide class of possible mean functions, we would have a very powerful learning an inference method. The problem faced by all of these methods is the well-known problem *overfitting*. If we are forced to choose just a single function with which to make predictions, we must carefully control the flexibility of the model we learn, generally prefering "simple" functions, or to choose a function from a restricted set.

If, on the other hand, we are allowed to keep in mind many possible explanations of the data, *there is no need to penalize complexity*. [cite Occam's razor paper?] The power of putting structure into the covariance function is that doing so allows us to implictly integrate over many functions, maintaining a posterior distribution over infinitely many functions, instead of choosing just one. In fact, each of the functions being considered can

be infinitely complex, without causing any form of overfitting. For example, each of the samples shown in figure **??** varies randomly over the whole real line, never repeating, each one requiring an infinite amount of information to describe. Choosing the one function which best fits the data will almost certainly cause overfitting. However, if we integrate over many such functions, we will end up with a posterior putting mass on only those functions which are compatible with the data. In other words, the parts of the function that we can determine from the data will be predicted with certainty, but the parts which are still uncertain will give rise to a wide range of predictions.

To repeat: *there is no need to assume that the function being modeled is simple, or to prefer simple explanations* in order to avoid overfitting, if we integrate over many possible explanations rather than choosing just the one.

In Chapter **??**, we will compare a system which estimates a parametric covariance function against one which estimates a parametric mean function.

1.12 Signal versus Noise

In most derivations of Gaussian processes, the model is given as

$$y = f(x) + \epsilon, \quad \text{where} \quad \epsilon \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma^2_{\text{noise}}\right) \tag{1.49}$$

However, ϵ can equivalently be thought of as another Gaussian process, and so this model can be written as $y(x) \sim \mathcal{GP}\left(0, k + \delta\right)$. The lack of a hard distinction between the noise model and the signal model raises the larger question: Which part of a model represents signal, and which represents noise?

We believe that it depends on what you want to do with the model - there is no hard distinction between signal and noise in general.

For example: often, we don't care about the short-term variations in a function, and only in the long-term trend. However, in many other cases, we with to de-trend our data to see more clearly how much a particular part of the signal deviated from normal.

1.13 Learning Hyperparameters

1.14 Learning Structure

In some situations, we might not know *a priori* whether a particular structure or symmetry is present in the function we are trying to model. By building kernels with and without such structure, we can compute the marginal likelihoods of the corresponding GP models. The quantities represent the relative amount of evidence that the data provide for each of these possibilities, providing the assumptions of the model are correect.

1.15 Conclusion

We've seen that kernels are a flexible and powerful language for building models of different types of functions. However, for a given problem, it can difficult to specify an appropriate kernel, even after looking at the data. Rather than choosing one kernel *a priori*, one should at least try out a few different kernels. However, it might be difficult to enumerate all plausible kernels, and tedious to search over them.

Analogously, we usually don't expect to simply guess the best value of some parameter. Rather, we specify a search space and an objective, and ask the computer to the search this space for us. In the next chapter, we'll see how to perform such a search over the discrete space of kernel expressions.

References

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959. (page 11)

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png. (page 18)

D. Ginsbourger, O. Roustant, and N. Durrande. Invariances of random fields paths, with applications in Gaussian process regression. Technical Report arXiv:1308.1359 [math.ST], August 2013. (page 13)

David Ginsbourger, Xavier Bay, Olivier Roustant, Laurent Carraro, et al. Argumentwise invariant kernels for the approximation of invariant functions. In *Annales de la Faculté de Sciences de Toulouse*, number 3, 2012. (page 13)

T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 10)

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008. (page 13)

N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *The Journal of Machine Learning Research*, 6:1783–1816, 2005. (page 17)

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989. (page 10)

D. Lerner and D. Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984. (page 18)

James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pages 415–446, 1909. (page 11)

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning theory*, pages 154–168. Springer, 2006. (page 11)

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in AI*, pages 689–690. IEEE, 2011. (page 10)

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. (page 5)