Chapter 1

Expressing Structure with Kernels

This chapter shows how to use kernels to build models of functions with many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, and changepoints. We also show several ways to encode group invariants into kernels. Combining a few simple kernels through addition and multiplication will give us a rich, open-ended language of models.

The properties of kernels discussed in this chapter are mostly known in the literature. The original contribution of this chapter is to gather them into a coherent whole and to offer a tutorial showing the implications of different kernel choices, and some of the structures which can be obtained by combining them.

1.1 Definition

A kernel (also called a covariance function, kernel function, or covariance kernel), is a positive-definite function of two inputs $\mathbf{x}, \mathbf{x}'$. In this chapter, $\mathbf{x}$ and $\mathbf{x}'$ are usually vectors in a Euclidean space, but kernels can also be defined on graphs, images, discrete or categorical inputs, or even text.

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

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

Colloquially, kernels are often said to specify the similarity between two objects. This is slightly misleading in this context, since what is actually being specified is the similarity between two values of a *function* evaluated on each object. The kernel specifies which

functions are likely under the GP prior, which in turn determines the generalization properties of the model.

1.2 A few basic kernels

To begin understanding the types of structures expressible by GPs, we will start by briefly examining the priors on functions encoded by some commonly used kernels: the squared-exponential (SE), periodic (Per), and linear (Lin) kernels. These kernels are defined in figure 1.1.

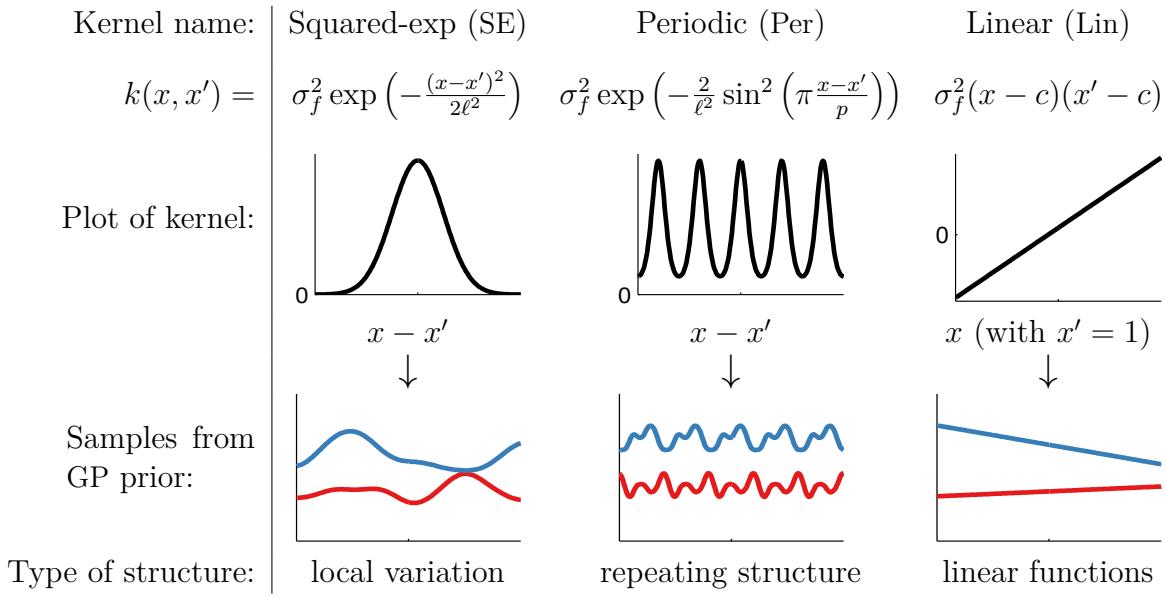


Figure 1.1 Examples of structures expressible by some basic kernels.

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. There exist many variants of "local" kernels similar to the SE kernel, each encoding slightly different assumptions about the smoothness of the function being modeled.

Kernel parameters Each kernel has a number of parameters which specify the precise shape of the covariance function. These are sometimes referred to as *hyper-parameters*, since they can be viewed as specifying a distribution over function parameters, instead of being parameters which specify a function directly. An example would be the lengthscale

parameter ℓ of the SE kernel, which specifies the width of the kernel and thereby the smoothness of the functions in the model.

Stationary and Non-stationary The SE and Per kernels are *stationary*, meaning that their value only depends on the difference $x - x'$. This implies that the probability of observing a particular dataset remains the same even if we move all the $\mathbf{x}$ values by the same amount. In contrast, the linear kernel (Lin) is non-stationary, meaning that the corresponding GP model will produce different predictions if the data were moved while the kernel parameters were kept fixed.

1.3 Combining kernels

What if the kind of structure we need is not expressed by any known kernel? For many types of structure, it is possible to build a "made to order" kernel with the desired properties. The next few sections of this chapter will explore ways in which kernels can be combined to create new ones with different properties. This will allow us to include as much high-level structure as necessary into our models.

1.3.1 Notation

Below, we will focus on two ways of combining kernels: addition and multiplication. We will often write these operations in shorthand, without arguments:

$$k_a + k_b = k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}') \tag{1.2}$$

$$k_a \times k_b = k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') \tag{1.3}$$

All of the basic kernels we considered in section 1.2 are one-dimensional, but kernels over multi-dimensional inputs can be constructed by adding and multiplying between kernels on different dimensions. The dimension on which a kernel operates is denoted by a subscripted integer. For example, SE_2 represents an SE kernel over the second dimension of vector $\mathbf{x}$. To remove clutter, we will usually refer to kernels without specifying their parameters.

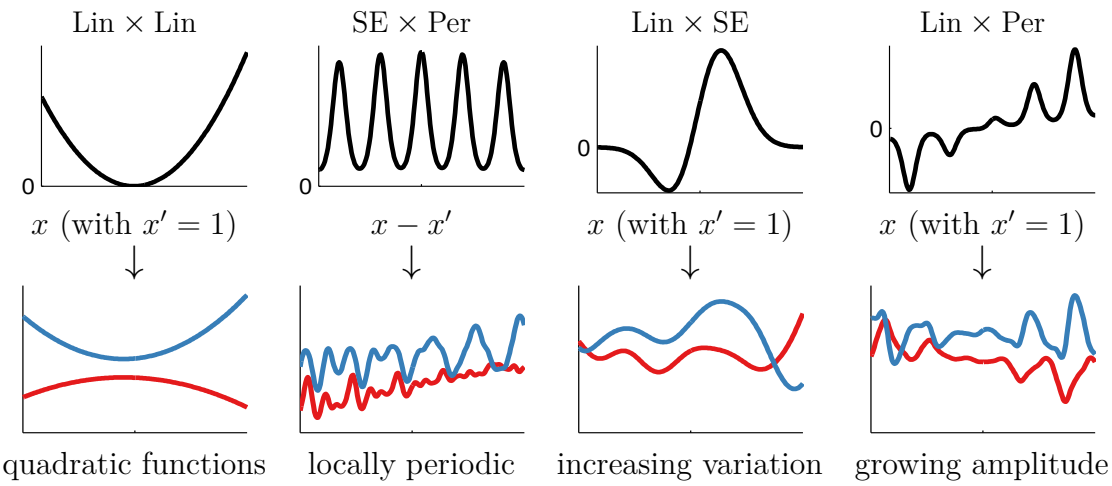


Figure 1.2 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 1.1.

1.3.2 Combining properties through multiplication

Multiplying two positive-definite kernels together always results in another positive-definite kernel. But what properties do these new kernels have? Figure 1.2 shows some kernels obtained by multiplying two basic kernels together.

Working with kernels, rather than the parametric form of the function itself, allows us to express high-level properties of functions that do not necessarily have a simple parametric form. Here, we discuss a few examples:

- **Polynomial Regression.** By multiplying together T linear kernels, we obtain a prior on polynomials of degree T. The first column of figure 1.2 shows a quadratic kernel.

- **Locally Periodic Functions.** In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to exactly periodic structure, whereas Per$\times$SE corresponds to locally periodic structure, as shown in the second column of figure 1.2.

- **Functions with Growing Amplitude.** Multiplying by a linear kernel means that the marginal standard deviation of the function being modeled grows linearly away from the location given by kernel parameter c. The third and fourth columns of figure 1.2 show two examples.

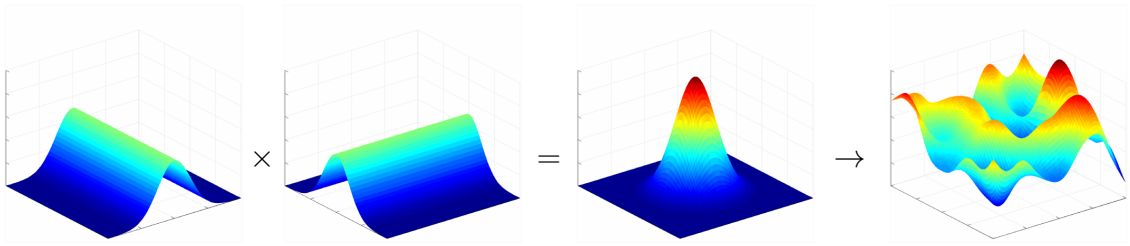
We can multiply any number of kernels together in this way to produce kernels combining several high-level properties. For example, the kernel SE×Lin×Per is a prior on functions which are locally periodic with linearly growing amplitude. We will a real datasets with this kind of structure in section 1.10.

1.3.3 Building multi-dimensional models

A flexible way to model functions having more than one input is to multiply together kernels defined on each individual input. For example, a product of SE kernels over different dimensions, each having a different lengthscale parameter, is called the SE-ARD kernel:

$$\text{SE-ARD}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} \sigma_d^2 \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) \tag{1.4}$$

Figure 1.3 illustrates the SE-ARD kernel in two dimensions.



$\text{SE}_1(x_1, x'_1)$ $\text{SE}_2(x_2, x'_2)$ $\text{SE}_1 \times \text{SE}_2$ $f(x_1, x_2)$ drawn from $\mathcal{GP}(0, \text{SE}_1 \times \text{SE}_2)$

Figure 1.3 A product of two one-dimensional kernels gives rise to a prior on functions which depend on both dimensions.

ARD stands for Automatic Relevance Determination, so named because estimating the lengthscale parameters $\ell_1, \ell_2, \ldots, \ell_D$, implicitly determines the "relevance" of each dimension. Input dimensions with relatively large lengthscales imply relatively little variation along those dimensions in the function being modeled.

SE-ARD kernels are the default kernel in most applications of GPs. This may be partly because they have relatively few parameters to estimate, and those parameters are relatively interpretable. In addition, there is a theoretical reason to use them: they are *universal* kernels (Micchelli et al., 2006), capable of learning any continuous function given enough data, under some conditions.

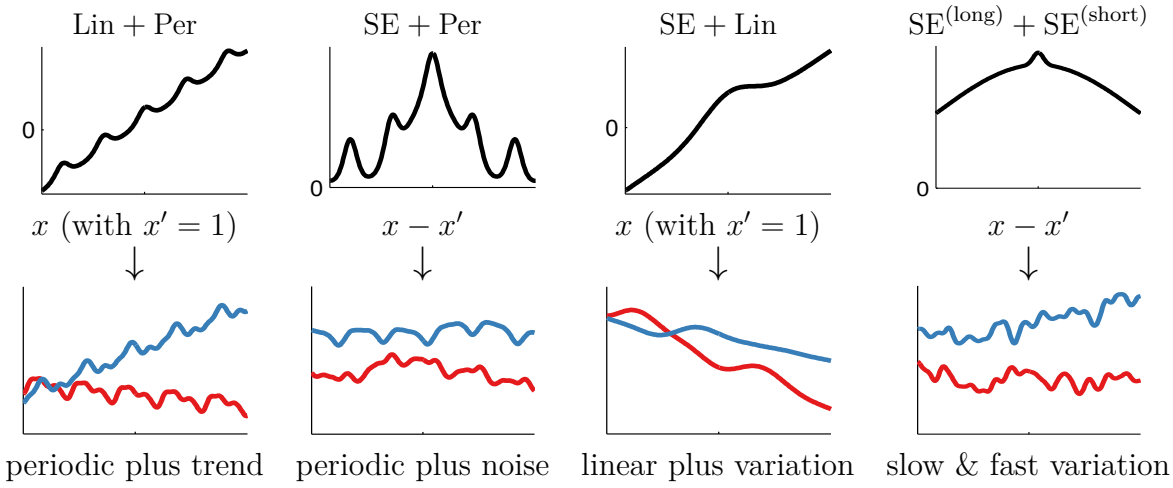


Figure 1.4 Examples of one-dimensional structures expressible by adding kernels. Rows have the same meaning as in figure 1.1. $\mathrm{SE}^{(\mathrm{long})}$ denotes a SE kernel whose lengthscale is long relative to that of $\mathrm{SE}^{(\mathrm{short})}$

However, this flexibility means that they can sometimes be relatively slow to learn, due to the *curse of dimensionality* (Bellman, 1956). In general, the more structure we account for, the less data we need - the *blessing of abstraction* (Goodman et al., 2011) counters the curse of dimensionality. Below, we will investigate ways to encode more structure into kernels.

1.4 Modeling sums of functions

An additive function is one which can be expressed as $f(\mathbf{x}) = f_a(\mathbf{x}) + f_b(\mathbf{x})$. Additivity is a useful modeling assumption in a wide variety of contexts, especially if it allows us to make strong assumptions about the individual components which make up the sum. Restricting the flexibility of component functions often aids in building interpretable models, and sometimes enables extrapolation in high dimensions.

It is easy to encode additivity into GP models. Suppose functions f_a, f_b are drawn independently from GP priors:

$$f_a \sim \mathcal{GP}(\mu_a, k_a) \tag{1.5}$$

$$f_b \sim \mathcal{GP}(\mu_b, k_b) \tag{1.6}$$

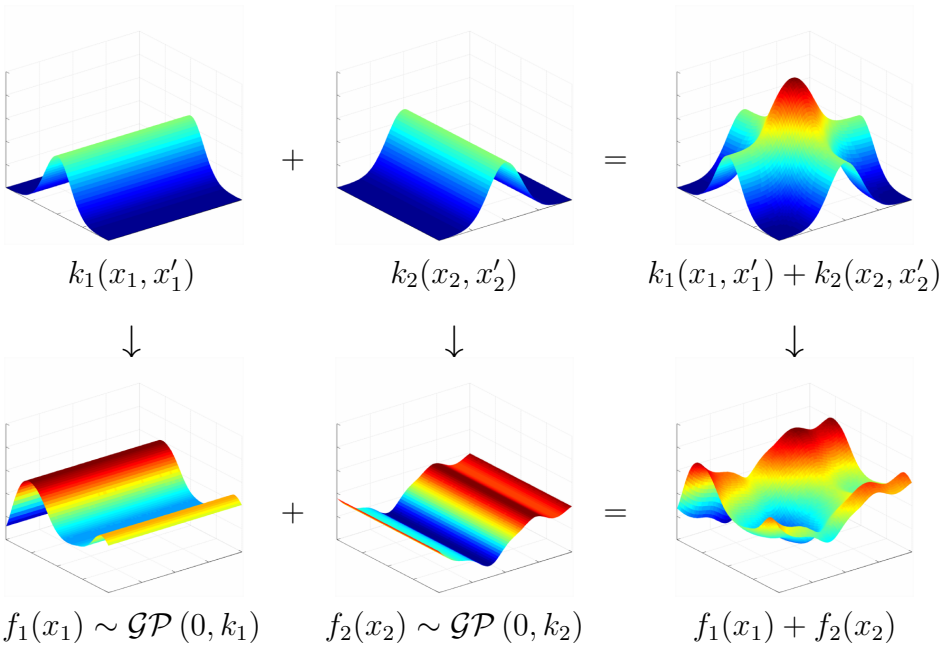


Figure 1.5 A sum of two orthogonal one-dimensional kernels. *Top row:* An additive kernel is any sum of kernels. *Bottom row:* A draw from an additive kernel corresponds to a sum of draws from independent GP priors, each having the corresponding kernel.

Then the sum of those functions is simply another GP:

$$f_a + f_b \sim \mathcal{GP}(\mu_a + \mu_b, k_a + k_b). \tag{1.7}$$

Kernels k_a and k_b can be of different types, allowing us to model the data as a sum of independent functions, each possibly representing a different type of structure. We can sum any number of components 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 of only a subset of input dimensions, then the implied prior over functions decomposes in the same way. For example,

$$f(x_1, x_2) \sim \mathcal{GP}(0, k_1(x_1, x_1') + k_2(x_2, x_2')) \tag{1.8}$$

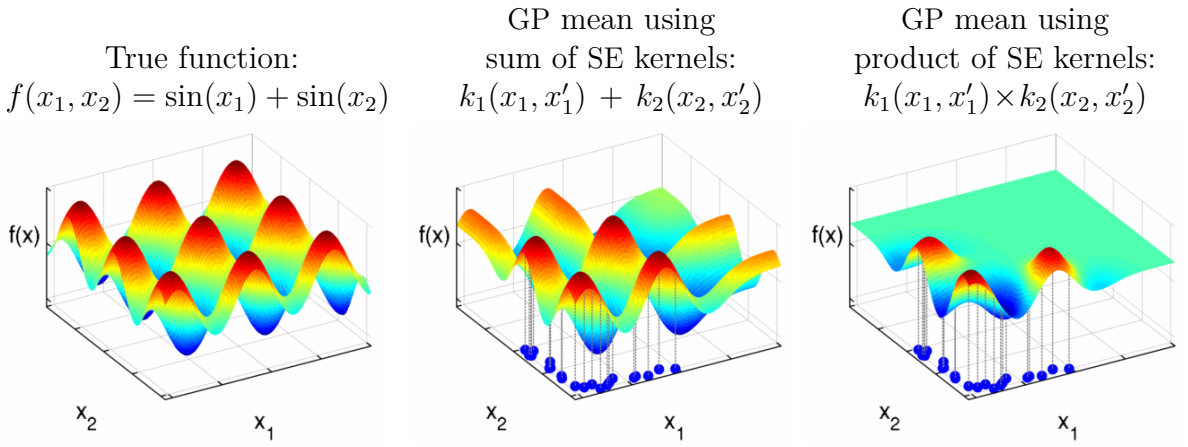


Figure 1.6 *Left:* A function with additive structure. *Center:* A GP with an additive kernel can extrapolate away from the training data. *Right:* A GP with a product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data. This causes the predictions to revert to the mean.

is equivalent to the model

$$f_1(x_1) \sim \mathcal{GP}(0, k_1(x_1, x_1')) \tag{1.9}$$

$$f_2(x_2) \sim \mathcal{GP}(0, k_2(x_2, x_2')) \tag{1.10}$$

$$f(x_1, x_2) = f_1(x_1) + f_2(x_2)\,. \tag{1.11}$$

Figure 1.5 illustrates a decomposition of this form. Note that the product of two kernels does not have an analogous interpretation as the product of two functions.

1.4.2 Extrapolation through additivity

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

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 multiple input dimensions. Section 1.10 considers this model class in more detail.

1.4.3 Example: An additive model of concrete strength

To illustrate how additive kernels give rise to interpretable models, we built an additive model of the strength of concrete as a function of the amount of seven different ingredients, plus the age of the concrete (Yeh, 1998).

Our simple additive model looks like

$$\begin{aligned} f(\mathbf{x}) = f_1(\text{cement}) + f_2(\text{slag}) + f_3(\text{fly ash}) + f_4(\text{water}) \\ + f_5(\text{plasticizer}) + f_6(\text{coarse}) + f_7(\text{fine}) + f_8(\text{age}) + \text{noise} \end{aligned} \tag{1.12}$$

where noise $\overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_n^2)$. The corresponding kernel has 9 additive components. After learning the kernel parameters by maximizing the marginal likelihood of the data, we can visualize the predictive distribution of each component of the model.

Figure 1.7 shows the marginal posterior distribution of each of the eight components in equation (1.12). The parameters 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, and another example of automatic relevance determination (ARD) (Neal, 1995).

The ability to learn kernel parameters in this way is much more difficult when using non-probabilistic methods such as Support Vector Machines (Cortes and Vapnik, 1995), for which cross-validation is often the best method to select kernel parameters.

1.4.4 Posterior variance of additive components

Here we derive the posterior variance and covariance of all of the additive components of a GP. These formulas allow one to make plots such as figure 1.7.

First, we will write down the joint prior over the sum of two functions drawn from GP priors. We 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).

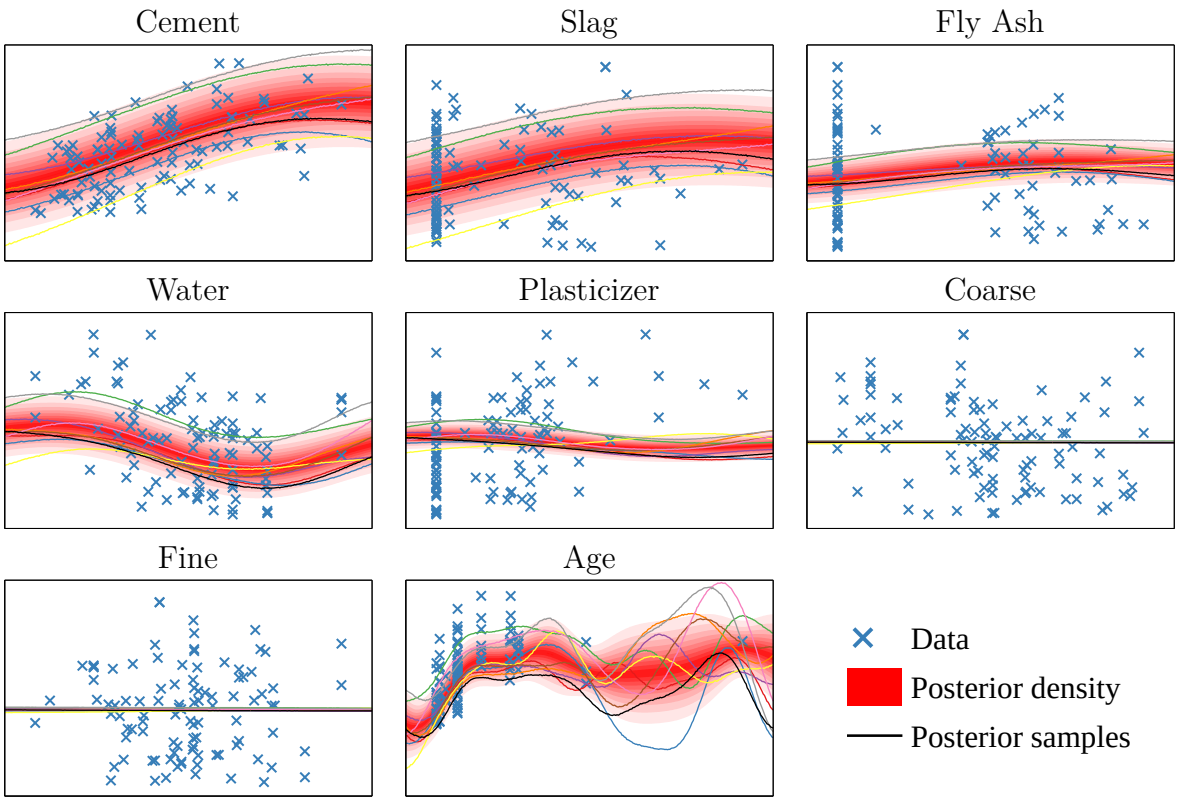


Figure 1.7 The predictive distribution of each component of a multi-dimensional additive model. Blue crosses indicate the original data projected on to each dimension, red indicates the marginal posterior density, and colored lines are samples from the marginal posterior of each component. The vertical axis is the same for all plots.

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

$$\begin{bmatrix} \boldsymbol{f}_1(\mathbf{X}) \\ \boldsymbol{f}_1(\mathbf{X}^\star) \\ \boldsymbol{f}_2(\mathbf{X}) \\ \boldsymbol{f}_2(\mathbf{X}^\star) \\ \boldsymbol{f}_1(\mathbf{X}) + \boldsymbol{f}_2(\mathbf{X}) \\ \boldsymbol{f}_1(\mathbf{X}^\star) + \boldsymbol{f}_2(\mathbf{X}^\star) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_1^\star \\ \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_2^\star \\ \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_1^\star + \boldsymbol{\mu}_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_1^\star & 0 & 0 & \mathbf{K}_1 & \mathbf{K}_1^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & 0 & 0 & \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} \\ 0 & 0 & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_2 & \mathbf{K}_2^\star \\ 0 & 0 & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} \\ \mathbf{K}_1 & \mathbf{K}_1^\star & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_1^\star + \mathbf{K}_2^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_1^\star + \mathbf{K}_2^\star & \mathbf{K}_1^{\star\star} + \mathbf{K}_2^{\star\star} \end{bmatrix} \right) \tag{1.13}$$

where we represent the Gram matrices, whose i, jth entry is given by $k(\mathbf{x}_i, \mathbf{x}_j)$ by

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

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

$$\mathbf{K}_1^{\star\star} = k_1(\mathbf{X}^\star, \mathbf{X}^\star) \tag{1.16}$$

The formula for Gaussian conditionals, **??**, can be used to give the conditional distribution of a GP-distributed function conditioned on its sum with another GP-distributed function:

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

These formulas express the model's posterior uncertainty about the different components of the signal, integrating over the possible configurations of the other components. To extend these formulas to a 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.

Posterior covariance of additive components

We can also compute the posterior covariance between any two components, conditioned on their sum:

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

If this quantity is negative, it means that there is ambiguity about which of the two components explains the data at that location. Figure 1.8 shows the posterior correlation between all non-zero components of the concrete model. Most of the correlation occurs within components, but there is also 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.

1.5 Changepoints

An example of how combining kernels can give rise to more structured priors is given by changepoint kernels. Changepoints kernels can be defined through addition and

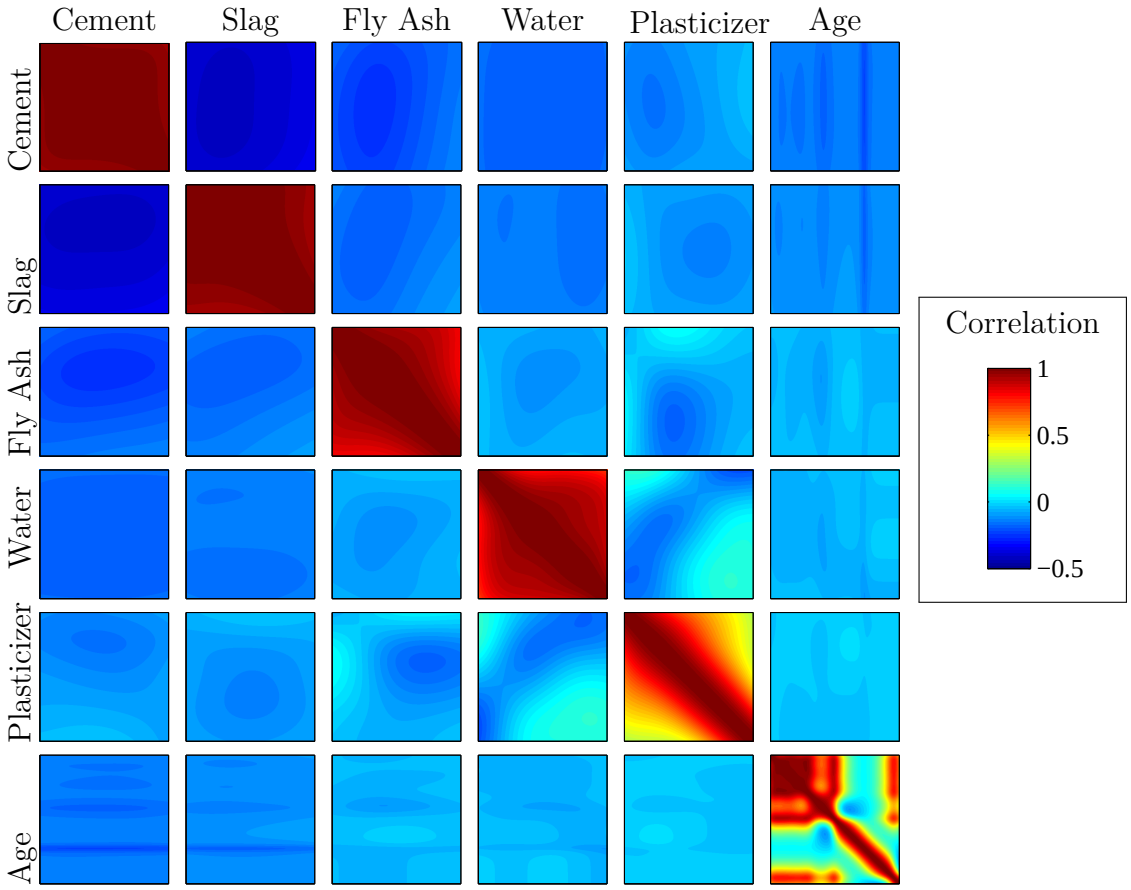


Figure 1.8 Posterior correlations between, and within, the additive components explaining the concrete dataset. Each plot shows the posterior correlations between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. Plots on the diagonal show posterior correlations within each component. Dimensions 'Coarse' and 'Fine' are not shown, because their variance is zero.

multiplication with sigmoidal functions such as $\sigma(x) = 1/1+\exp(-x)$:

$$\mathrm{CP}(k_1, k_2)(x, x') = \sigma(x)k_1(x, x')\sigma(x') + (1 - \sigma(x))k_2(x, x')(1 - \sigma(x')) \tag{1.19}$$

which can be written in shorthand as

$$\mathrm{CP}(k_1, k_2) = k_1 \times \boldsymbol{\sigma} \; + \; k_2 \times \bar{\boldsymbol{\sigma}} \tag{1.20}$$

where $\boldsymbol{\sigma} = \sigma(x)\sigma(x')$ and $\bar{\boldsymbol{\sigma}} = (1 - \sigma(x))(1 - \sigma(x'))$.

This compound kernel expresses a change from one kernel to another. The parameters of the sigmoid determine where, and how rapidly, this change occurs. Figure 1.9 shows some examples.

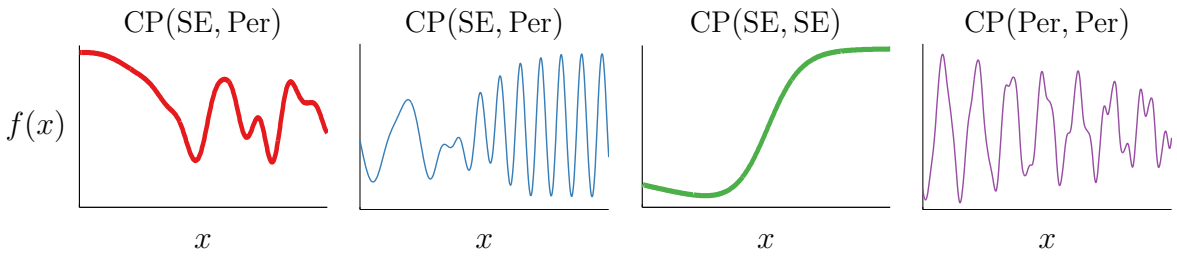


Figure 1.9 Draws from different priors on using changepoint kernels, constructed by adding and multiplying together base kernels with sigmoidal functions.

We can also build a model of functions whose structure changes only within some interval – a *change-window* – by replacing $\sigma(x)$ with a product of two sigmoids, one increasing and one decreasing.

1.5.1 Multiplication by a known function

More generally, we can model an unknown function that's been multiplied by any fixed, known function $a(x)$, by multiplying the kernel by $a(\mathbf{x})a(\mathbf{x}')$. Formally,

$$f(\mathbf{x}) = a(\mathbf{x})g(\mathbf{x}), \quad g \sim \mathcal{GP}(\,0, k(\mathbf{x}, \mathbf{x}')) \quad \Longleftrightarrow \quad f \sim \mathcal{GP}(\,0, a(\mathbf{x})k(\mathbf{x}, \mathbf{x}')a(\mathbf{x}'))\,. \tag{1.21}$$

1.6 Feature representation of kernels

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 $\mathbf{x}$ and at $\mathbf{x}'$:

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

For example, the squared-exponential kernel (SE) on the real line has a representation in terms of infinitely many radial-basis functions of the form $h_i(x) \propto \exp(-\frac{1}{2}\frac{(x-c_i)^2}{2\ell^2})$. More generally, any stationary kernel can be represented by a set of sines and cosines - a Fourier representation (Bochner, 1959). In general, any particular feature representation of a kernel is not unique (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*, explored in **??**.

1.6.1 Relation to linear regression

Surprisingly, GP regression is equivalent to Bayesian linear regression on the implicit features $\mathbf{h}(\mathbf{x})$ which give rise to the kernel:

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{h}(\mathbf{x}), \quad \mathbf{w} \sim \mathcal{N}(0, \mathbf{I}) \quad \iff \quad f \sim \mathcal{GP}\Big(0, \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x})\Big) \tag{1.23}$$

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

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. For example, given two kernels

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}') \tag{1.24}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{1.25}$$

their addition has the form:

$$k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') + \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') = \begin{bmatrix} \mathbf{a}(\mathbf{x}) \\ \mathbf{b}(\mathbf{x}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{a}(\mathbf{x}') \\ \mathbf{b}(\mathbf{x}') \end{bmatrix} \tag{1.26}$$

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:

$$k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') = \Big[\mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}')\Big] \times \Big[\mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}')\Big] \tag{1.27}$$

$$= \sum_i a_i(\mathbf{x})a_i(\mathbf{x}') \times \sum_j b_j(\mathbf{x})b_j(\mathbf{x}') \tag{1.28}$$

$$= \sum_{i,j} \Big[a_i(\mathbf{x})b_j(\mathbf{x})\Big]\Big[a_i(\mathbf{x}')b_j(\mathbf{x}')\Big] \tag{1.29}$$

In words, the features of $k_a \times k_b$ are made of up all pairs of the original two sets of features. For example, the features of the product of two one-dimensional SE kernels cover the plane with two-dimensional radial-basis functions of the form

$$h_{ij}(x_1, x_2) \propto \exp\left(-\frac{1}{2}\frac{(x_1 - c_i)^2}{2\ell_1^2}\right) \exp\left(-\frac{1}{2}\frac{(x_2 - c_j)^2}{2\ell_2^2}\right) \tag{1.30}$$

1.7 Expressing symmetries and invariances

When modeling functions, encoding known symmetries can improve predictive accuracy. This section looks at different ways to encode symmetries into a prior on functions. Many types of symmetry can be enforced through operations on the kernel.

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

1.7.1 Three recipes for invariant priors

Consider the scenario where we have a finite set of transformations of the input space $\{g_1, g_2, \ldots\}$ to which we wish our function to remain invariant:

$$f(\mathbf{x}) = f(g(\mathbf{x})) \quad \forall \mathbf{x} \in \mathcal{X}, \quad \forall g \in G \tag{1.31}$$

As an example, imagine we want to build a model of functions invariant to swapping their inputs: $f(x_1, x_2) = f(x_2, x_1)$. Being invariant to a set of operations is equivalent to being invariant to all compositions of those operations, the set of which form a group. (Armstrong et al., 1988, chapter 21). In our example, the elements of the group G_{swap}

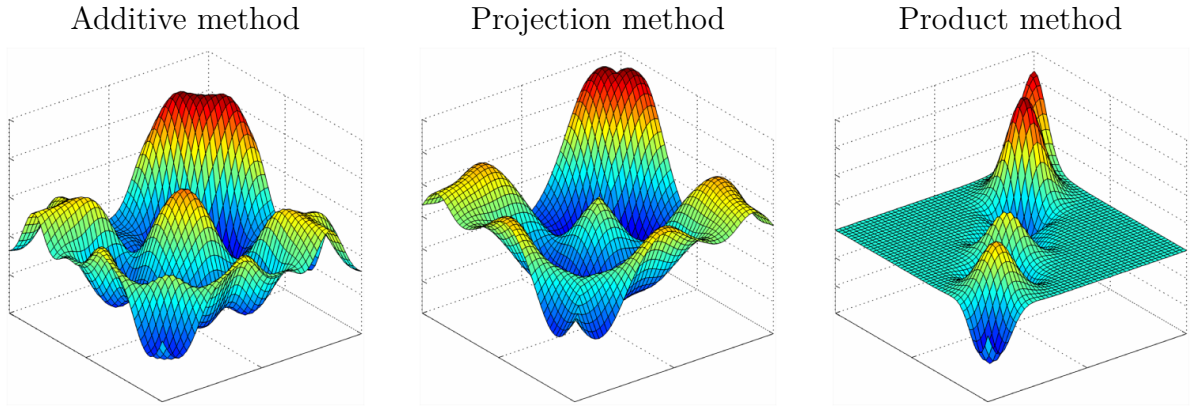


Figure 1.10 Three methods of introducing symmetry, illustrated through draws from the corresponding priors. All three methods introduce a different type of nonstationarity.

containing the operations the functions are invariant to has two elements:

$$g_1([x_1, x_2]) = [x_2, x_1] \qquad \text{(swap)} \tag{1.32}$$

$$g_2([x_1, x_2]) = [x_1, x_2] \qquad \text{(identity)} \tag{1.33}$$

How can we construct a prior on functions which respect these symmetries?

Ginsbourger et al. (2012) and Ginsbourger et al. (2013) showed that the only way to construct a GP prior on functions which respect a set of invariances is to construct a kernel which respects the same invariances with respect to each of its two inputs:

$$k(\mathbf{x}, \mathbf{x}') = k(g(\mathbf{x}), g(\mathbf{x}')), \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \forall g, g' \in G \tag{1.34}$$

Formally, given a finite group G whose elements are operations to which we wish our function to remain invariant, and $f \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$, then every f is invariant under G (up to a modification) if and only if $k(\cdot, \cdot)$ is argument-wise invariant under G.

It might not always be clear how to construct a kernel respecting such argument-wise invariances. Fortunately, there are a few simple ways to do this for any finite group:

1. **Sum over the orbit.** Ginsbourger et al. (2012) and Kondor (2008) suggest enforcing invariances through a double sum over the orbits of $\mathbf{x}$ and $\mathbf{x}'$ with respect

to G:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \sum_{g,\in G} \sum_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{1.35}$$

For the group G_{swap}, this operation results in the kernel:

$$\begin{aligned} k_{\text{switch}}(\mathbf{x}, \mathbf{x}') &= \sum_{g \in G_{\text{swap}}} \sum_{g' \in G_{\text{swap}}} k(g(\mathbf{x}), g'(\mathbf{x}')) && (1.36) \\ &= k(x_1, x_2, x_1', x_2') + k(x_1, x_2, x_2', x_1') \\ &\quad + k(x_2, x_1, x_1', x_2') + k(x_2, x_1, x_2', x_1') && (1.37) \end{aligned}$$

For stationary kernels, some pairs of elements in this sum will be identical, and can be ignored. Figure 1.10(left) shows a draw from a GP prior with a product of SE kernels symmetrized in this way. This construction has the property that the marginal variance is doubled near $x_1 = x_2$, which may or may not be desirable.

2. **Project onto a fundamental domain.** Ginsbourger et al. (2013) also explore the possibility of projecting each datapoint into a fundamental domain of the group, using a mapping A_G:

$$k_{\text{proj}}(\mathbf{x}, \mathbf{x}') = k(A_G(\mathbf{x}), A_G(\mathbf{x}')) \tag{1.38}$$

For the group G_{swap}, a fundamental domain is $\{x_1, x_2 : x_1 < x_2\}$, which can be mapped to using $A_{G_{\text{swap}}}(x_1, x_2) = \Big[\min(x_1, x_2), \max(x_1, x_2)\Big]$. Constructing a kernel using this method introduces a non-differentiable "seam" along $x_1 = x_2$, as shown in figure 1.10(center).

3. **Multiply over the orbit.** Ryan P. Adams (personal communication) suggested a construction enforcing invariances through products over the orbits:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \prod_{g \in G} \prod_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{1.39}$$

This method can sometimes produce GP priors with zero variance in some regions of the space, as in figure 1.10(right).

There are many possible ways to achieve a given symmetry, but we must be careful to do so without compromising other qualities of the model we are constructing. 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.

1.7.2 Example: Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

$$f(x) = f(x + \tau) \tag{1.40}$$

where τ is the period.

The most popular method for building a periodic kernel is due to MacKay (1998), who used the projection method in combination with an SE kernel. A fundamental domain of the symmetry group is a circle, so the kernel

$$\mathrm{Per}(x, x') = \mathrm{SE}\left(\sin(x), \sin(x')\right) \times \mathrm{SE}\left(\cos(x), \cos(x')\right) \tag{1.41}$$

achieves the invariance in equation (1.40). Simple algebra reduces this kernel to the form shown in figure 1.1.

1.7.3 Example: Symmetry about zero

Another simple example of an easily-enforceable symmetry is symmetry about zero:

$$f(x) = f(-x) \tag{1.42}$$

using the sum over orbits method, by the transform

$$k_{\mathrm{reflect}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{1.43}$$

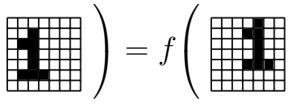
1.7.4 Example: Translation invariance in images

Many models of images are invariant to spatial translations (LeCun and Bengio, 1995). Similarly, many models of sounds are also invariant to translation through time.

This sort of translation invariance is completely distinct from the stationarity of kernels such as SE or Per. A stationary kernel implies that the prior is invariant to translations of the entire training and test set. In contrast, here we use translation invariance to refer to situations where the signal has been discretized, and each pixel (or the audio equivalent) corresponds to a different input dimension. We are interested

in creating priors on functions that are invariant to swapping pixels in a manner that corresponds to shifting the signal in some direction:

$$f\left(\quad\right) = f\left(\quad\right) \tag{1.44}$$



For example, in a one-dimensional image or audio signal, translation of an input vector by i pixels can be defined as

$$\text{shift}(\mathbf{x}, i) = \left[x_{\text{mod}(i+1,D)}, x_{\text{mod}(i+2,D)}, \ldots, x_{\text{mod}(i+D,D)}\right]^{\mathsf{T}} \tag{1.45}$$

As above, translation invariance in one dimension can be achieved by a double sum over the orbit

$$k_{\text{invariant}}\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{i=1}^{D}\sum_{j=1}^{D} k(\text{shift}(\mathbf{x}, i), \text{shift}(\mathbf{x}, j))\,, \tag{1.46}$$

defining the covariance between two signals to be the sum of all covariances between all translations of those two signals.

The extension to two dimensions, $\text{shift}(\mathbf{x}, i, j)$, is straightforward, but notationally cumbersome. Kondor (2008) built a more elaborate kernel between images, approximately invariant to both translation and rotation, using the projection method.

1.8 Generating topological manifolds

In this section we give a geometric illustration of the symmetries encoded by different compositions of kernels. The work presented in this section is based on a collaboration with David Reshef, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. The derivation of the Möbius kernel was an original contribution by myself.

Priors on functions obeying invariants can be used to create a prior on topological manifolds by using such functions to warp a simply-connected surface into a higher-dimensional space. For example, to build a prior on 2-dimensional manifolds embedded in 3-dimensional space, we simply need a prior on mappings from $\mathbb{R}^2$ to $\mathbb{R}^3$, which we can construct using three independent functions $[f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]$, each mapping from $\mathbb{R}^2$ to $\mathbb{R}$. GP priors on the mapping functions implictly give rise to a prior on warped surfaces. Symmetries in $[f_1, f_2, f_3]$ will connect different parts of the manifolds, giving rise to non-trivial topologies on the sampled surfaces. Figure 1.11 shows 2D meshes

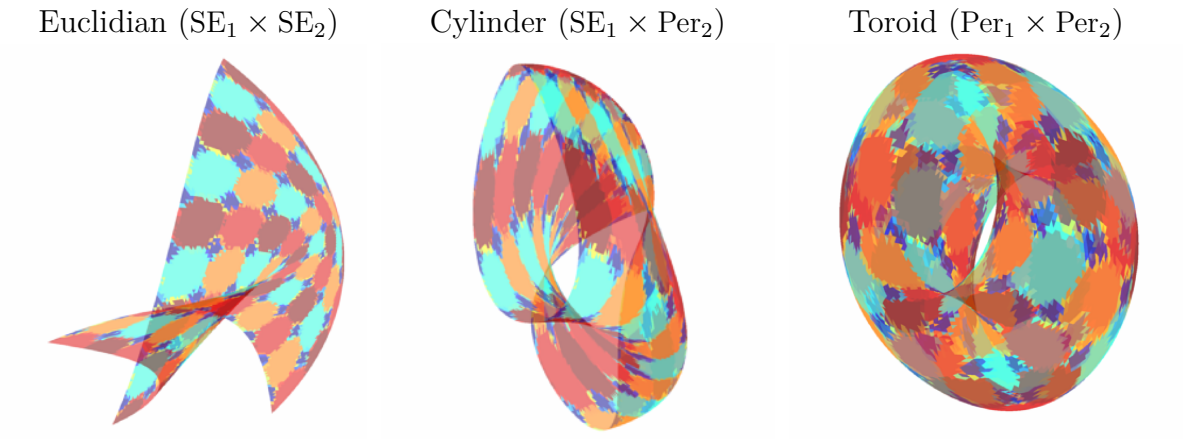


Figure 1.11 Generating 2D manifolds with different topologies. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey certain symmetries, the surfaces created have corresponding topologies, ignoring self-intersections.

warped into 3D by functions drawn from GP priors with different kernels, giving rise to a variety of different topologies. Higher-dimensional analogues of these shapes can be constructed by increasing the latent dimension and including corresponding terms in the kernel. For example, an N-dimensional space with kernel $\mathrm{Per}_1 \times \mathrm{Per}_2 \times \ldots \times \mathrm{Per}_N$ will give rise to a prior on N-dimensional toruses.

This construction is similar in spirit to the GP latent variable model (GP-LVM) of Lawrence (2005), which learns a latent embedding of the data into a low-dimensional space, using a GP prior on the mapping from the latent space to the observed space.

1.8.1 Möbius strips

A space having the topology of a Möbius strip can be constructed by enforcing the following invariance to the following operations (Reid and Szendrői, 2005, chapter 7):

$$g_{p_1}([x_1, x_2]) = [x_1 + \tau, x_2] \qquad \text{(periodic in } x_1\text{)} \tag{1.47}$$

$$g_{p_2}([x_1, x_2]) = [x_1, x_2 + \tau] \qquad \text{(periodic in } x_2\text{)} \tag{1.48}$$

$$g_s([x_1, x_2]) = [x_2, x_1] \qquad \text{(symmetric about } x_1 = x_2\text{)} \tag{1.49}$$

Section 1.7 already showed how to build GP priors invariant to each of these types of transformations. We'll call a kernel which enforces all of these symmetries a *Möbius*

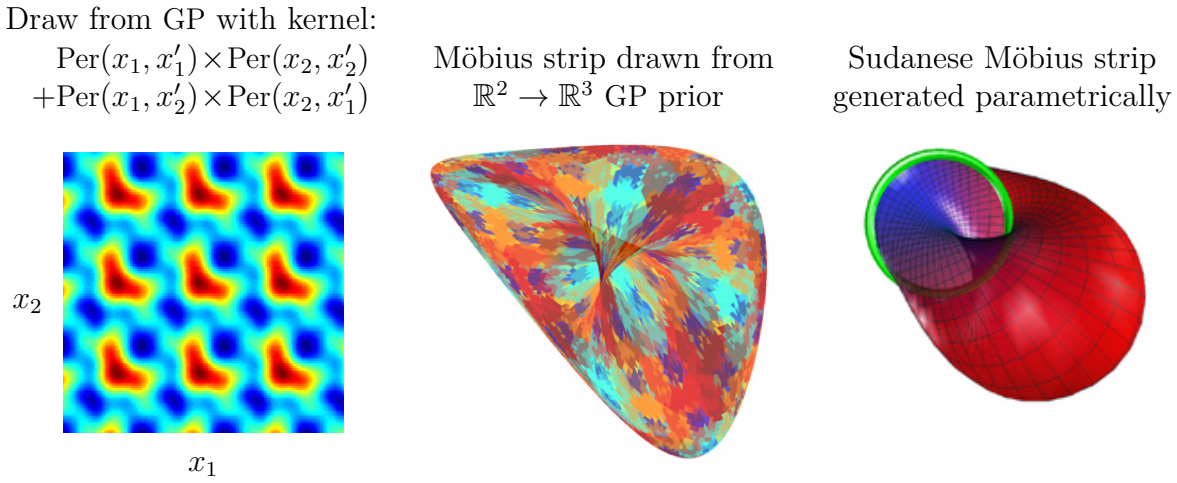


Figure 1.12 Generating Möbius strips. *Left:* A function drawn from a GP prior obeying the symmetries given by equations (1.47) to (1.49). *Center:* Simply-connected surfaces mapped from $\mathbb{R}^2$ to $\mathbb{R}^3$ by functions obeying those symmetries have a topology corresponding to a Möbius strip. Surfaces generated this way do not have the familiar shape of a flat surface connected to itself with a half-twist. Instead, 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 Wikimedia Commons (2005).

kernel. An example of such a kernel is:

$$k(x_1, x_2, x_1', x_2') = \mathrm{Per}(x_1, x_1') \times \mathrm{Per}(x_2, x_2') \; + \; \mathrm{Per}(x_1, x_2') \times \mathrm{Per}(x_2, x_1') \tag{1.50}$$

Moving along the diagonal $x_1 = x_2$ of a function drawn from the corresponding GP prior is equivalent to moving along the edge of a notional Möbius strip which has had that function mapped on to its surface. Figure 1.12(a) shows an example of a function drawn from such a prior. Figure 1.12(b) shows an example of a 2D mesh mapped to 3D by functions drawn from such a prior. This surface doesn't resemble the typical representation of a Möbius strip, but instead resembles an embedding known as the Sudanese Möbius strip (Lerner and Asimov, 1984), shown in figure 1.12(c).

1.9 Kernels on categorical variables

Categorical variables are variables which can take values only from a discrete, unordered set, such as $\{\texttt{blue}, \texttt{green}, \texttt{red}\}$. A flexible way to construct a kernel over categorical variables is to represent that variable by a set of binary variables, using a one-of-k

encoding. For example, if $\mathbf{x}$ can take one of four values, $x \in \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}$, then a one-of-k encoding of x will correspond to four binary inputs, and one-of-k$(\mathtt{C}) = [0, 0, 1, 0]$. Given a one-of-k encoding, we can place any multi-dimensional kernel on that space, such as the SE-ARD:

$$k_{\text{categorical}}(x, x') = \text{SE-ARD}(\text{one-of-k}(x), \text{one-of-k}(x')) \tag{1.51}$$

Short lengthscales on any particular dimension of the SE-ARD kernel indicate that the function value corresponding to that category is uncorrelated with the others.

A more flexible parameterization suggested by Kevin Swersky (personal communication) allows complete flexibility about which pairs of categories are similar to one another, replacing the SE-ARD kernel with a fully-parameterized kernel, SE-full:

$$\text{SE-full}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}'\right) \tag{1.52}$$

where $\mathbf{L}$ is a symmetric matrix, individually parameterizing the covariance between each pair of function values of categories.

1.10 Building a kernel in practice

This chapter outlined ways to choose the parametric form of a kernel in order to express different sorts of structure. Once the parametric form has been chosen, one still needs to choose, or integrating over, the kernel parameters. If the kernel we construct has relatively few parameters, these parameters can be estimated by maximum marginal likelihood, using gradient-based optimizers. The kernel parameters estimated in the examples above were optimized using the GPML toolbox, available at http://www.gaussianprocess.org/gpml/code.

A systematic search over kernel parameters is necessary when appropriate parameters aren't known. Likewise, sometimes appropriate kernel structures are hard to guess. The next chapter will show how to perform an automatic search not just over the kernel parameters, but also over the open-ended space of kernel expressions.

Source code

Source code to produce all figures and examples in this chapter is available at http://www.github.com/duvenaud/phd-thesis.

References

Mark A. Armstrong, Gérard Iooss, and Daniel D. Joseph. *Groups and symmetry*. Springer, 1988. (page 15)

Richard Bellman. Dynamic programming and Lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956. (page 6)

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959. (page 14)

Corinna Cortes and Vladimir N. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. (page 9)

David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro. Argumentwise invariant kernels for the approximation of invariant functions. In *Annales de la Faculté de Sciences de Toulouse*, 2012. (page 16)

David Ginsbourger, Olivier Roustant, and Nicolas Durrande. Invariances of random fields paths, with applications in Gaussian process regression. *arXiv preprint arXiv:1308.1359 [math.ST]*, August 2013. (pages 16 and 17)

Noah D. Goodman, Tomer D. Ullman, and Joshua B. Tenenbaum. Learning a theory of causality. *Psychological review*, 118(1):110, 2011. (page 6)

Trevor J. Hastie and Robert J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 8)

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008. (pages 16 and 19)

Neil D. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005. (page 20)

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995. (page 18)

Doug Lerner and Dan Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984. (page 21)

David J.C. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998. (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 14)

Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7:2651–2667, 2006. (page 5)

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning theory*, pages 154–168. Springer, 2006. (page 14)

Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (page 9)

Miles A. Reid and Balázs Szendrői. *Geometry and topology*. Cambridge University Press, 2005. (page 20)

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL `http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png`. (page 21)

I-Cheng Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. (page 9)