Chapter 1

Expressing Structure with Kernels

In this chapter, we'll show how to use kernels (also called *covariance functions*) to express many different kinds of priors on functions. By combining a few simple base kernels, we'll be able to create many different kinds of structured priors.

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

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

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

1.1 Base 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, or exhaustive about these kernels in particular, except that they represent a 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.

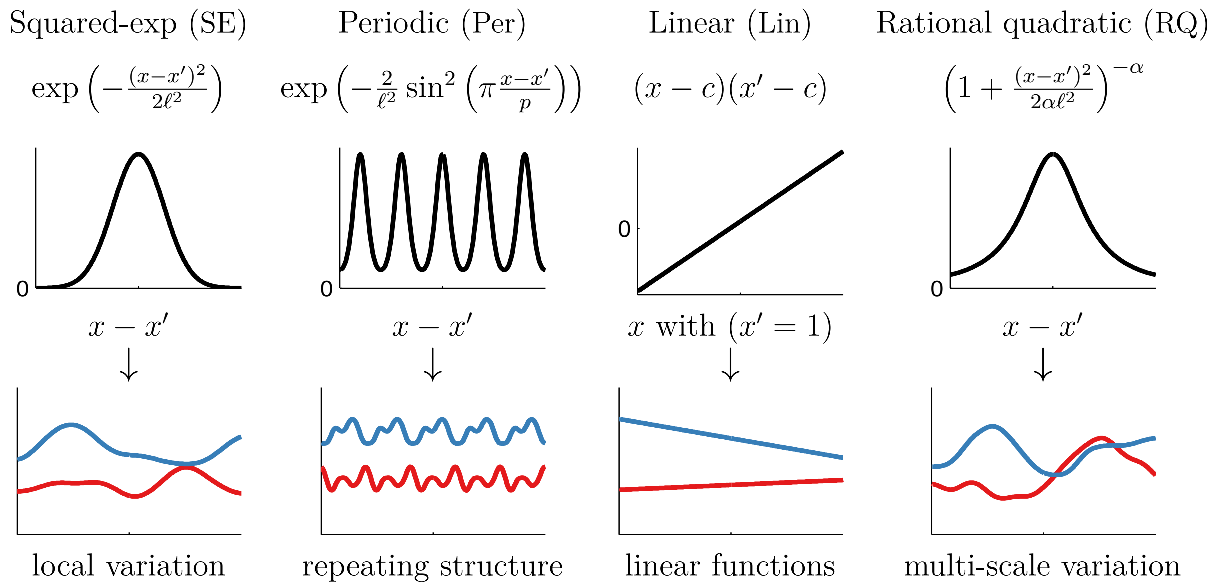


Fig. 1.1 Examples of structures expressible by base kernels. Left and third columns: base kernels $k(\cdot, 0)$. Second and fourth columns: draws from a GP with each repective kernel. The x-axis has the same range on all plots.

1.1.1 Definition of a Kernel

A covariance function $k(x, x') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ can be used to define a measure of similarity between two points x, x' in some space $\mathcal{X}$.

Feature representation

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

$$k(x, x') = \mathbf{h}(x)^{\mathsf{T}}\mathbf{h}(x') \tag{1.2}$$

GP regression is actually equivalent to performing Bayesian linear regression on the features $\mathbf{h}(x)$. The links between Gaussian processes, linear regression, and neural networks is explored further in chapter **??**.

1.2 Combining Kernels

There are several ways to combine existing kernels to create new ones with different properties. For an overview, see (Rasmussen and Williams, 2006, Chapter 4). In this chapter, we'll focus on two methods for combining kernels: addition and multiplication.

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

$$k_1 \times k_2 = k_1(x, x') \times k_2(x, x') \tag{1.4}$$

Combining kernels using these operations can yield kernels encoding for richer structures than are encoded in the original kernels.

This allows one to create richly structured and interpretable kernels from well understood base components.

All of the base kernels we use 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.

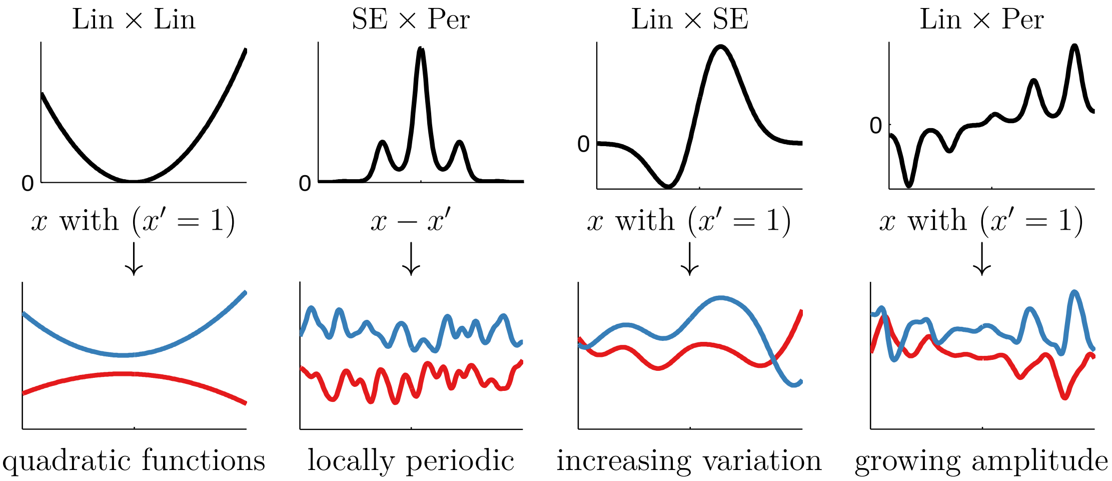


Fig. 1.2 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 1.1.

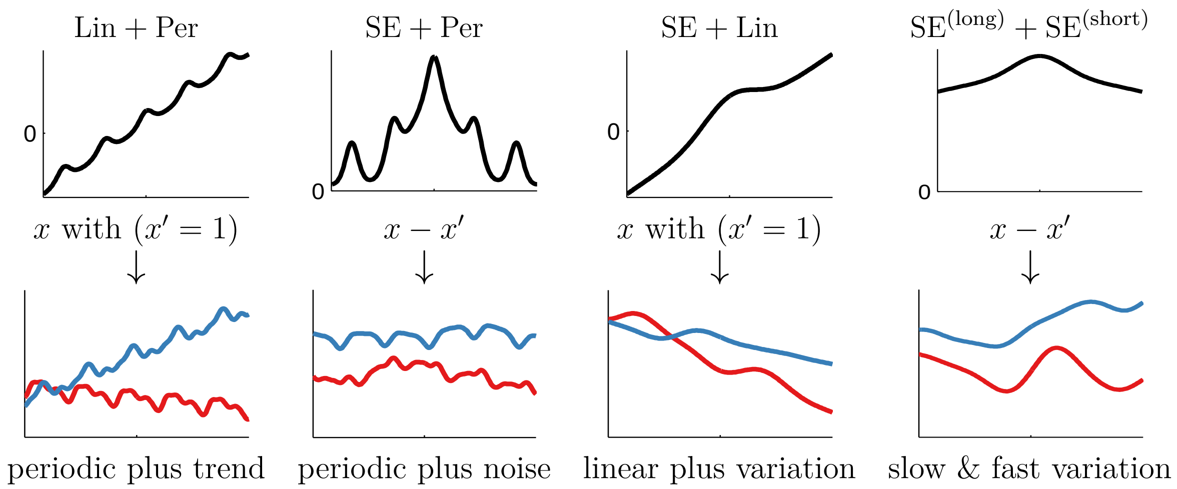


Fig. 1.3 Examples of one-dimensional structures expressible by adding kernels. Plots have same meaning as in figure 1.1.

1.3 Structure through Additivity

By summing kernels, we can model the data as a sum of independent functions, possibly representing different structures. Suppose functions f_1, f_2 are drawn independently, $f_1 \sim \mathcal{GP}(\mu_1, k_1)$, $f_2 \sim \mathcal{GP}(\mu_2, k_2)$. Then

$$f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2). \tag{1.5}$$

In time series models, sums of kernels can express superposition of different processes, possibly operating at different scales. In multiple dimensions, summing kernels gives additive structure over different dimensions, similar to generalized additive models (Hastie and Tibshirani, 1990). These two kinds of structure are demonstrated in rows 2 and 4 of figure 1.4, respectively.

A theme throughout this thesis is exploring the idea that a lot of the expressivity of GP models comes from the fact that these models can be combined and decomposed additively.

1.3.1 Marginal Variance and Covariance of Components

In this section, we derive the posterior marginal variance and covariance of the additive components of a GP. These formulas let us plot the marginal variance of each component separately. These formulas can also be used to examine the posterior covariance between pairs of components.

Let’s examine the joint prior over the sum of two functions. We’ll distinguish between $f(\mathbf{x})$ (the function values at the training locations) and $f(\mathbf{x}^\star)$ (the function values at some other, query locations) so that it’s clear which matrices to use to extrapolate.

If $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are *a priori* independent, and $\boldsymbol{f}_1 \sim \mathrm{GP}(\mu_1, k_1)$ and $\boldsymbol{f}_2 \sim \mathrm{GP}(\mu_2, k_2)$, then

$$\begin{bmatrix} \boldsymbol{f}_1(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) \\ \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_2(\mathbf{x}^\star) \\ \boldsymbol{f}_1(\mathbf{x}) + \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) + \boldsymbol{f}_2(\mathbf{x}^\star) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_1^\star \\ \mu_2 \\ \mu_2^\star \\ \mu_1 + \mu_2 \\ \mu_1^\star + \mu_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_1^\star & 0 & 0 & \mathbf{K}_1 & \mathbf{K}_1^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & 0 & 0 & \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} \\ 0 & 0 & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_2 & \mathbf{K}_2^\star \\ 0 & 0 & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} \\ \mathbf{K}_1 & \mathbf{K}_1^\star & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_1^\star + \mathbf{K}_2^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_1^\star + \mathbf{K}_2^\star & \mathbf{K}_1^{\star\star} + \mathbf{K}_2^{\star\star} \end{bmatrix} \right) \tag{1.6}$$

where $\mathbf{k}_1 = k_1(\mathbf{X}, \mathbf{X})$ and $\mathbf{k}_1^\star = k_1(\mathbf{X}^\star, \mathbf{X})$.

By the formula for Gaussian conditionals:

$$\mathbf{x}_A | \mathbf{x}_B \sim \mathcal{N}(\mu_A + \boldsymbol{\Sigma}_{AB} \boldsymbol{\Sigma}_{BB}^{-1} (\mathbf{x}_B - \mu_B), \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB} \boldsymbol{\Sigma}_{BB}^{-1} \boldsymbol{\Sigma}_{BA}), \tag{1.7}$$

we get that the conditional variance of a Gaussian conditioned on its sum with another Gaussian is given by

$$\boldsymbol{f}_1^\star | \boldsymbol{f} \sim \mathcal{N}\Big(\mu_1^\star + \mathbf{k}_1^{\star\mathsf{T}} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} (\boldsymbol{f} - \mu_1 - \mu_2), \mathbf{k}_1^{\star\star} - \mathbf{k}_1^{\star\mathsf{T}} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} \mathbf{k}_1^\star\Big) \tag{1.8}$$

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

$$\mathrm{cov}\left[\boldsymbol{f}_1^\star, \boldsymbol{f}_2^\star | \boldsymbol{f}\right] = -\mathbf{k}_1^{\star\mathsf{T}} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} \mathbf{k}_2^\star \tag{1.9}$$

1.9 is not always negative.

These formulae express the posterior model uncertainty about different components of the signal, integrating over the possible configurations of the other components.

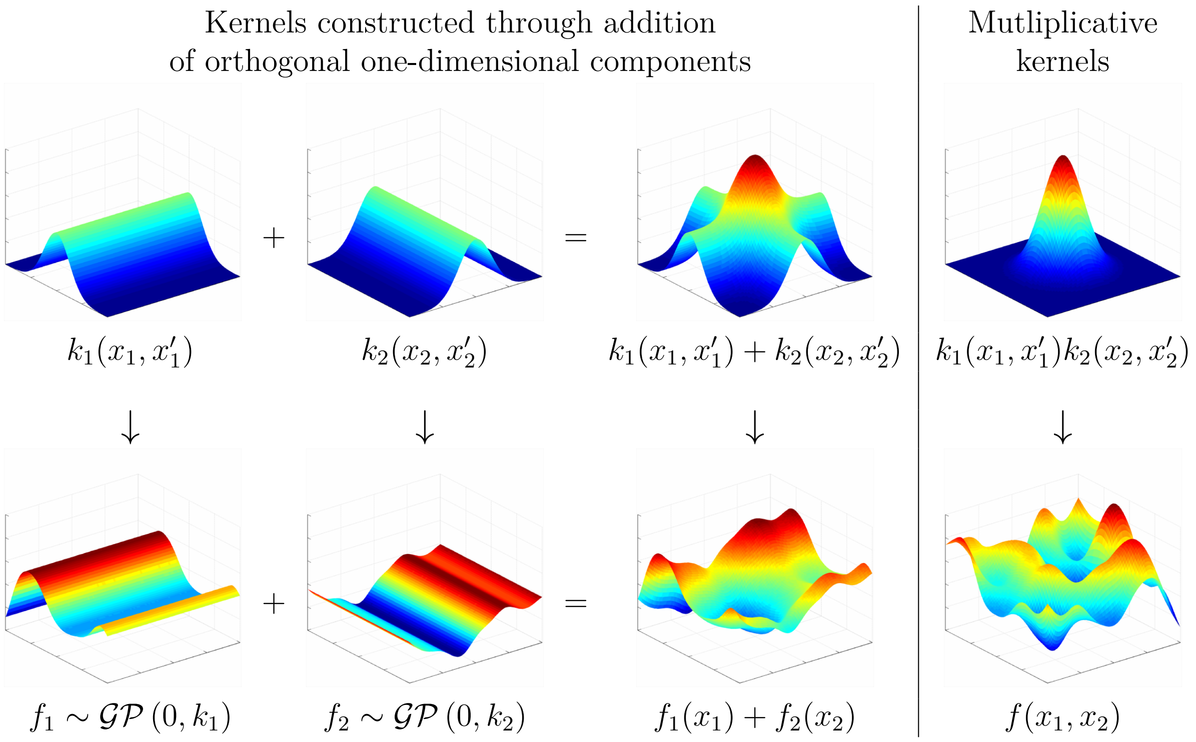


Fig. 1.4 Top left: An additive kernel is simply a sum of kernels. Bottom left: A draw from an additive kernel corresponds to a sum of draws from GPs with the corresponding kernels. Top right: a product kernel is a product of kernels. Bottom right: A draw from a product kernel does not correspond to a product of draws from the corresponding kernels.

1.3.2 Additivity Across Multiple Dimensions

Figure 1.4 compares, for two-dimensional functions, a first-order additive kernel with a second-order kernel.

Long-range extrapolation through additivity

Because additive kernels can discover non-local structure in data, they are exceptionally well-suited to problems where local interpolation fails. Figure 1.5 shows the very different extrapolations made by different GP models. We constructed a training set by evaluating a sum of two axis-aligned sine functions in a a small, L-shaped area.

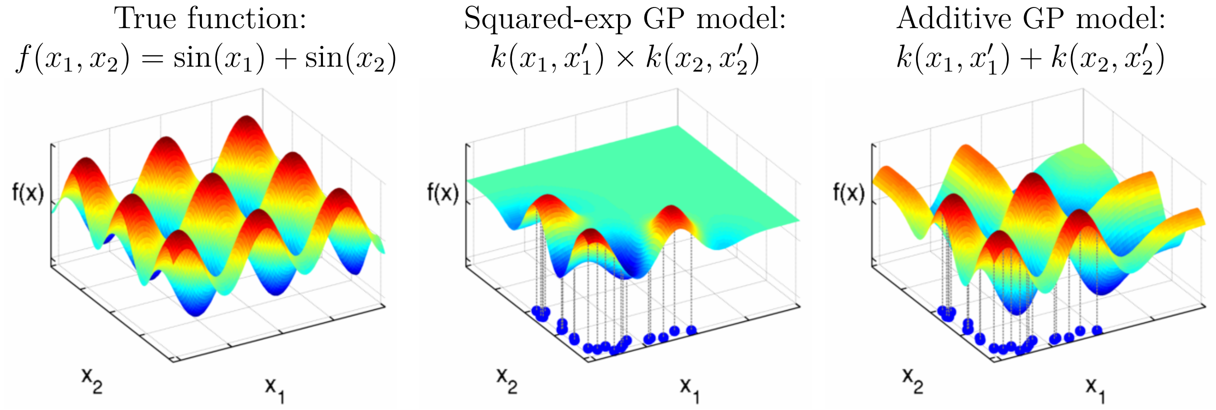


Fig. 1.5 Inference in functions with additive structure. When the function we're modeling has additive structure, we can epxloit this fact to extrapolate far from the training data.

1.3.3 Interpreting Additive Models in High Dimensions

One advantage of additive models is their interpretability. Plate (1999) demonstrated that by allowing high-order interactions as well as low-order interactions, one can trade off interpretability with predictive accuracy. In the case where the hyperparameters indicate that most of the variance in a function can be explained by low-order interactions, it is informative to plot the corresponding low-order functions, as in Figure 1.6.

1.4 Structure through Multiplication

Multiplying kernels allows us to account for interactions between different input dimensions or different notions of similarity. For instance, in multidimensional data, the

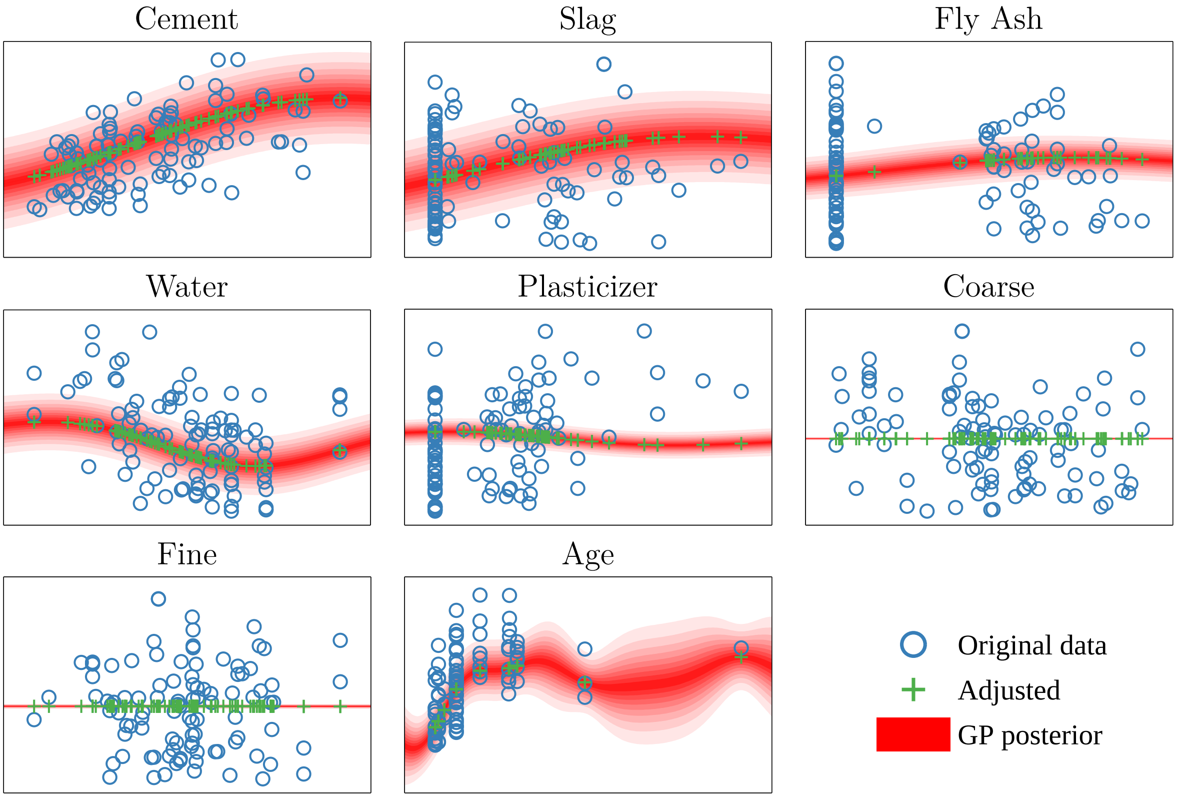


Fig. 1.6 By considering only one-dimensional terms of the additive kernel, we recover a form of Generalized Additive Model, and can plot the corresponding 1-dimensional functions. Blue points indicate the original data, green points are data after the contribution from all other terms has been subtracted. The vertical axis is the same for all plots.

multiplicative kernel $\text{SE}_1 \times \text{SE}_3$ 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 1.4.

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

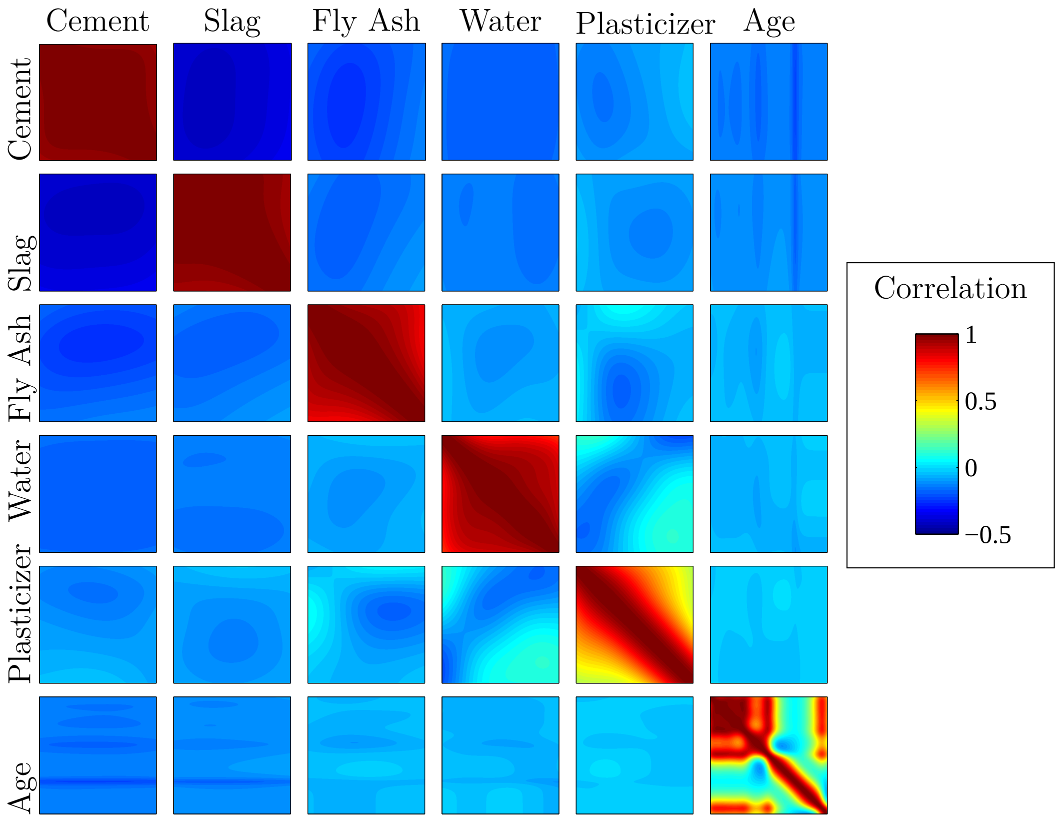


Fig. 1.7 Visualizing posterior correlations between the components explaining the concrete dataset. Each plot shows the posterior correlation between two components. Within each plot, color indicates the amount of correlation between the function value of the two components. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. We can see that there is negative correlation between the "Cement" and "Slag" variables. This reflects a degeneracy in the model. The 'Coarse' and 'Fine' dimensions are not shown here because their variance was zero.

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.

1.4.1 Multiplying by a constant function

If we wish to model a function that's been multiplied by some fixed 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.

1.5 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

$$k_1(x, x') = \mathbf{h}_1(x)^\mathsf{T} \mathbf{h}_1(x') \tag{1.10}$$

$$k_2(x, x') = \mathbf{h}_2(x)^\mathsf{T} \mathbf{h}_2(x') \tag{1.11}$$

then

$$k_1(x, x') + k_2(x, x) = \mathbf{h}_1(x)^\mathsf{T} \mathbf{h}_2(x') + \mathbf{h}_1(x)^\mathsf{T} \mathbf{h}_2(x') \tag{1.12}$$

$$= \begin{bmatrix} \mathbf{h}_1(x) \\ \mathbf{h}_2(x) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{h}_1(x') \\ \mathbf{h}_2(x') \end{bmatrix} \tag{1.13}$$

In other words, the features of $k_1 + k_2$ are the concatenation of the features of each kernel.

We can examine kernel multiplication in a similar way:

$$k_1(x, x') \times k_2(x, x') = \left[\mathbf{h}_1(x)^\mathsf{T} \mathbf{h}_1(x')\right] \times \left[\mathbf{h}_2(x)^\mathsf{T} \mathbf{h}_2(x')\right] \tag{1.14}$$

$$= \sum_i h_i(x) h_i(x') \times \sum_j h_j(x) h_j(x') \tag{1.15}$$

$$= \sum_i \sum_j h_i(x) h_i(x') h_j(x) h_j(x') \tag{1.16}$$

$$= \sum_{i,j} \left[h_i(x) h_j(x)\right] \left[h_i(x') h_j(x')\right] \tag{1.17}$$

$$= \operatorname{vec}\left(\mathbf{h}_1(x) \otimes \mathbf{h}_2(x')\right)^\mathsf{T} \operatorname{vec}\left(\mathbf{h}_1(x) \otimes \mathbf{h}_2(x')\right) \tag{1.18}$$

In other words, the features of $k_1 \times k_2$ are the cartesian product (all possible combinations) of the original sets of features. For example, one set of features $\mathbf{h}(x)$ 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.

1.5.1 Signal versus noise

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

$$y = f(x) + \epsilon, \quad \text{where} \quad \epsilon \overset{\text{iid}}{\sim} \mathcal{N}\Big(0, \sigma^2_{\text{noise}}\Big) \tag{1.19}$$

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?

Our answer is: *it depends on what you want to do with the model.* 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.

1.6 Expressing Symmetries

When modeling functions, encoding known symmetries greatly aids learning and prediction. We demonstrate that in nonparametric regression, many types of symmetry can be enforced through operations on the covariance function. These symmetries can be composed to produce nonparametric priors on functions whose domains have interesting topological structure such as spheres, torii, and Möbius strips.

Ginsbourger et al. (2012) and Ginsbourger et al. (2013) characterized GP priors on functions invariant to 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:

$$\forall \mathbf{x}, \forall g \in G, \quad k(g(\mathbf{x}), x') = k(\mathbf{x}, \mathbf{x}') \tag{1.20}$$

In this section, we give recipes for expressing several classes of symmetries. Later, we will show how these can be combined to produce more interesting structures.

1.6.1 Periodicity

Given D dimensions, we can enforce rotational symmetry on any subset of the dimensions:

$$f(x) = f(x_i + \tau_i) \tag{1.21}$$

by the applying a kernel between pairs transformed coordinates $\sin(x), \cos(x)$:

$$k_{\text{periodic}}(x, x') = k(\sin(x), \cos(x), \sin(x'), \cos(x')) \tag{1.22}$$

We can also apply rotational symmetry repeatedy to a single dimension.

1.6.2 Reflective Symmetry along an axis

we can enforce the symmetry

$$f(x) = f(-x) \tag{1.23}$$

by the kernel transform

$$k_{\text{symm arg1}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{1.24}$$

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 $k(-x, -x')$.

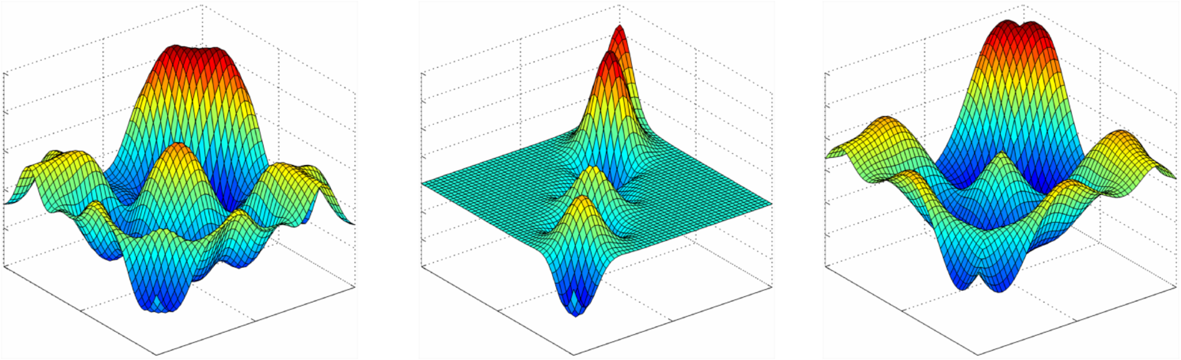
1.6.3 Reflective Symmetry along a diagonal

We can enforce symmetry between any two dimensions:

$$f(x, y) = f(y, x) \tag{1.25}$$

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

$$k_{\text{reflect add}}(x, y, x', y') = k(x, y, x', y') + k(x, y, y', x') + k(y, x, x', y') + k(y, x, y', x') \tag{1.26}$$



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.

or by

$$k_{\text{reflect min}}(x, y, x', y') = k(\,\min(x, y), \max(x, y), \min(x', y'), \max(x', y')) \qquad (1.27)$$

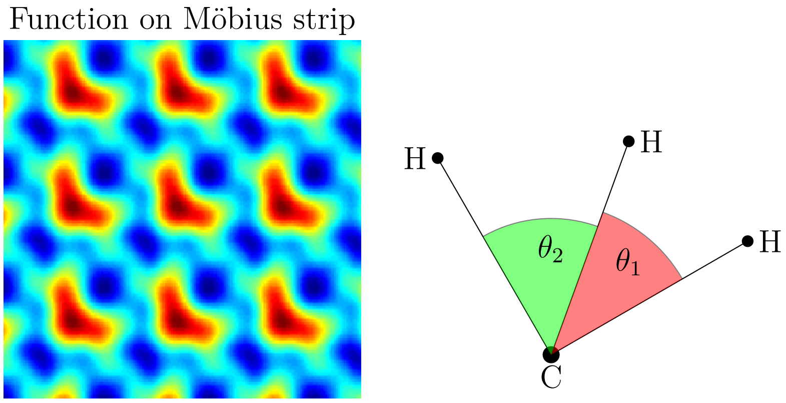
however, the second method will in general lead to non-differentiability along $x = y$. Figure 1.8 shows the difference.

1.6.4 Checking for Symmetries

In some situations, we might not know *a priori* whether a particular symmetry is present in a function. Because GPs let us build models both with and without certain symmetries, we can compute the amount of evidence that the data provide for each of these possibilities. To do so, we simple need to compare the marginal likelihood of the data We demonstrate that marginal likeihood an be used to automatically search over such structures.

1.6.5 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.



Function on Möbius strip

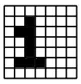
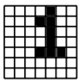
Fig. 1.9 An example of a function expressing the same symmetries as a Möbius strip in two of its arguments. The energy of a molecular configuration $f(\theta_1, \theta_2)$ depends only on the relative angles between atoms, and because each atom is indistiguishable, is invariant to permuting the atoms.

1.6.6 Example: Translation invariance in images

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

Note that this sort of translational invariance is completely distinct from the stationarity 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:

$f\Big($  $\Big) = f\Big($  $\Big)$ (1.28)

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:

$$k\left((x_1, x_2, \dots, x_D), (x'_1, x'_2, \dots, x'_D)\right) = \sum_{i=1}^{D} \prod_{j=1}^{D} k(x_j, x'_{i+j \bmod D}) \tag{1.29}$$

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:

$$k_{\text{conv}}\left((x_{11}, x_{12}, \ldots, x_{RC}), (x'_{11}, x'_{12}, \ldots, x'_{RC})\right) = \sum_{i=-L}^{L} \sum_{j=-L}^{L} k\left(\mathbf{x}, T_{ij}(\mathbf{x}')\right) \tag{1.30}$$

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.

1.7 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.

1.7.1 Surfaces, Cylinders and Torii

Figure 1.10 shows...

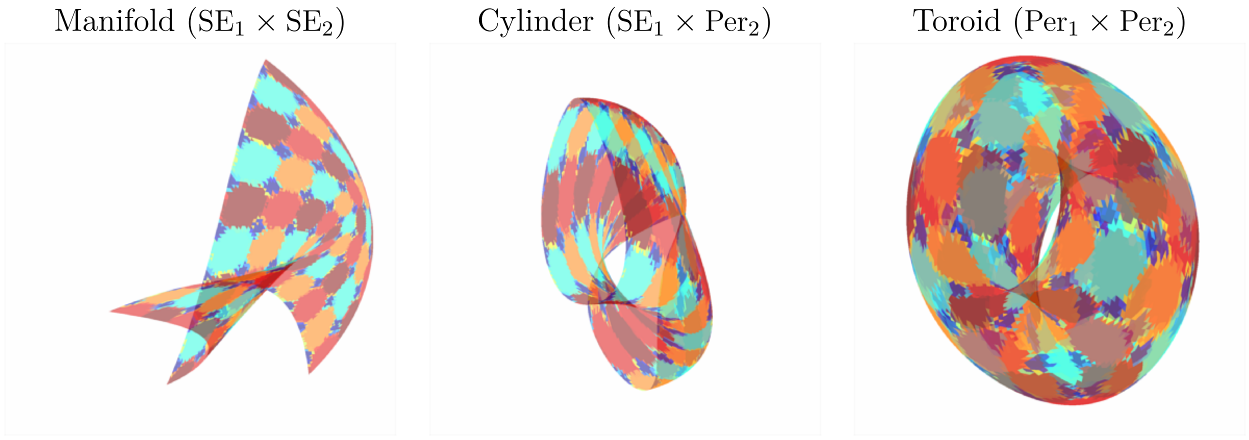
1.7.2 Möbius strips

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

$$f(x, y) = f(x, y + \tau_y) \tag{1.31}$$

$$f(x, y) = f(x + \tau_x, y) \tag{1.32}$$

$$f(x, y) = f(y, x) \tag{1.33}$$



Manifold ($\mathrm{SE}_1 \times \mathrm{SE}_2$) Cylinder ($\mathrm{SE}_1 \times \mathrm{Per}_2$) Toroid ($\mathrm{Per}_1 \times \mathrm{Per}_2$)

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.

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 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

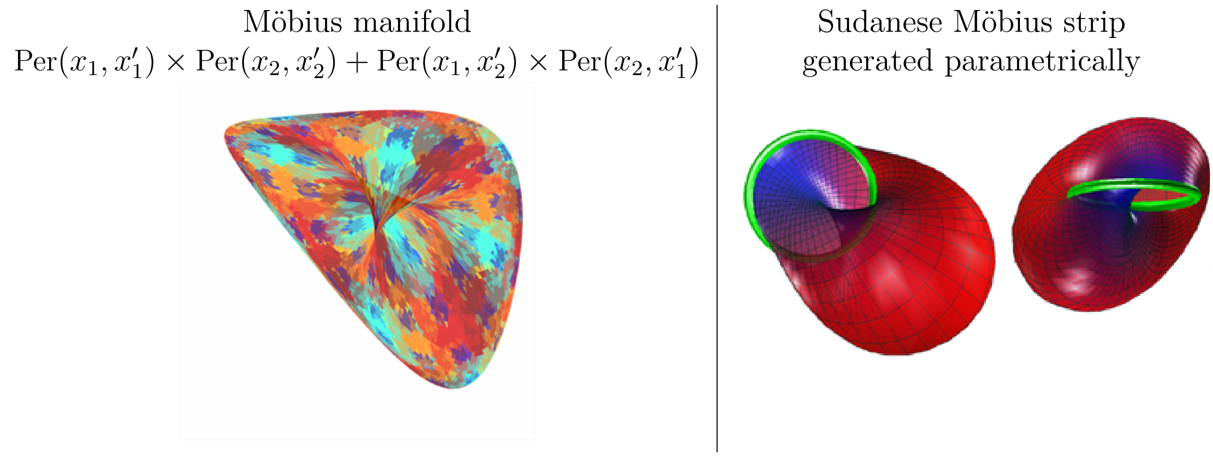
1.8 Kernels on discrete data

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.

1.8.1 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



Möbius manifold

$\mathrm{Per}(x_1, x'_1) \times \mathrm{Per}(x_2, x'_2) + \mathrm{Per}(x_1, x'_2) \times \mathrm{Per}(x_2, x'_1)$

Sudanese Möbius strip generated parametrically

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).

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.

1.9 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:

$$f \sim \mathcal{GP}(cm(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad c \sim \mathcal{N}\Big(0, \sigma_c^2\Big) \iff f \sim \mathcal{GP}\Big(\mathbf{0}, c^2 m(\mathbf{x})m(\mathbf{x}') + k(\mathbf{x}, \mathbf{x}')\Big) \tag{1.34}$$

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.

$$m \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}')) \,, \quad f \sim \mathcal{GP}(m(\mathbf{x}), k_2(\mathbf{x}, \mathbf{x}')) \iff f \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')) \tag{1.35}$$

1.9.1 Why not simply learn the mean function?

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.

1.10 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

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png. (page 17)

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 11)

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 11)

T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 4)

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 15)

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 8)

D. Lerner and D. Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984. (page 17)

T.A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using Gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417. (page 7)

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in AI*, pages 689–690. IEEE, 2011. (page 8)

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*, volume 38. The MIT Press, Cambridge, MA, USA, 2006. (page 3)