15 *Gaussian processes*

15.1 Introduction

In supervised learning, we observe some inputs $\mathbf{x}_i$ and some outputs y_i. We assume that $y_i = f(\mathbf{x}_i)$, for some unknown function f, possibly corrupted by noise. The optimal approach is to infer a *distribution over functions* given the data, $p(f|\mathbf{X}, \mathbf{y})$, and then to use this to make predictions given new inputs, i.e., to compute

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|f, \mathbf{x}_*)p(f|\mathbf{X}, \mathbf{y})df \tag{15.1}$$

Up until now, we have focussed on parametric representations for the function f, so that instead of inferring $p(f|\mathcal{D})$, we infer $p(\boldsymbol{\theta}|\mathcal{D})$. In this chapter, we discuss a way to perform Bayesian inference over functions themselves.

Our approach will be based on **Gaussian processes** or **GPs**. A GP defines a prior over functions, which can be converted into a posterior over functions once we have seen some data. Although it might seem difficult to represent a distribution over a function, it turns out that we only need to be able to define a distribution over the function's values at a finite, but arbitrary, set of points, say $\mathbf{x}_1, \ldots, \mathbf{x}_N$. A GP assumes that $p(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))$ is jointly Gaussian, with some mean $\boldsymbol{\mu}(\mathbf{x})$ and covariance $\boldsymbol{\Sigma}(\mathbf{x})$ given by $\Sigma_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, where κ is a positive definite kernel function (see Section 14.2 information on kernels). The key idea is that if $\mathbf{x}_i$ and $\mathbf{x}_j$ are deemed by the kernel to be similar, then we expect the output of the function at those points to be similar, too. See Figure 15.1 for an illustration.

It turns out that, in the regression setting, all these computations can be done in closed form, in $O(N^3)$ time. (We discuss faster approximations in Section 15.6.) In the classification setting, we must use approximations, such as the Gaussian approximation, since the posterior is no longer exactly Gaussian.

GPs can be thought of as a Bayesian alternative to the kernel methods we discussed in Chapter 14, including LIVM, RVM and SVM. Although those methods are sparser and therefore faster, they do not give well-calibrated probabilistic outputs (see Section 15.4.4 for further discussion). Having properly tuned probabilistic output is important in certain applications, such as online tracking for vision and robotics (Ko and Fox 2009), reinforcement learning and optimal control (Engel et al. 2005; Deisenroth et al. 2009), global optimization of non-convex functions (Mockus et al. 1996; Lizotte 2008; Brochu et al. 2009), experiment design (Santner et al. 2003), etc.

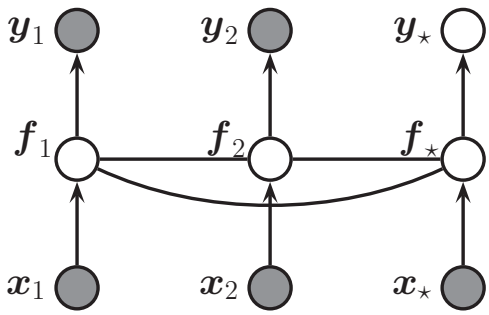


Figure 15.1 A Gaussian process for 2 training points and 1 testing point, represented as a mixed directed and undirected graphical model representing $p(\mathbf{y}, \mathbf{f}|\mathbf{x}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}(\mathbf{x})) \prod_i p(y_i|f_i)$. The hidden nodes $f_i = f(\mathbf{x}_i)$ represent the value of the function at each of the data points. These hidden nodes are fully interconnected by undirected edges, forming a Gaussian graphical model; the edge strengths represent the covariance terms $\Sigma_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. If the test point $\mathbf{x}_*$ is similar to the training points $\mathbf{x}_1$ and $\mathbf{x}_2$, then the predicted output y_* will be similar to y_1 and y_2.

Our presentation is closely based on (Rasmussen and Williams 2006), which should be consulted for futher details. See also (Diggle and Ribeiro 2007), which discusses the related approach known as **kriging**, which is widely used in the spatial statistics literature.

15.2 GPs for regression

In this section, we discuss GPs for regression. Let the prior on the regression function be a GP, denoted by

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}')) \tag{15.2}$$

where $m(\mathbf{x})$ is the mean function and $\kappa(\mathbf{x}, \mathbf{x}')$ is the kernel or covariance function, i.e.,

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{15.3}$$

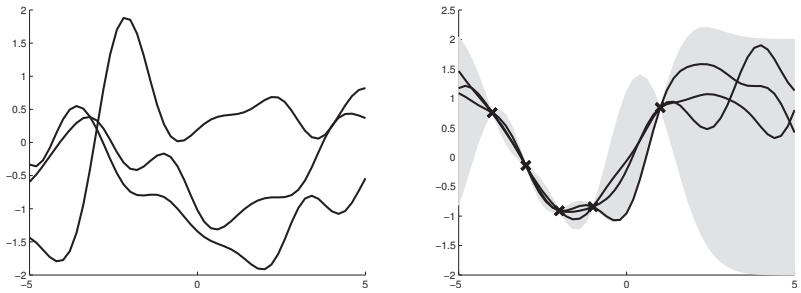
$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^T\right] \tag{15.4}$$

We obviously require that $\kappa()$ be a positive definite kernel. For any finite set of points, this process defines a joint Gaussian:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K}) \tag{15.5}$$

where $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ and $\boldsymbol{\mu} = (m(\mathbf{x}_1), \ldots, m(\mathbf{x}_N))$.

Note that it is common to use a mean function of $m(\mathbf{x}) = 0$, since the GP is flexible enough to model the mean arbitrarily well, as we will see below. However, in Section 15.2.6 we will consider parametric models for the mean function, so the GP just has to model the residual errors. This semi-parametric approach combines the interpretability of parametric models with the accuracy of non-parametric models.



(a) (b)

Figure 15.2 Left: some functions sampled from a GP prior with SE kernel. Right: some samples from a GP posterior, after conditioning on 5 noise-free observations. The shaded area represents $\mathbb{E}[f(\mathbf{x})] \pm 2\text{std}(f(\mathbf{x}))$. Based on Figure 2.2 of (Rasmussen and Williams 2006). Figure generated by `gprDemoNoiseFree`.

15.2.1 Predictions using noise-free observations

Suppose we observe a training set $\mathcal{D} = \{(\mathbf{x}_i, f_i), i = 1 : N\}$, where $f_i = f(\mathbf{x}_i)$ is the noise-free observation of the function evaluated at $\mathbf{x}_i$. Given a test set $\mathbf{X}_*$ of size $N_* \times D$, we want to predict the function outputs $\mathbf{f}_*$.

If we ask the GP to predict $f(\mathbf{x})$ for a value of $\mathbf{x}$ that it has already seen, we want the GP to return the answer $f(\mathbf{x})$ with no uncertainty. In other words, it should act as an **interpolator** of the training data. This will only happen if we assume the observations are noiseless. We will consider the case of noisy observations below.

Now we return to the prediction problem. By definition of the GP, the joint distribution has the following form

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right) \tag{15.6}$$

where $\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$ is $N \times N$, $\mathbf{K}_* = \kappa(\mathbf{X}, \mathbf{X}_*)$ is $N \times N_*$, and $\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*)$ is $N_* \times N_*$. By the standard rules for conditioning Gaussians (Section 4.3), the posterior has the following form

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{15.7}$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1}(\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})) \tag{15.8}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \tag{15.9}$$

This process is illustrated in Figure 15.2. On the left we show sample samples from the prior, $p(\mathbf{f}|\mathbf{X})$, where we use a **squared exponential kernel**, aka Gaussian kernel or RBF kernel. In 1d, this is given by

$$\kappa(x, x') = \sigma_f^2 \exp(-\frac{1}{2\ell^2}(x - x')^2) \tag{15.10}$$

Here ℓ controls the horizontal length scale over which the function varies, and σ_f^2 controls the vertical variation. (We discuss how to estimate such kernel parameters below.) On the right we

show samples from the posterior, $p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f})$. We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.

One application of noise-free GP regression is as a computationally cheap proxy for the behavior of a complex simulator, such as a weather forecasting program. (If the simulator is stochastic, we can define f to be its mean output; note that there is still no observation noise.) One can then estimate the effect of changing simulator parameters by examining their effect on the GP's predictions, rather than having to run the simulator many times, which may be prohibitively slow. This strategy is known as DACE, which stands for design and analysis of computer experiments (Santner et al. 2003).

15.2.2 Predictions using noisy observations

Now let us consider the case where what we observe is a noisy version of the underlying function, $y = f(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. In this case, the model is not required to interpolate the data, but it must come "close" to the observed data. The covariance of the observed noisy responses is

$$\text{cov}\left[y_p, y_q\right] = \kappa(\mathbf{x}_p, \mathbf{x}_q) + \sigma_y^2 \delta_{pq} \tag{15.11}$$

where $\delta_{pq} = \mathbb{I}(p = q)$. In other words

$$\text{cov}\left[\mathbf{y}|\mathbf{X}\right] = \mathbf{K} + \sigma_y^2 \mathbf{I}_N \triangleq \mathbf{K}_y \tag{15.12}$$

The second matrix is diagonal because we assumed the noise terms were independently added to each observation.

The joint density of the observed data and the latent, noise-free function on the test points is given by

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_y & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) \tag{15.13}$$

where we are assuming the mean is zero, for notational simplicity. Hence the posterior predictive density is

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{15.14}$$

$$\boldsymbol{\mu}_* = \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{y} \tag{15.15}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_* \tag{15.16}$$

In the case of a single test input, this simplifies as follows

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(f_*|\mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{y}, k_{**} - \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{k}_*) \tag{15.17}$$

where $\mathbf{k}_* = [\kappa(\mathbf{x}_*, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}_*, \mathbf{x}_N)]$ and $k_{**} = \kappa(\mathbf{x}_*, \mathbf{x}_*)$. Another way to write the posterior mean is as follows:

$$\overline{f}_* = \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{y} = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}_*) \tag{15.18}$$

where $\boldsymbol{\alpha} = \mathbf{K}_y^{-1} \mathbf{y}$. We will revisit this expression later.

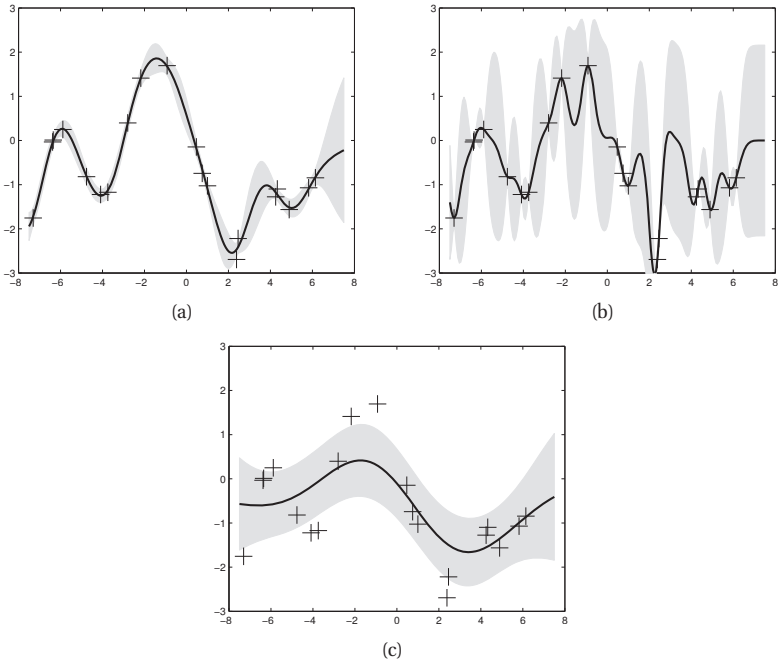


Figure 15.3 Some 1d GPs with SE kernels but different hyper-parameters fit to 20 noisy observations. The kernel has the form in Equation 15.19. The hyper-parameters $(\ell, \sigma_f, \sigma_y)$ are as follows: (a) (1,1,0.1) (b) (0.3, 0.1.08, 0.00005), (c) (3.0, 1.16, 0.89). Based on Figure 2.5 of (Rasmussen and Williams 2006). Figure generated by `gprDemoChangeHparams`, written by Carl Rasmussen.

15.2.3 Effect of the kernel parameters

The predictive performance of GPs depends exclusively on the suitability of the chosen kernel. Suppose we choose the following squared-exponential (SE) kernel for the noisy observations

$$\kappa_y(x_p, x_q) = \sigma_f^2 \exp(-\frac{1}{2\ell^2}(x_p - x_q)^2) + \sigma_y^2 \delta_{pq} \tag{15.19}$$

Here ℓ is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale of the function, and σ_y^2 is the noise variance. Figure 15.3 illustrates the effects of changing these parameters. We sampled 20 noisy data points from the SE kernel using $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$, and then made predictions various parameters, conditional on the data. In Figure 15.3(a), we use $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$, and the result is a good fit. In Figure 15.3(b), we reduce the length scale to $\ell = 0.3$ (the other parameters were optimized by maximum (marginal) likelihood, a technique we discuss below); now the function looks more "wiggly". Also, the uncertainty goes up faster, since the effective distance from the training points increases more rapidly. In Figure 15.3(c), we increase the length scale to $\ell = 3$; now the function looks smoother.

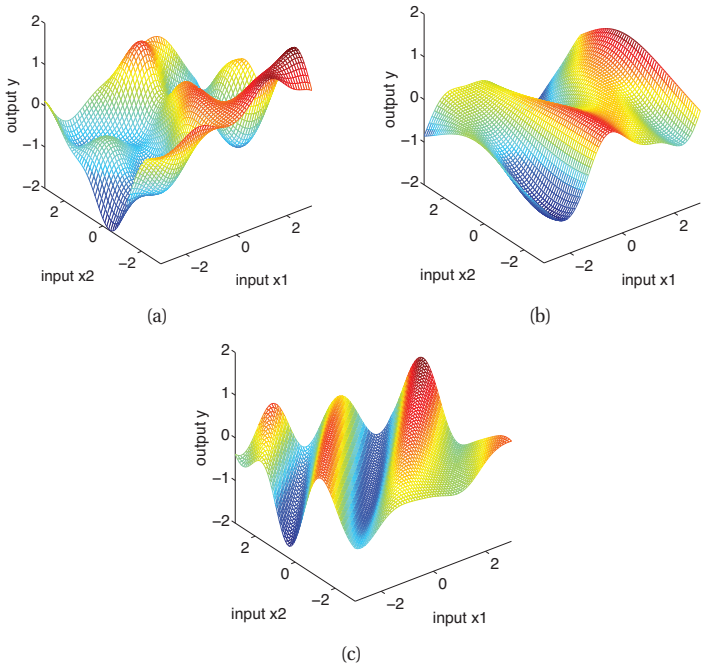


Figure 15.4 Some 2d functions sampled from a GP with an SE kernel but different hyper-parameters. The kernel has the form in Equation 15.20 where (a) $\mathbf{M} = \mathbf{I}$, (b) $\mathbf{M} = \text{diag}(1, 3)^{-2}$, (c) $\mathbf{M} = (1, -1; -1, 1) + \text{diag}(6, 6)^{-2}$. Based on Figure 5.1 of (Rasmussen and Williams 2006). Figure generated by `gprDemoArd`, written by Carl Rasmussen.

We can extend the SE kernel to multiple dimensions as follows:

$$\kappa_y(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T \mathbf{M}(\mathbf{x}_p - \mathbf{x}_q)) + \sigma_y^2 \delta_{pq} \tag{15.20}$$

We can define the matrix $\mathbf{M}$ in several ways. The simplest is to use an isotropic matrix, $\mathbf{M}_1 = \ell^{-2}\mathbf{I}$. See Figure 15.4(a) for an example. We can also endow each dimension with its own characteristic length scale, $\mathbf{M}_2 = \text{diag}(\boldsymbol{\ell})^{-2}$. If any of these length scales become large, the corresponding feature dimension is deemed "irrelevant", just as in ARD (Section 13.7). In Figure 15.4(b), we use $\mathbf{M} = \mathbf{M}_2$ with $\boldsymbol{\ell} = (1, 3)$, so the function changes faster along the x_1 direction than the x_2 direction.

We can also create a matrix of the form $\mathbf{M}_3 = \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T + \text{diag}(\boldsymbol{\ell})^{-2}$, where $\boldsymbol{\Lambda}$ is a $D \times K$ matrix, where $K < D$. (Rasmussen and Williams 2006, p107) calls this the **factor analysis distance** function, by analogy to the fact that factor analysis (Section 12.1) approximates a covariance matrix as a low rank matrix plus a diagonal matrix. The columns of $\boldsymbol{\Lambda}$ correspond to relevant directions in input space. In Figure 15.4(c), we use $\boldsymbol{\ell} = (6; 6)$ and $\boldsymbol{\Lambda} = (1; -1)$, so the function changes mostly rapidly in the direction which is perpendicular to (1,1).

15.2.4 Estimating the kernel parameters

To estimate the kernel parameters, we could use exhaustive search over a discrete grid of values, with validation loss as an objective, but this can be quite slow. (This is the approach used to tune kernels used by SVMs.) Here we consider an empirical Bayes approach, which will allow us to use continuous optimization methods, which are much faster. In particular, we will maximize the marginal likelihood[1]

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f},\mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f} \tag{15.21}$$

Since $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0},\mathbf{K})$, and $p(\mathbf{y}|\mathbf{f}) = \prod_i \mathcal{N}(y_i|f_i,\sigma_y^2)$, the marginal likelihood is given by

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0},\mathbf{K}_y) = -\frac{1}{2}\mathbf{y}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}_y| - \frac{N}{2}\log(2\pi) \tag{15.22}$$

The first term is a data fit term, the second term is a model complexity term, and the third term is just a constant. To understand the tradeoff between the first two terms, consider a SE kernel in 1D, as we vary the length scale ℓ and hold σ_y^2 fixed. Let $J(\ell) = -\log p(\mathbf{y}|\mathbf{X},\ell)$. For short length scales, the fit will be good, so $\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y}$ will be small. However, the model complexity will be high: $\mathbf{K}$ will be almost diagonal (as in Figure 14.3, top right), since most points will not be considered "near" any others, so the $\log|\mathbf{K}_y|$ will be large. For long length scales, the fit will be poor but the model complexity will be low: $\mathbf{K}$ will be almost all 1's (as in Figure 14.3, bottom right), so $\log|\mathbf{K}_y|$ will be small.

We now discuss how to maximize the marginal likelhiood. Let the kernel parameters (also called hyper-parameters) be denoted by $\boldsymbol{\theta}$. One can show that

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{X}) = \frac{1}{2}\mathbf{y}^T\mathbf{K}_y^{-1}\frac{\partial\mathbf{K}_y}{\partial\theta_j}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(\mathbf{K}_y^{-1}\frac{\partial\mathbf{K}_y}{\partial\theta_j}) \tag{15.23}$$

$$= \frac{1}{2}\mathrm{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \mathbf{K}_y^{-1})\frac{\partial\mathbf{K}_y}{\partial\theta_j}\right) \tag{15.24}$$

where $\boldsymbol{\alpha} = \mathbf{K}_y^{-1}\mathbf{y}$. It takes $O(N^3)$ time to compute $\mathbf{K}_y^{-1}$, and then $O(N^2)$ time per hyper-parameter to compute the gradient.

The form of $\frac{\partial\mathbf{K}_y}{\partial\theta_j}$ depends on the form of the kernel, and which parameter we are taking derivatives with respect to. Often we have constraints on the hyper-parameters, such as $\sigma_y^2 \geq 0$. In this case, we can define $\theta = \log(\sigma_y^2)$, and then use the chain rule.

Given an expression for the log marginal likelihood and its derivative, we can estimate the kernel parameters using any standard gradient-based optimizer. However, since the objective is not convex, local minima can be a problem, as we illustrate below.

15.2.4.1 Example

Consider Figure 15.5. We use the SE kernel in Equation 15.19 with $\sigma_f^2 = 1$, and plot $\log p(\mathbf{y}|\mathbf{X},\ell,\sigma_y^2)$ (where $\mathbf{X}$ and $\mathbf{y}$ are the 7 data points shown in panels b and c) as we vary ℓ and σ_y^2. The two

1. The reason it is called the marginal likelihood, rather than just likelihood, is because we have marginalized out the latent Gaussian vector $\mathbf{f}$. This moves us up one level of the Bayesian hierarchy, and reduces the chances of overfitting (the number of kernel parameters is usually fairly small compared to a standard parametric model).

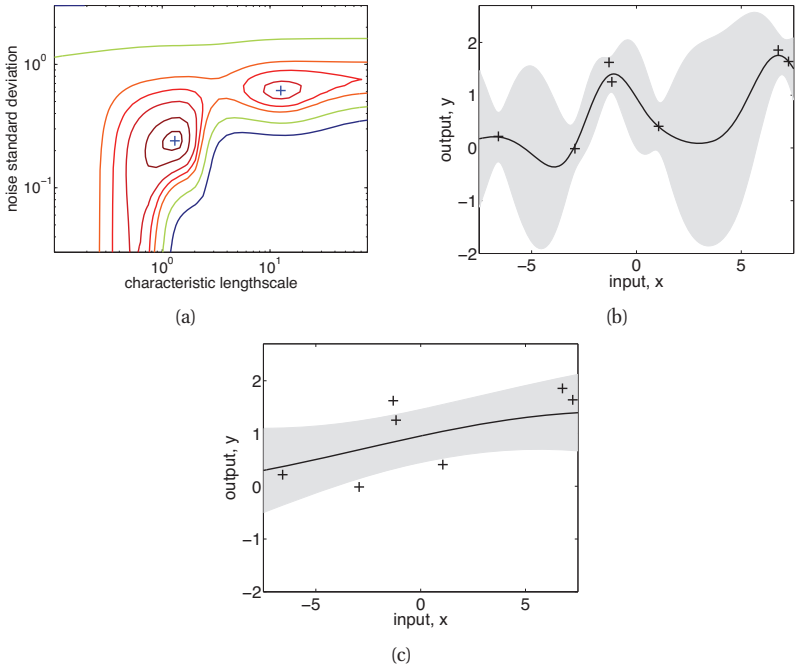


Figure 15.5 Illustration of local minima in the marginal likelihood surface. (a) We plot the log marginal likelihood vs σ_y^2 and ℓ, for fixed $\sigma_f^2 = 1$, using the 7 data points shown in panels b and c. (b) The function corresponding to the lower left local minimum, $(\ell, \sigma_n^2) \approx (1, 0.2)$. This is quite "wiggly" and has low noise. (c) The function corresponding to the top right local minimum, $(\ell, \sigma_n^2) \approx (10, 0.8)$. This is quite smooth and has high noise. The data was generated using $(\ell, \sigma_n^2) = (1, 0.1)$. Source: Figure 5.5 of (Rasmussen and Williams 2006). Figure generated by `gprDemoMarglik`, written by Carl Rasmussen.

local optima are indicated by +. The bottom left optimum corresponds to a low-noise, short-length scale solution (shown in panel b). The top right optimum corresponds to a high-noise, long-length scale solution (shown in panel c). With only 7 data points, there is not enough evidence to confidently decide which is more reasonable, although the more complex model (panel b) has a marginal likelihood that is about 60% higher than the simpler model (panel c). With more data, the MAP estimate should come to dominate.

Figure 15.5 illustrates some other interesting (and typical) features. The region where $\sigma_y^2 \approx 1$ (top of panel a) corresponds to the case where the noise is very high; in this regime, the marginal likelihood is insensitive to the length scale (indicated by the horizontal contours), since all the data is explained as noise. The region where $\ell \approx 0.5$ (left hand side of panel a) corresponds to the case where the length scale is very short; in this regime, the marginal likelihood is insensitive to the noise level, since the data is perfectly interpolated. Neither of these regions would be chosen by a good optimizer.

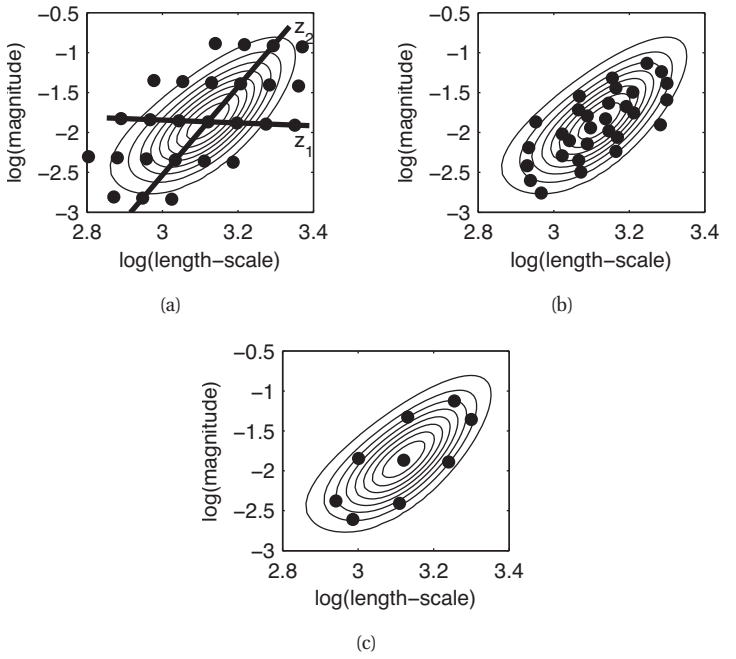


Figure 15.6 Three different approximations to the posterior over hyper-parameters: grid-based, Monte Carlo, and central composite design. Source: Figure 3.2 of (Vanhatalo 2010). Used with kind permission of Jarno Vanhatalo.

15.2.4.2 Bayesian inference for the hyper-parameters

An alternative to computing a point estimate of the hyper-parameters is to compute their posterior. Let $\boldsymbol{\theta}$ represent all the kernel parameters, as well as σ_y^2. If the dimensionality of $\boldsymbol{\theta}$ is small, we can compute a discrete grid of possible values, centered on the MAP estimate $\hat{\boldsymbol{\theta}}$ (computed as above). We can then approximate the posterior over the latent variables using

$$p(\mathbf{f}|\mathcal{D}) \propto \sum_{s=1}^{S} p(\mathbf{f}|\mathcal{D}, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_s|\mathcal{D}) \delta_s \tag{15.25}$$

where δ_s denotes the weight for grid point s.

In higher dimensions, a regular grid suffers from the curse of dimensionality. An obvious alternative is Monte Carlo, but this can be slow. Another approach is to use a form of quasi-Monte Carlo, whereby we place grid points at the mode, and at a distance ± 1sd from the mode along each dimension, for a total of $2|\boldsymbol{\theta}| + 1$ points. This is called a **central composite design** (Rue et al. 2009). (This is also used in the unscented Kalman filter, see Section 18.5.2.) To make this Gaussian-like approximation more reasonable, we often log-transform the hyper-parameters. See Figure 15.6 for an illustration.

15.2.4.3 Multiple kernel learning

A quite different approach to optimizing kernel parameters known as **multiple kernel learning**. The idea is to define the kernel as a weighted sum of base kernels, $\kappa(\mathbf{x}, \mathbf{x}') = \sum_j w_j \kappa_j(\mathbf{x}, \mathbf{x}')$, and then to optimize the weights w_j instead of the kernel parameters themselves. This is particularly useful if we have different kinds of data which we wish to fuse together. See e.g., (Rakotomamonjy et al. 2008) for an approach based on risk-minimization and convex optimization, and (Girolami and Rogers 2005) for an approach based on variational Bayes.

15.2.5 Computational and numerical issues *

The predictive mean is given by $\overline{f_*} = \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{y}$. For reasons of numerical stability, it is unwise to directly invert $\mathbf{K}_y$. A more robust alternative is to compute a Cholesky decomposition, $\mathbf{K}_y = \mathbf{L}\mathbf{L}^T$. We can then compute the predictive mean and variance, and the log marginal likelihood, as shown in the pseudo-code in Algorithm 6 (based on (Rasmussen and Williams 2006, p19)). It takes $O(N^3)$ time to compute the Cholesky decomposition, and $O(N^2)$ time to solve for $\boldsymbol{\alpha} = \mathbf{K}_y^{-1}\mathbf{y} = \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{y}$. We can then compute the mean using $\mathbf{k}_*^T \boldsymbol{\alpha}$ in $O(N)$ time and the variance using $k_{**} - \mathbf{k}_*^T \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{k}_*$ in $O(N^2)$ time for each test case.

An alternative to Cholesky decomposition is to solve the linear system $\mathbf{K}_y \boldsymbol{\alpha} = \mathbf{y}$ using conjugate gradients (CG). If we terminate this algorithm after k iterations, it takes $O(kN^2)$ time. If we run for $k = N$, it gives the exact solution in $O(N^3)$ time. Another approach is to approximate the matrix-vector multiplies needed by CG using the fast Gauss transform. (Yang et al. 2005); however, this doesn't scale to high-dimensional inputs. See also Section 15.6 for a discussion of other speedup techniques.

Algorithm 15.1: GP regression

1 $\mathbf{L} = \text{cholesky}(\mathbf{K} + \sigma_y^2 \mathbf{I})$;
2 $\boldsymbol{\alpha} = \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{y})$;
3 $\mathbb{E}[f_*] = \mathbf{k}_*^T \boldsymbol{\alpha}$;
4 $\mathbf{v} = \mathbf{L} \setminus \mathbf{k}_*$;
5 $\text{var}[f_*] = \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$;
6 $\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2}\log(2\pi)$

15.2.6 Semi-parametric GPs *

Sometimes it is useful to use a linear model for the mean of the process, as follows:

$$f(\mathbf{x}) = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) + r(\mathbf{x}) \tag{15.26}$$

where $r(\mathbf{x}) \sim \text{GP}(0, \kappa(\mathbf{x}, \mathbf{x}'))$ models the residuals. This combines a parametric and a non-parametric model, and is known as a **semi-parametric model**.

If we assume $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{b}, \mathbf{B})$, we can integrate these parameters out to get a new GP (O'Hagan 1978):

$$f(\mathbf{x}) \sim \text{GP}\left(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{b},\ \kappa(\mathbf{x}, \mathbf{x}') + \boldsymbol{\phi}(\mathbf{x})^T \mathbf{B} \boldsymbol{\phi}(\mathbf{x}')\right) \tag{15.27}$$

$\log p(y_i\|f_i)$	$\frac{\partial}{\partial f_i}\log p(y_i\|f_i)$	$\frac{\partial^2}{\partial f_i^2}\log p(y_i\|f_i)$
$\log \text{sigm}(y_i f_i)$	$t_i - \pi_i$	$-\pi_i(1-\pi_i)$
$\log \Phi(y_i f_i)$	$\frac{y_i \phi(f_i)}{\Phi(y_i f_i)}$	$-\frac{\phi_i^2}{\Phi(y_i f_i)^2} - \frac{y_i f_i \phi(f_i)}{\Phi(y_i f_i)}$

Table 15.1 Likelihood, gradient and Hessian for binary logistic/ probit GP regression. We assume $y_i \in \{-1, +1\}$ and define $t_i = (y_i + 1)/2 \in \{0, 1\}$ and $\pi_i = \text{sigm}(f_i)$ for logistic regression, and $\pi_i = \Phi(f_i)$ for probit regression. Also, ϕ and Φ are the pdf and cdf of $\mathcal{N}(0, 1)$. From (Rasmussen and Williams 2006, p43).

Integrating out $\boldsymbol{\beta}$, the corresponding predictive distribution for test inputs $\mathbf{X}_*$ has the following form (Rasmussen and Williams 2006, p28):

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\overline{\mathbf{f}_*}, \text{cov}\,[f_*]) \quad (15.28)$$

$$\overline{\mathbf{f}_*} = \boldsymbol{\Phi}_*^T \overline{\boldsymbol{\beta}} + \mathbf{K}_*^T \mathbf{K}_y^{-1}(\mathbf{y} - \boldsymbol{\Phi}\overline{\boldsymbol{\beta}}) \quad (15.29)$$

$$\overline{\boldsymbol{\beta}} = (\boldsymbol{\Phi}^T \mathbf{K}_y^{-1} \boldsymbol{\Phi} + \mathbf{B}^{-1})^{-1}(\boldsymbol{\Phi}\mathbf{K}_y^{-1}\mathbf{y} + \mathbf{B}^{-1}\mathbf{b}) \quad (15.30)$$

$$\text{cov}\,[\mathbf{f}_*] = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_* + \mathbf{R}^T(\mathbf{B}^{-1} + \boldsymbol{\Phi}\mathbf{K}_y^{-1}\boldsymbol{\Phi}^T)^{-1}\mathbf{R} \quad (15.31)$$

$$\mathbf{R} = \boldsymbol{\Phi}_* - \boldsymbol{\Phi}\mathbf{K}_y^{-1}\boldsymbol{\Phi}_* \quad (15.32)$$

The predictive mean is the output of the linear model plus a correction term due to the GP, and the predictive covariance is the usual GP covariance plus an extra term due to the uncertainty in $\boldsymbol{\beta}$.

15.3 GPs meet GLMs

In this section, we extend GPs to the GLM setting, focussing on the classification case. As with Bayesian logistic regression, the main difficulty is that the Gaussian prior is not conjugate to the bernoulli/ multinoulli likelihood. There are several approximations one can adopt: Gaussian approximation (Section 8.4.3), expectation propagation (Kuss and Rasmussen 2005; Nickisch and Rasmussen 2008), variational (Girolami and Rogers 2006; Opper and Archambeau 2009), MCMC (Neal 1997; Christensen et al. 2006), etc. Here we focus on the Gaussian approximation, since it is the simplest and fastest.

15.3.1 Binary classification

In the binary case, we define the model as $p(y_i|\mathbf{x}_i) = \sigma(y_i f(\mathbf{x}_i))$, where, following (Rasmussen and Williams 2006), we assume $y_i \in \{-1, +1\}$, and we let $\sigma(z) = \text{sigm}(z)$ (logistic regression) or $\sigma(z) = \Phi(z)$ (probit regression). As for GP regression, we assume $f \sim \text{GP}(0, \kappa)$.

15.3.1.1 Computing the posterior

Define the log of the unnormalized posterior as follows:

$$\ell(\mathbf{f}) = \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|\mathbf{X}) = \log p(\mathbf{y}|\mathbf{f}) - \frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathbf{K}| - \frac{N}{2}\log 2\pi \quad (15.33)$$

Let $J(f) \triangleq -\ell(f)$ be the function we want to minimize. The gradient and Hessian of this are given by

$$\mathbf{g} = -\nabla \log p(\mathbf{y}|\mathbf{f}) + \mathbf{K}^{-1}\mathbf{f} \tag{15.34}$$

$$\mathbf{H} = -\nabla\nabla \log p(\mathbf{y}|\mathbf{f}) + \mathbf{K}^{-1} = \mathbf{W} + \mathbf{K}^{-1} \tag{15.35}$$

Note that $\mathbf{W} \triangleq -\nabla\nabla \log p(\mathbf{y}|\mathbf{f})$ is a diagonal matrix because the data are iid (conditional on $\mathbf{f}$). Expressions for the gradient and Hessian of the log likelihood for the logit and probit case are given in Sections 8.3.1 and 9.4.1, and summarized in Table 15.1.

We can use IRLS to find the MAP estimate. The update has the form

$$\mathbf{f}^{new} = \mathbf{f} - \mathbf{H}^{-1}\mathbf{g} = \mathbf{f} + (\mathbf{K}^{-1} + \mathbf{W})^{-1}(\nabla \log p(\mathbf{y}|\mathbf{f}) - \mathbf{K}^{-1}\mathbf{f}) \tag{15.36}$$

$$= (\mathbf{K}^{-1} + \mathbf{W})^{-1}(\mathbf{W}\mathbf{f} + \nabla \log p(\mathbf{y}|\mathbf{f})) \tag{15.37}$$

At convergence, the Gaussian approximation of the posterior takes the following form:

$$p(\mathbf{f}|\mathbf{X}, \mathbf{y}) \approx \mathcal{N}(\hat{\mathbf{f}}, (\mathbf{K}^{-1} + \mathbf{W})^{-1}) \tag{15.38}$$

15.3.1.2 Computing the posterior predictive

We now compute the posterior predictive. First we predict the latent function at the test case $\mathbf{x}_*$. For the mean we have

$$\mathbb{E}[f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}] = \int \mathbb{E}[f_*|\mathbf{f}, \mathbf{x}_*, \mathbf{X}, \mathbf{y}]\, p(\mathbf{f}|\mathbf{X}, \mathbf{y}) d\mathbf{f} \tag{15.39}$$

$$= \int \mathbf{k}_*^T \mathbf{K}^{-1}\mathbf{f}\ p(\mathbf{f}|\mathbf{X}, \mathbf{y}) d\mathbf{f} \tag{15.40}$$

$$= \mathbf{k}_*^T \mathbf{K}^{-1}\mathbb{E}[\mathbf{f}|\mathbf{X}, \mathbf{y}] \approx \mathbf{k}_*^T \mathbf{K}^{-1}\hat{\mathbf{f}} \tag{15.41}$$

where we used Equation 15.8 to get the mean of f_* given noise-free $\mathbf{f}$.

To compute the predictive variance, we use the rule of iterated variance:

$$\mathrm{var}[f_*] = \mathbb{E}[\mathrm{var}[f_*|\mathbf{f}]] + \mathrm{var}[\mathbb{E}[f_*|\mathbf{f}]] \tag{15.42}$$

where all probabilities are conditioned on $\mathbf{x}_*, \mathbf{X}, \mathbf{y}$. From Equation 15.9 we have

$$\mathbb{E}[\mathrm{var}[f_*|\mathbf{f}]] = \mathbb{E}\left[k_{**} - \mathbf{k}_*^T \mathbf{K}^{-1}\mathbf{k}_*\right] = k_{**} - \mathbf{k}_*^T \mathbf{K}^{-1}\mathbf{k}_* \tag{15.43}$$

From Equation 15.9 we have

$$\mathrm{var}[\mathbb{E}[f_*|\mathbf{f}]] = \mathrm{var}\left[\mathbf{k}_*\mathbf{K}^{-1}\mathbf{f}\right] = \mathbf{k}_*^T \mathbf{K}^{-1}\mathrm{cov}[\mathbf{f}]\, \mathbf{K}^{-1}\mathbf{k}_* \tag{15.44}$$

Combining these we get

$$\mathrm{var}[f_*] = k_{**} - \mathbf{k}_*^T(\mathbf{K}^{-1} - \mathbf{K}^{-1}\mathrm{cov}[\mathbf{f}]\, \mathbf{K}^{-1})\mathbf{k}_* \tag{15.45}$$

From Equation 15.38 we have $\mathrm{cov}[f] \approx (\mathbf{K}^{-1} + \mathbf{W})^{-1}$. Using the matrix inversion lemma we get

$$\mathrm{var}[f_*] \approx k_{**} - \mathbf{k}_*^T \mathbf{K}^{-1}\mathbf{k}_* + \mathbf{k}_*^T \mathbf{K}^{-1}(\mathbf{K}^{-1} + \mathbf{W})^{-1}\mathbf{K}^{-1}\mathbf{k}_* \tag{15.46}$$

$$= k_{**} - \mathbf{k}_*^T(\mathbf{K} + \mathbf{W}^{-1})^{-1}\mathbf{k}_* \tag{15.47}$$

So in summary we have

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbb{E}[f_*], \text{var}[f_*]) \tag{15.48}$$

To convert this in to a predictive distribution for binary responses, we use

$$\pi_* = p(y_* = 1|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \approx \int \sigma(f_*) p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) df_* \tag{15.49}$$

This can be approximated using any of the methods discussed in Section 8.4.4, where we discussed Bayesian logistic regression. For example, using the probit approximation of Section 8.4.4.2, we have $\pi_* \approx \text{sigm}(\kappa(v)\mathbb{E}[f_*])$, where $v = \text{var}[f_*]$ and $\kappa^2(v) = (1 + \pi v/8)^{-1}$.

15.3.1.3 Computing the marginal likelihood

We need the marginal likelihood in order to optimize the kernel parameters. Using the Laplace approximation in Equation 8.54 we have

$$\log p(\mathbf{y}|\mathbf{X}) \approx \ell(\hat{\mathbf{f}}) - \frac{1}{2}\log|\mathbf{H}| + \text{const} \tag{15.50}$$

Hence

$$\log p(\mathbf{y}|\mathbf{X}) \approx \log p(\mathbf{y}|\hat{\mathbf{f}}) - \frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\log|\mathbf{K}| - \frac{1}{2}\log|\mathbf{K}^{-1} + \mathbf{W}| \tag{15.51}$$

Computing the derivatives $\frac{\partial \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})}{\partial \theta_j}$ is more complex than in the regression case, since $\hat{\mathbf{f}}$ and $\mathbf{W}$, as well as $\mathbf{K}$, depend on $\boldsymbol{\theta}$. Details can be found in (Rasmussen and Williams 2006, p125).

15.3.1.4 Numerically stable computation *

To implement the above equations in a numerically stable way, it is best to avoid inverting $\mathbf{K}$ or $\mathbf{W}$. (Rasmussen and Williams 2006, p45) suggest defining

$$\mathbf{B} = \mathbf{I}_N + \mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{W}^{\frac{1}{2}} \tag{15.52}$$

which has eigenvalues bounded below by 1 (because of the $\mathbf{I}$) and above by $1 + \frac{N}{4}\max_{ij} K_{ij}$ (because $w_{ii} = \pi_i(1-\pi) \leq 0.25$), and hence can be safely inverted.

One can use the matrix inversion lemma to show

$$(\mathbf{K}^{-1} + \mathbf{W})^{-1} = \mathbf{K} - \mathbf{K}\mathbf{W}^{\frac{1}{2}}\mathbf{B}^{-1}\mathbf{W}^{\frac{1}{2}}\mathbf{K} \tag{15.53}$$

Hence the IRLS update becomes

$$\mathbf{f}^{new} = (\mathbf{K}^{-1} + \mathbf{W})^{-1}\underbrace{(\mathbf{W}\mathbf{f} + \nabla \log p(\mathbf{y}|\mathbf{f}))}_{\mathbf{b}} \tag{15.54}$$

$$= \mathbf{K}(\mathbf{I} - \mathbf{W}^{\frac{1}{2}}\mathbf{B}^{-1}\mathbf{W}^{\frac{1}{2}}\mathbf{K})\mathbf{b} \tag{15.55}$$

$$= \mathbf{K}\underbrace{(\mathbf{b} - \mathbf{W}^{\frac{1}{2}}\mathbf{L}^T \setminus (\mathbf{L} \setminus (\mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{b})))}_{\mathbf{a}} \tag{15.56}$$

where $\mathbf{B} = \mathbf{L}\mathbf{L}^T$ is a Cholesky decomposition of $\mathbf{B}$. The fitting algorithm takes in $O(TN^3)$ time and $O(N^2)$ space, where T is the number of Newton iterations.

At convergence we have $\mathbf{a} = \mathbf{K}^{-1}\hat{\mathbf{f}}$, so we can evaluate the log marginal likelihood (Equation 15.51) using

$$\log p(\mathbf{y}|\mathbf{X}) = \log p(\mathbf{y}|\hat{\mathbf{f}}) - \frac{1}{2}\mathbf{a}^T\hat{\mathbf{f}} - \sum_i \log L_{ii} \tag{15.57}$$

where we exploited the fact that

$$|\mathbf{B}| = |\mathbf{K}||\mathbf{K}^{-1} + \mathbf{W}| = |\mathbf{I}_N + \mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{W}^{\frac{1}{2}}| \tag{15.58}$$

We now compute the predictive distribution. Rather than using $\mathbb{E}[f_*] = \mathbf{k}_*^T\mathbf{K}^{-1}\hat{\mathbf{f}}$, we exploit the fact that at the mode, $\nabla\ell = 0$, so $\hat{\mathbf{f}} = \mathbf{K}(\nabla \log p(\mathbf{y}|\hat{\mathbf{f}}))$. Hence we can rewrite the predictive mean as follows:[2]

$$\mathbb{E}[f_*] = \mathbf{k}_*^T \nabla \log p(\mathbf{y}|\hat{\mathbf{f}}) \tag{15.59}$$

To compute the predictive variance, we exploit the fact that

$$(\mathbf{K} + \mathbf{W}^{-1})^{-1} = \mathbf{W}^{\frac{1}{2}}\mathbf{W}^{-\frac{1}{2}}(\mathbf{K} + \mathbf{W}^{-1})^{-1}\mathbf{W}^{-\frac{1}{2}}\mathbf{W}^{\frac{1}{2}} = \mathbf{W}^{\frac{1}{2}}\mathbf{B}^{-1}\mathbf{W}^{\frac{1}{2}} \tag{15.60}$$

to get

$$\mathrm{var}[f_*] = k_{**} - \mathbf{k}_*^T\mathbf{W}^{\frac{1}{2}}(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{W}^{\frac{1}{2}}\mathbf{k}_* = k_{**} - \mathbf{v}^T\mathbf{v} \tag{15.61}$$

where $\mathbf{v} = \mathbf{L} \setminus (\mathbf{W}^{\frac{1}{2}}\mathbf{k}_*)$. We can then compute π_*.

The whole algorithm is summarized in Algorithm 16, based on (Rasmussen and Williams 2006, p46). Fitting takes $O(N^3)$ time, and prediction takes $O(N^2 N_*)$ time, where N_* is the number of test cases.

15.3.1.5 Example

In Figure 15.7, we show a synthetic binary classification problem in 2d. We use an SE kernel. On the left, we show predictions using hyper-parameters set by hand; we use a short length scale, hence the very sharp turns in the decision boundary. On the right, we show the predictions using the learned hyper-parameters; the model favors a more parsimonious explanation of the data.

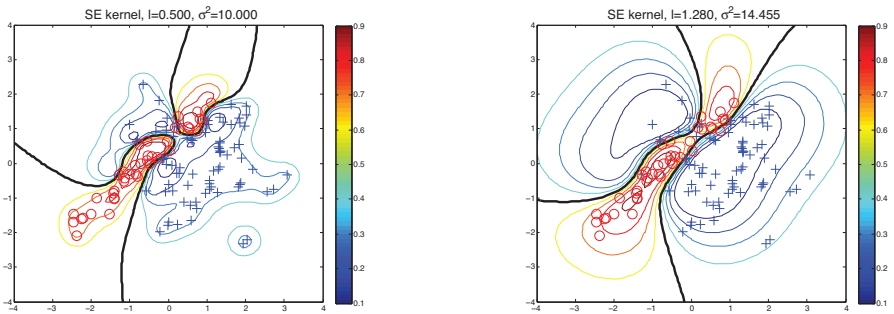
15.3.2 Multi-class classification

In this section, we consider a model of the form $p(y_i|\mathbf{x}_i) = \mathrm{Cat}(y_i|\mathcal{S}(\mathbf{f}_i))$, where $\mathbf{f}_i = (f_{i1}, \ldots, f_{iC})$, and we assume $f_{\cdot c} \sim \mathrm{GP}(0, \kappa_c)$. Thus we have one latent function per class, which are a priori independent, and which may use different kernels. As before, we will use a Gaussian approximation to the posterior. (A similar model, but using the multinomial probit function instead of the multinomial logit, is described in (Girolami and Rogers 2006).)

2. We see that training points that are well-predicted by the model, for which $\nabla_i \log p(y_i|f_i) \approx 0$, do not contribute strongly to the prediction at test points; this is similar to the behavior of support vectors in an SVM (see Section 14.5).

Algorithm 15.2: GP binary classification using Gaussian approximation

// First compute MAP estimate using IRLS;
$\mathbf{f} = \mathbf{0}$;
repeat
  $\mathbf{W} = -\nabla\nabla \log p(\mathbf{y}|\mathbf{f})$;
  $\mathbf{B} = \mathbf{I}_N + \mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{W}^{\frac{1}{2}}$;
  $\mathbf{L} = \text{cholesky}(\mathbf{B})$;
  $\mathbf{b} = \mathbf{W}\mathbf{f} + \nabla \log p(\mathbf{y}|\mathbf{f})$;
  $\mathbf{a} = \mathbf{b} - \mathbf{W}^{\frac{1}{2}}\mathbf{L}^T \setminus (\mathbf{L} \setminus (\mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{b}))$;
  $\mathbf{f} = \mathbf{K}\mathbf{a}$;
until *converged*;
$\log p(\mathbf{y}|\mathbf{X}) = \log p(\mathbf{y}|\mathbf{f}) - \frac{1}{2}\mathbf{a}^T\mathbf{f} - \sum_i \log L_{ii}$;
// Now perform prediction ;
$\mathbb{E}[f_*] = \mathbf{k}_*^T \nabla \log p(\mathbf{y}|\mathbf{f})$;
$\mathbf{v} = \mathbf{L} \setminus (\mathbf{W}^{\frac{1}{2}}\mathbf{k}_*)$;
$\text{var}[f_*] = k_{**} - \mathbf{v}^T\mathbf{v}$;
$p(y_* = 1) = \int \text{sigm}(z)\mathcal{N}(z|\mathbb{E}[f_*], \text{var}[f_*])dz$;



(a) (b)

Figure 15.7 Contours of the posterior predictive probability for the red circle class generated by a GP with an SE kernel. Thick black line is the decision boundary if we threshold at a probability of 0.5. (a) Manual parameters, short length scale. (b) Learned parameters, long length scale. Figure generated by `gpcDemo2d`, based on code by Carl Rasmussen.

15.3.2.1 Computing the posterior

The unnormalized log posterior is given by

$$\ell(\mathbf{f}) = -\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} + \mathbf{y}^T\mathbf{f} - \sum_{i=1}^{N}\log\left(\sum_{c=1}^{C}\exp f_{ic}\right) - \frac{1}{2}\log|\mathbf{K}| - \frac{CN}{2}\log 2\pi \tag{15.62}$$

where

$$\mathbf{f} = (f_{11},\ldots,f_{N1},f_{12},\ldots,f_{N2},\cdots,f_{1C},\ldots,f_{NC})^T \tag{15.63}$$

and $\mathbf{y}$ is a dummy encoding of the y_i's which has the same layout as $\mathbf{f}$. Also, $\mathbf{K}$ is a block diagonal matrix containing $\mathbf{K}_c$, where $\mathbf{K}_c = [\kappa_c(\mathbf{x}_i,\mathbf{x}_j)]$ models the correlation of the c'th latent function.

The gradient and Hessian are given by

$$\nabla\ell = -\mathbf{K}^{-1}\mathbf{f} + \mathbf{y} - \boldsymbol{\pi} \tag{15.64}$$

$$\nabla\nabla\ell = -\mathbf{K}^{-1} - \mathbf{W} \tag{15.65}$$

where $\mathbf{W} \triangleq \mathrm{diag}(\boldsymbol{\pi}) - \boldsymbol{\Pi}\boldsymbol{\Pi}^T$, where $\boldsymbol{\Pi}$ is a $CN \times N$ matrix obtained by stacking $\mathrm{diag}(\boldsymbol{\pi}_{:c})$ vertically. (Compare these expressions to standard logistic regression in Section 8.3.7.)

We can use IRLS to compute the mode. The Newton step has the form

$$\mathbf{f}^{new} = (\mathbf{K}^{-1} + \mathbf{W})^{-1}(\mathbf{W}\mathbf{f} + \mathbf{y} - \boldsymbol{\pi}) \tag{15.66}$$

Naively implementing this would take $O(C^3N^3)$ time. However, we can reduce this to $O(CN^3)$, as shown in (Rasmussen and Williams 2006, p52).

15.3.2.2 Computing the posterior predictive

We can compute the posterior predictive in a manner analogous to Section 15.3.1.2. For the mean of the latent response we have

$$\mathbb{E}[f_{*c}] = \mathbf{k}_c(\mathbf{x}_*)^T\mathbf{K}_c^{-1}\hat{\mathbf{f}}_c = \mathbf{k}_c(\mathbf{x}_*)^T(\mathbf{y}_c - \hat{\boldsymbol{\pi}}_c) \tag{15.67}$$

We can put this in vector form by writing

$$\mathbb{E}[\mathbf{f}_*] = \mathbf{Q}_* T(\mathbf{y} - \hat{\boldsymbol{\pi}}) \tag{15.68}$$

where

$$\mathbf{Q}_* = \begin{pmatrix} \mathbf{k}_1(\mathbf{x}_*) & \ldots & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & \ldots & \mathbf{k}_C(\mathbf{x}_*) \end{pmatrix} \tag{15.69}$$

Using a similar argument to Equation 15.47, we can show that the covariance of the latent response is given by

$$\mathrm{cov}[\mathbf{f}_*] = \boldsymbol{\Sigma} + \mathbf{Q}_*^T\mathbf{K}^{-1}(\mathbf{K}^{-1} + \mathbf{W})^{-1}\mathbf{K}^{-1}\mathbf{Q}_* \tag{15.70}$$

$$= \mathrm{diag}(\mathbf{k}(\mathbf{x}_*,\mathbf{x}_*)) - \mathbf{Q}_*^T(\mathbf{K} + \mathbf{W}^{-1})^{-1}\mathbf{Q}_* \tag{15.71}$$

where $\mathbf{\Sigma}$ is a $C \times C$ diagonal matrix with $\Sigma_{cc} = \kappa_c(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_c^T(\mathbf{x}_*)\mathbf{K}_c^{-1}\mathbf{k}_c(\mathbf{x}_*)$, and $\mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) = [\kappa_c(\mathbf{x}_*, \mathbf{x}_*)]$.

To compute the posterior predictive for the visible response, we need to use

$$p(y|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \approx \int \mathrm{Cat}(y|\mathcal{S}(\mathbf{f}_*))\mathcal{N}(\mathbf{f}_*|\mathbb{E}[\mathbf{f}_*], \mathrm{cov}[\mathbf{f}_*])d\mathbf{f}_* \tag{15.72}$$

We can use any of deterministic approximations to the softmax function discussed in Section 21.8.1.1 to compute this. Alternatively, we can just use Monte Carlo.

15.3.2.3 Computing the marginal likelihood

Using arguments similar to the binary case, we can show that

$$\log p(\mathbf{y}|\mathbf{X}) \approx -\frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}^{-1}\hat{\mathbf{f}} + \mathbf{y}^T\hat{\mathbf{f}} - \sum_{i=1}^{N}\log\left(\sum_{c=1}^{C}\exp\hat{f}_{ic}\right) - \frac{1}{2}\log|\mathbf{I}_{C_N} + \mathbf{W}^{\frac{1}{2}}\mathbf{K}\mathbf{W}^{\frac{1}{2}}| \tag{15.73}$$

This can be optimized numerically in the usual way.

15.3.2.4 Numerical and computational issues

One can implement model fitting in $O(TCN^3)$ time and $O(CN^2)$ space, where T is the number of Newton iterations, using the techniques described in (Rasmussen and Williams 2006, p50). Prediction takes $O(CN^3 + CN^2N_*)$ time, where N_* is the number of test cases.

15.3.3 GPs for Poisson regression

In this section, we illustrate GPs for Poisson regression. An interesting application of this is to spatial **disease mapping**. For example, (Vanhatalo et al. 2010) discuss the problem of modeling the relative risk of heart attack in different regions in Finland. The data consists of the heart attacks in Finland from 1996-2000 aggregated into 20km x 20km lattice cells. The model has the following form:

$$y_i \sim \mathrm{Poi}(e_i r_i) \tag{15.74}$$

where e_i is the known expected number of deaths (related to the population of cell i and the overall death rate), and r_i is the **relative risk** of cell i which we want to infer. Since the data counts are small, we regularize the problem by sharing information with spatial neighbors. Hence we assume $f \triangleq \log(r) \sim \mathrm{GP}(0, \kappa)$, where we use a Matern kernel with $\nu = 3/2$, and a length scale and magnitude that are estimated from data.

Figure 15.8 gives an example of the kind of output one can obtain from this method, based on data from 911 locations. On the left we plot the posterior mean relative risk (RR), and on the right, the posterior variance. We see that the RR is higher in Eastern Finland, which is consistent with other studies. We also see that the variance in the North is higher, since there are fewer people living there.

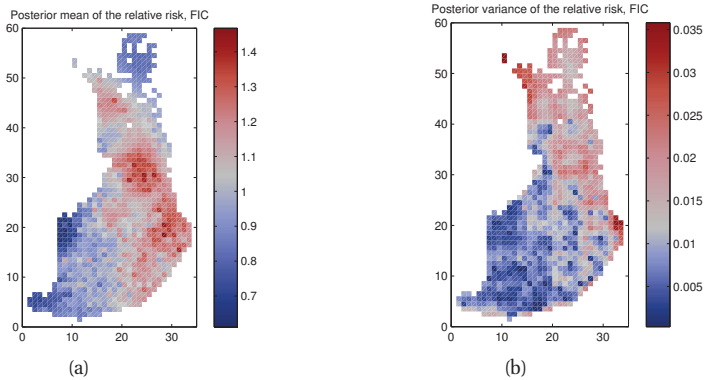


Figure 15.8 We show the relative risk of heart disease in Finland using a Poisson GP. Left: posterior mean. Right: posterior variance. Figure generated by `gpSpatialDemoLaplace`, written by Jarno Vanhatalo.

15.4 Connection with other methods

There are variety of other methods in statistics and machine learning that are closely related to GP regression/ classification. We give a brief review of some of these below.

15.4.1 Linear models compared to GPs

Consider Bayesian linear regression for D-dimensional features, where the prior on the weights is $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. The posterior predictive distribution is given by the following;

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu, \sigma^2) \tag{15.75}$$

$$\mu = \frac{1}{\sigma_y^2}\mathbf{x}_*^T\mathbf{A}^{-1}\mathbf{X}^T\mathbf{y} \tag{15.76}$$

$$\sigma^2 = \mathbf{x}_*^T\mathbf{A}^{-1}\mathbf{x}_* \tag{15.77}$$

where $\mathbf{A} = \sigma_y^{-2}\mathbf{X}^T\mathbf{X} + \mathbf{\Sigma}^{-1}$. One can show that we can rewrite the above distribution as follows

$$\mu = \mathbf{x}_*^T\mathbf{\Sigma}\mathbf{X}^T(\mathbf{K} + \sigma_y^2\mathbf{I})^{-1}\mathbf{y} \tag{15.78}$$

$$\sigma^2 = \mathbf{x}_*^T\mathbf{\Sigma}\mathbf{x}_* - \mathbf{x}_*^T\mathbf{\Sigma}\mathbf{X}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{X}\mathbf{\Sigma}\mathbf{x}_* \tag{15.79}$$

where we have defined $\mathbf{K} = \mathbf{X}\mathbf{\Sigma}\mathbf{X}^T$, which is of size $N \times N$. Since the features only ever appear in the form $\mathbf{X}\mathbf{\Sigma}\mathbf{X}^T$, $\mathbf{x}_*^T\mathbf{\Sigma}\mathbf{X}^T$ or $\mathbf{x}_*^T\mathbf{\Sigma}\mathbf{x}_*$, we can kernelize the above expression by defining $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{\Sigma}\mathbf{x}'$.

Thus we see that Bayesian linear regression is equivalent to a GP with covariance function $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{\Sigma}\mathbf{x}'$. Note, however, that this is a **degenerate** covariance function, since it has at most D non-zero eigenvalues. Intuitively this reflects the fact that the model can only represent a limited number of functions. This can result in underfitting, since the model is not flexible enough to capture the data. What is perhaps worse, it can result in overconfidence, since the

model's prior is so impoverished that its posterior will become too concentrated. So not only is the model wrong, it think it's right!

15.4.2 Linear smoothers compared to GPs

A **linear smoother** is a regression function which is a linear function of the training outputs:

$$\hat{f}(\mathbf{x}_*) = \sum_i w_i(\mathbf{x}_*)\ y_i \tag{15.80}$$

where $w_i(\mathbf{x}_*)$ is called the **weight function** (Silverman 1984). (Do not confuse this with a linear model, where the output is a linear function of the input vector.)

There are a variety of linear smoothers, such as kernel regression (Section 14.7.4), locally weighted regression (Section 14.7.5), smoothing splines (Section 15.4.6), and GP regression. To see that GP regession is a linear smoother, note that the mean of the posterior predictive distribution of a GP is given by

$$\overline{f}(\mathbf{x}_*) = \mathbf{k}_*^T(\mathbf{K} + \sigma_y^2\mathbf{I}_N)^{-1}\mathbf{y} = \sum_{i=1}^N y_i w_i(\mathbf{x}_*) \tag{15.81}$$

where $w_i(\mathbf{x}_*) = [(\mathbf{K} + \sigma_y^2\mathbf{I}_N)^{-1}\mathbf{k}_*]_i$.

In kernel regression, we derive the weight function from a smoothing kernel rather than a Mercer kernel, so it is clear that the weight function will then have local support. In the case of a GP, things are not as clear, since the weight function depends on the inverse of $\mathbf{K}$. For certain GP kernel functions, we can analytically derive the form of $w_i(\mathbf{x})$; this is known as the **equivalent kernel** (Silverman 1984). One can show that $\sum_{i=1}^N w_i(\mathbf{x}_*) = 1$, although we may have $w_i(\mathbf{x}_*) < 0$, so we are computing a linear combination but not a convex combination of the y_i's. More interestingly, $w_i(\mathbf{x}_*)$ is a local function, even if the original kernel used by the GP is not local. Futhermore the effective bandwidth of the equivalent kernel of a GP automatically decreases as the sample size N increases, whereas in kernel smoothing, the bandwidth h needs to be set by hand to adapt to N. See e.g., (Rasmussen and Williams 2006, Sec 2.6,Sec 7.1) for details.

15.4.2.1 Degrees of freedom of linear smoothers

It is clear why this method is called "linear", but why is it called a "smoother"? This is best explained in terms of GPs. Consider the prediction on the training set:

$$\overline{\mathbf{f}} = \mathbf{K}(\mathbf{K} + \sigma_y^2)^{-1}\mathbf{y} \tag{15.82}$$

Now let $\mathbf{K}$ have the eigendecomposition $\mathbf{K} = \sum_{i=1}^N \lambda_i \mathbf{u}_i \mathbf{u}_i^T$. Since $\mathbf{K}$ is real and symmetric positive definite, the eigenvalues λ_i are real and non-negative, and the eigenvectors $\mathbf{u}_i$ are orthonormal. Now let $\mathbf{y} = \sum_{i=1}^N \gamma_i \mathbf{u}_i$, where $\gamma_i = \mathbf{u}_i^T\mathbf{y}$. Then we can rewrite the above equation as follows:

$$\overline{\mathbf{f}} = \sum_{i=1}^N \frac{\gamma_i \lambda_i}{\lambda_i + \sigma_y^2}\mathbf{u}_i \tag{15.83}$$

This is the same as Equation 7.47, except we are working with the eigenvectors of the Gram matrix $\mathbf{K}$ instead of the data matrix $\mathbf{X}$. In any case, the interpretation is similar: if $\frac{\lambda_i}{\lambda_i+\sigma_y^2} \ll 1$, then the corresponding basis function $\mathbf{u}_i$ will not have much influence. Consequently the high-frequency components in $\mathbf{y}$ are smoothed out. The effective **degrees of freedom** of the linear smoother is defined as

$$\text{dof} \triangleq \text{tr}(\mathbf{K}(\mathbf{K}+\sigma_y^2\mathbf{I})^{-1}) = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i+\sigma_y^2} \tag{15.84}$$

This specifies how "wiggly" the curve is.

15.4.3 SVMs compared to GPs

We saw in Section 14.5.2 that the SVM objective for binary classification is given by Equation 14.57

$$J(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(1-y_i f_i)_+ \tag{15.85}$$

We also know from Equation 14.59 that the optimal solution has the form $\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$, so $||\mathbf{w}||^2 = \sum_{i,j} \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$. Kernelizing we get $||\mathbf{w}||^2 = \boldsymbol{\alpha}\mathbf{K}\boldsymbol{\alpha}$. From Equation 14.61, and absorbing the $\hat{w}_0$ term into one of the kernels, we have $\mathbf{f} = \mathbf{K}\boldsymbol{\alpha}$, so $||\mathbf{w}||^2 = \mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}$. Hence the SVM objective can be rewritten as

$$J(\mathbf{f}) = \frac{1}{2}\mathbf{f}^T\mathbf{f} + C\sum_{i=1}^{N}(1-y_i f_i)_+ \tag{15.86}$$

Compare this to MAP estimation for GP classifier:

$$J(\mathbf{f}) = \frac{1}{2}\mathbf{f}^T\mathbf{f} - \sum_{i=1}^{N}\log p(y_i|f_i) \tag{15.87}$$

It is tempting to think that we can "convert" an SVM into a GP by figuring out what likelihood would be equivalent to the hinge loss. However, it turns out there is no such likelihood (Sollich 2002), although there is a pseudo-likelihood that matches the SVM (see Section 14.5.5).

From Figure 6.7 we saw that the hinge loss and the logistic loss (as well as the probit loss) are quite similar to each other. The main difference is that the hinge loss is strictly 0 for errors larger than 1. This gives rise to a sparse solution. In Section 14.3.2, we discussed other ways to derive sparse kernel machines. We discuss the connection between these methods and GPs below.

15.4.4 LIVM and RVMs compared to GPs

Sparse kernel machines are just linear models with basis function expansion of the form $\boldsymbol{\phi}(\mathbf{x}) = [\kappa(\mathbf{x},\mathbf{x}_1),\ldots,\kappa(\mathbf{x},\mathbf{x}_N)]$. From Section 15.4.1, we know that this is equivalent to a GP with the following kernel:

$$\kappa(\mathbf{x},\mathbf{x}') = \sum_{j=1}^{D}\frac{1}{\alpha_j}\phi_j(\mathbf{x})\phi_j(\mathbf{x}') \tag{15.88}$$

where $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \text{diag}(\alpha_j^{-1}))$. This kernel function has two interesting properties. First, it is degenerate, meaning it has at most N non-zero eigenvalues, so the joint distribution $p(\mathbf{f}, \mathbf{f}_*)$ will be highly constrained. Second, the kernel depends on the training data. This can cause the model to be overconfident when extrapolating beyond the training data. To see this, consider a point $\mathbf{x}_*$ far outside the convex hull of the data. All the basis functions will have values close to 0, so the prediction will back off to the mean of the GP. More worryingly, the variance will back off to the noise variance. By contrast, when using a non-degenerate kernel function, the predictive variance increases as we move away from the training data, as desired. See (Rasmussen and Quiñonero-Candela 2005) for further discussion.

15.4.5 Neural networks compared to GPs

In Section 16.5, we will discuss neural networks, which are a nonlinear generalization of GLMs. In the binary classification case, a neural network is defined by a logistic regression model applied to a logistic regression model:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Ber}\left(y|\text{sigm}\left(\mathbf{w}^T \text{sigm}(\mathbf{V}\mathbf{x})\right)\right) \tag{15.89}$$

It turns out there is an interesting connection between neural networks and Gaussian processes, as first pointed out by (Neal 1996).

To explain the connection, we follow the presentation of (Rasmussen and Williams 2006, p91). Consider a neural network for regression with one hidden layer. This has the form

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|f(\mathbf{x}; \boldsymbol{\theta}), \sigma^2) \tag{15.90}$$

where

$$f(\mathbf{x}) = b + \sum_{j=1}^{H} v_j g(\mathbf{x}; \mathbf{u}_j) \tag{15.91}$$

where b is the offset of bias term, v_j is the output weight from hidden unit j to the response y, $\mathbf{u}_j$ are the inputs weights to unit j from the input $\mathbf{x}$, and $g()$ is the hidden unit activation function. This is typically the sigmoid or tanh function, but can be any smooth function.

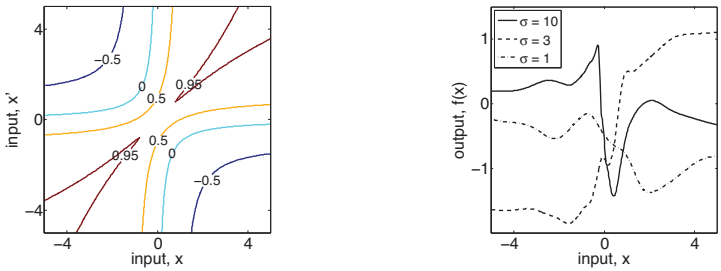
Let us use the following priors on the weights: where $b \sim \mathcal{N}(0, \sigma_b^2)$ $\mathbf{v} \sim \prod_j \mathcal{N}(v_j|0, \sigma_w^2)$, $\mathbf{u} \sim \prod_j p(\mathbf{u}_j)$ for some unspecified $p(\mathbf{u}_j)$. Denoting all the weights by $\boldsymbol{\theta}$ we have

$$\mathbb{E}_{\boldsymbol{\theta}}[f(\mathbf{x})] = 0 \tag{15.92}$$

$$\mathbb{E}_{\boldsymbol{\theta}}[f(\mathbf{x})f(\mathbf{x}')] = \sigma_b^2 + \sum_j \sigma_v^2 \mathbb{E}_{\mathbf{v}}[g(\mathbf{x}; \mathbf{u}_j)g(\mathbf{x}'; \mathbf{u}_j)] \tag{15.93}$$

$$= \sigma_b^2 + H\sigma_v^2 \mathbb{E}_{\mathbf{u}}[g(\mathbf{x}; \mathbf{u})g(\mathbf{x}'; \mathbf{u})] \tag{15.94}$$

where the last equality follows since the H hidden units are iid. If we let σ_v^2 scale as ω^2/H (since more hidden units will increase the input to the final node, so we should scale down the magnitude of the weights), then the last term becomes $\omega^2 \mathbb{E}_{\mathbf{u}}[g(\mathbf{x}; \mathbf{u})g(\mathbf{x}'; \mathbf{u})]$. This is a sum over H iid random variables. Assuming that g is bounded, we can apply the central limit theorem. The result is that as $H \to \infty$, we get a Gaussian process.



(a) (b)

Figure 15.9 (a) Covariance function $\kappa_{NN}(x, x')$ for $\sigma_0 = 10$, $\sigma = 10$. (b) Samples from from a GP with this kernel, using various values of σ. Figure generated by `gpnnDemo`, written by Chris Williams.

If we use as activation / transfer function $g(\mathbf{x}; \mathbf{u}) = \text{erf}(u_0 + \sum_{j=1}^{D} u_j x_j)$, where $\text{erf}(z) = 2/\sqrt{\pi} \int_0^z e^{-t^2} dt$, and we choose $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$, then (Williams 1998) showed that the covariance kernel has the form

$$\kappa_{NN}(\mathbf{x}, \mathbf{x}') = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^T \mathbf{\Sigma} \tilde{\mathbf{x}}'}{\sqrt{(1 + 2\tilde{\mathbf{x}}^T \mathbf{\Sigma} \tilde{\mathbf{x}})(1 + 2(\tilde{\mathbf{x}}')^T \mathbf{\Sigma} \tilde{\mathbf{x}}')}} \right) \tag{15.95}$$

where $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_D)$. This is a true "neural network" kernel, unlike the "sigmoid" kernel $\kappa(\mathbf{x}, \mathbf{x}') = \tanh(a + b\mathbf{x}^T \mathbf{x}')$, which is not positive definite.

Figure 15.9(a) illustrates this kernel when $D = 2$ and $\mathbf{\Sigma} = \text{diag}(\sigma_0^2, \sigma^2)$. Figure 15.9(b) shows some functions sampled from the corresponding GP. These are equivalent to functions which are superpositions of $\text{erf}(u_0 + ux)$ where u_0 and u are random. As σ^2 increases, the variance of u increases, so the function varies more quickly. Unlike the RBF kernel, functions sampled from this kernel do not tend to 0 away from the data, but rather they tend to remain at the same value they had at the "edge" of the data.

Now suppose we use an RBF network, which is equivalent to a hidden unit activation function of the form $g(\mathbf{x}; \mathbf{u}) = \exp(-|\mathbf{x} - \mathbf{u}|^2/(2\sigma_g^2))$. If $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \sigma_u^2 \mathbf{I})$, one can show that the coresponding kernel is equivalent to the RBF or SE kernel.

15.4.6 Smoothing splines compared to GPs *

Smoothing splines are a widely used non-parametric method for smoothly interpolating data (Green and Silverman 1994). They are are a special case of GPs, as we will see. They are usually used when the input is 1 or 2 dimensional.

15.4.6.1 Univariate splines

The basic idea is to fit a function f by minimizing the discrepancy to the data plus a smoothing term that penalizes functions that are "too wiggly". If we penalize the m'th derivative of the

function, the objective becomes

$$J(f) = \sum_{i=1}^{N}(f(x_i) - y_i)^2 + \lambda \int (\frac{d^m}{dx^m} f(x))^2 dx \tag{15.96}$$

One can show (Green and Silverman 1994) that the solution is a **piecewise polynomial** where the polynomials have order $2m-1$ in the interior bins $[x_{i-1}, x_i]$ (denoted $\mathcal{I}$), and order $m-1$ in the two outermost intervals $(-\infty, x_1]$ and $[x_N, \infty)$:

$$f(x) = \sum_{j=0}^{m-1} \beta_j x^j + \mathbb{I}(x \in \mathcal{I}) \left(\sum_{i=1}^{N} \alpha_i (x - x_i)_+^{2m-1} \right) + \mathbb{I}(x \notin \mathcal{I}) \left(\sum_{i=1}^{N} \alpha_i (x - x_i)_+^{m-1} \right) \tag{15.97}$$

For example, if $m = 2$, we get the (natural) **cubic spline**

$$f(x) = \beta_0 + \beta_1 x + \mathbb{I}(x \in \mathcal{I}) \left(\sum_{i=1}^{N} \alpha_i (x - x_i)_+^3 \right) + \mathbb{I}(x \notin \mathcal{I}) \left(\sum_{i=1}^{N} \alpha_i (x - x_i)_+ \right) \tag{15.98}$$

which is a series of truncated cubic polynomials, whose left hand sides are located at each of the N training points. (The fact that the model is linear on the edges prevents it from extrapolating too wildly beyond the range of the data; if we drop this requirement, we get an "unrestricted" spline.)

We can clearly fit this model using ridge regression: $\hat{\mathbf{w}} = (\mathbf{\Phi}^T \mathbf{\Phi} + \lambda \mathbf{I}_N)^{-1} \mathbf{\Phi}^T \mathbf{y}$, where the columns of $\mathbf{\Phi}$ are 1, x_i and $(x - x_i)_+^3$ for $i = 2 : N-1$ and $(x - x_i)_+$ for $i = 1$ or $i = N$. However, we can also derive an $O(N)$ time method (Green and Silverman 1994, Sec 2.3.3).

15.4.6.2 Regression splines

In general, we can place the polynomials at a fixed set of K locations known as **knots**, denoted ξ_k. The result is called a **regression spline**. This is a parametric model, which uses basis function expansion of the following form (where we drop the interior/ exterior distinction for simplicity):

$$f(x) = \beta_0 + \beta_1 x + \sum_{k=1}^{K} \alpha_j (x - \xi_k)_+^3 \tag{15.99}$$

Choosing the number and locations of the knots is just like choosing the number and values of the support vectors in Section 14.3.2. If we impose an ℓ_2 regularizer on the regression coefficients α_j, the method is known as **penalized splines**. See Section 9.6.1 for a practical example of penalized splines.

15.4.6.3 The connection with GPs

One can show (Rasmussen and Williams 2006, p139) that the cubic spline is the MAP estimate of the following function

$$f(x) = \beta_0 + \beta_1 x + r(x) \tag{15.100}$$

where $p(\beta_j) \propto 1$ (so that we don't penalize the zero'th and first derivatives of f), and $r(x) \sim \text{GP}(0, \sigma_f^2 \kappa_{sp}(x, x'))$, where

$$\kappa_{sp}(x, x') \triangleq \int_0^1 (x - u)_+ (x' - u)_+ du \tag{15.101}$$

Note that the kernel in Equation 15.101 is rather unnatural, and indeed posterior samples from the resulting GP are rather unsmooth. However, the posterior mode/mean is smooth. This shows that regularizers don't always make good priors.

15.4.6.4 2d input (thin-plate splines)

One can generalize cubic splines to 2d input by defining a regularizer of the following form:

$$\int\int \left[\left(\frac{\partial^2 f(x)}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f(x)}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 f(x)}{\partial x_2^2} \right)^2 \right] dx_1 dx_2 \tag{15.102}$$

One can show that the solution has the form

$$f(x) = \beta_0 + \boldsymbol{\beta}_1^T \mathbf{x} + \sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}_i(\mathbf{x}) \tag{15.103}$$

where $\boldsymbol{\phi}_i(\mathbf{x}) = \eta(||\mathbf{x} - \mathbf{x}_i||)$, and $\eta(z) = z^2 \log z^2$. This is known as a **thin plate spline**. This is equivalent to MAP estimation with a GP whose kernel is defined in (Williams and Fitzgibbon 2006).

15.4.6.5 Higher-dimensional inputs

It is hard to analytically solve for the form of the optimal solution when using higher-order inputs. However, in the parametric regression spline setting, where we forego the regularizer on f, we have more freedom in defining our basis functions. One way to handle multiple inputs is to use a **tensor product basis**, defined as the cross product of 1d basis functions. For example, for 2d input, we can define

$$f(x_1, x_2) = \beta_0 + \sum_m \beta_{1m}(x_1 - \xi_{1m})_+ + \sum_m \beta_{2m}(x_2 - \xi_{2m})_+ \tag{15.104}$$

$$+ \sum_m \beta_{12m}(x_1 - \xi_{1m})_+ (x_2 - \xi_{2m})_+ \tag{15.105}$$

It is clear that for high-dimensional data, we cannot allow higher-order interactions, because there will be too many parameters to fit. One approach to this problem is to use a search procedure to look for useful interaction terms. This is known as MARS, which stands for "multivariate adaptive regression splines". See Section 16.3.3 for details.

15.4.7 RKHS methods compared to GPs *

We can generalize the idea of penalizing derivatives of functions, as used in smoothing splines, to fit functions with a more general notion of smoothness. Recall from Section 14.2.3 that

Mercer's theorem says that any positive definite kernel function can be represented in terms of eigenfunctions:

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}') \tag{15.106}$$

The ϕ_i form an orthormal basis for a function space:

$$\mathcal{H}_k = \{f : f(\mathbf{x}) = \sum_{i=1}^{\infty} f_i \phi_i(\mathbf{x}),\ \sum_{i=1}^{\infty} f_i^2/\lambda_i < \infty\} \tag{15.107}$$

Now define the inner product between two functions $f(\mathbf{x}) = \sum_{i=1}^{\infty} f_i \phi_i(\mathbf{x})$ and $g(\mathbf{x}) = \sum_{i=1}^{\infty} g_i \phi_i(\mathbf{x})$ in this space as follows:

$$\langle f, g \rangle_{\mathcal{H}} \triangleq \sum_{i=1}^{\infty} \frac{f_i g_i}{\lambda_i} \tag{15.108}$$

In Exercise 15.1, we show that this definition implies that

$$\langle \kappa(\mathbf{x}_1, \cdot), \kappa(\mathbf{x}_2, \cdot) \rangle_{\mathcal{H}} = \kappa(\mathbf{x}_1, \mathbf{x}_2) \tag{15.109}$$

This is called the **reproducing property**, and the space of functions $\mathcal{H}_k$ is called a **reproducing kernel Hilbert space** or **RKHS**.

Now consider an optimization problem of the form

$$J(f) = \frac{1}{2\sigma_y^2} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 + \frac{1}{2} ||f||_H^2 \tag{15.110}$$

where $||f||_J$ is the **norm of a function**:

$$||f||_H = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{f_i^2}{\lambda_i} \tag{15.111}$$

The intuition is that functions that are complex wrt the kernel will have large norms, because they will need many eigenfunctions to represent them. We want to pick a simple function that provides a good fit to the data.

One can show (see e.g., (Schoelkopf and Smola 2002)) that the solution must have the form

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i) \tag{15.112}$$

This is known as the **representer theorem**, and holds for other convex loss functions besides squared error.

We can solve for the $\boldsymbol{\alpha}$ by substituting in $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$ and using the reproducing property to get

$$J(\boldsymbol{\alpha}) = \frac{1}{2\sigma_y^2} |\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}|^2 + \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \tag{15.113}$$

Minimizing wrt $\boldsymbol{\alpha}$ we find

$$\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \tag{15.114}$$

and hence

$$\hat{f}(\mathbf{x}_*) = \sum_i \hat{\alpha}_i \kappa(\mathbf{x}_*, \mathbf{x}_i) = \mathbf{k}_*^T (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} \tag{15.115}$$

This is identical to Equation 15.18, the posterior mean of a GP predictive distribution. Indeed, since the mean and mode of a Gaussian are the same, we can see that linear regresson with an RKHS regularizer is equivalent to MAP estimation with a GP. An analogous statement holds for the GP logistic regression case, which also uses a convex likelihood / loss function.

15.5 GP latent variable model

In Section 14.4.4, we discussed kernel PCA, which applies the kernel trick to regular PCA. In this section, we discuss a different way to combine kernels with probabilistic PCA. The resulting method is known as the **GP-LVM**, which stands for "Gaussian process latent variable model" (Lawrence 2005).

To explain the method, we start with PPCA. Recall from Section 12.2.4 that the PPCA model is as follows:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i | \mathbf{0}, \mathbf{I}) \tag{15.116}$$

$$p(\mathbf{y}_i | \mathbf{z}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_i | \mathbf{W}\mathbf{z}_i, \sigma^2 \mathbf{I}) \tag{15.117}$$

We can fit this model by maximum likelihood, by integrating out the $\mathbf{z}_i$ and maximizing $\mathbf{W}$ (and σ^2). The objective is given by

$$p(\mathbf{Y} | \mathbf{W}, \sigma^2) = (2\pi)^{-DN/2} |\mathbf{C}|^{-N/2} \exp\left(-\frac{1}{2} \mathrm{tr}(\mathbf{C}^{-1} \mathbf{Y}^T \mathbf{Y}) \right) \tag{15.118}$$

where $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}$. As we showed in Theorem 12.2.2, the MLE for this can be computed in terms of the eigenvectors of $\mathbf{Y}^T \mathbf{Y}$.

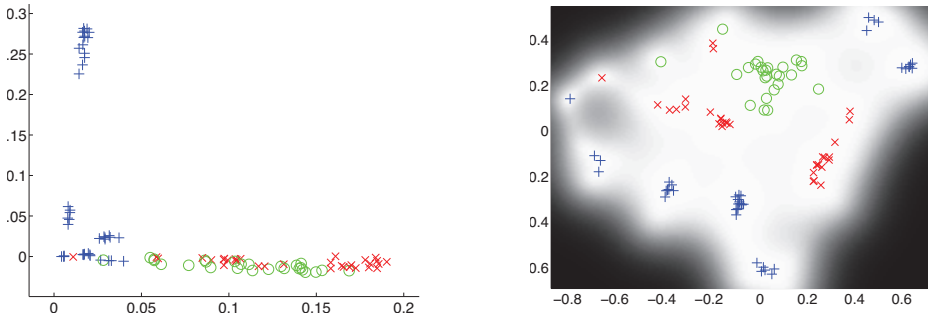
Now we consider the dual problem, whereby we maximize $\mathbf{Z}$ and integrate out $\mathbf{W}$. We will use a prior of the form $p(\mathbf{W}) = \prod_j \mathcal{N}(\mathbf{w}_j | \mathbf{0}, \mathbf{I})$. The corresponding likelihood becomes

$$p(\mathbf{Y} | \mathbf{Z}, \sigma^2) = \prod_{d=1}^{D} \mathcal{N}(\mathbf{y}_{:,d} | \mathbf{0}, \mathbf{Z}\mathbf{Z}^T + \sigma^2 \mathbf{I}) \tag{15.119}$$

$$= (2\pi)^{-DN/2} |\mathbf{K}_z|^{-D/2} \exp\left(-\frac{1}{2} \mathrm{tr}(\mathbf{K}_z^{-1} \mathbf{Y}\mathbf{Y}^T) \right) \tag{15.120}$$

where $\mathbf{K}_z = \mathbf{Z}\mathbf{Z}^T + \sigma^2 \mathbf{I}$. Based on our discussion of the connection between the eigenvalues of $\mathbf{Y}\mathbf{Y}^T$ and of $\mathbf{Y}^T \mathbf{Y}$ in Section 14.4.4, it should come as no surprise that we can also solve the dual problem using eigenvalue methods (see (Lawrence 2005) for the details).

If we use a linear kernel, we recover PCA. But we can also use a more general kernel: $\mathbf{K}_z = \mathbf{K} + \sigma^2 \mathbf{I}$, where $\mathbf{K}$ is the Gram matrix for $\mathbf{Z}$. The MLE for $\hat{\mathbf{Z}}$ will no longer be available



(a) (b)

Figure 15.10 2d representation of 12 dimensional oil flow data. The different colors/symbols represent the 3 phases of oil flow. (a) Kernel PCA with Gaussian kernel. (b) GP-LVM with Gaussian kernel. The shading represents the precision of the posterior, where lighter pixels have higher precision. From Figure 1 of (Lawrence 2005). Used with kind permission of Neil Lawrence.

via eigenvalue methods; instead we must use gradient-based optimization. The objective is given by

$$\ell = -\frac{D}{2}\log|\mathbf{K}_z| - \frac{1}{2}\mathrm{tr}(\mathbf{K}_z^{-1}\mathbf{Y}\mathbf{Y}^T) \tag{15.121}$$

and the gradient is given by

$$\frac{\partial \ell}{\partial Z_{ij}} = \frac{\partial \ell}{\partial \mathbf{K}_z}\frac{\partial \mathbf{K}_z}{\partial Z_{ij}} \tag{15.122}$$

where

$$\frac{\partial \ell}{\partial \mathbf{K}_z} = \mathbf{K}_z^{-1}\mathbf{Y}\mathbf{Y}^T\mathbf{K}_z^{-1} - D\mathbf{K}_z^{-1} \tag{15.123}$$

The form of $\frac{\partial \mathbf{K}_z}{\partial Z_{ij}}$ will of course depend on the kernel used. (For example, with a linear kernel, where $\mathbf{K}_z = \mathbf{Z}\mathbf{Z}^T + \sigma^2\mathbf{I}$, we have $\frac{\partial \mathbf{K}_z}{\partial \mathbf{Z}} = \mathbf{Z}$.) We can then pass this gradient to any standard optimizer, such as conjugate gradient descent.

Let us now compare GP-LVM to kernel PCA. In kPCA, we learn a kernelized mapping from the observed space to the latent space, whereas in GP-LVM, we learn a kernelized mapping from the latent space to the observed space. Figure 15.10 illustrates the results of applying kPCA and GP-LVM to visualize the 12 dimensional oil flow data shown in In Figure 14.9(a). We see that the embedding produced by GP-LVM is far better. If we perform nearest neighbor classification in the latent space, GP-LVM makes 4 errors, while kernel PCA (with the same kernel but separately optimized hyper-parameters) makes 13 errors, and regular PCA makes 20 errors.

GP-LVM inherits the usual advantages of probabilistic generative models, such as the ability to handle missing data and data of different types, the ability to use gradient-based methods (instead of grid search) to tune the kernel parameters, the ability to handle prior information,

etc. For a discussion of some other probabilistic methods for (spectral) dimensionality reduction, see (Lawrence 2012).

15.6 Approximation methods for large datasets

The principal drawback of GPs is that they take $O(N^3)$ time to use. This is because of the need to invert (or compute the Cholesky decomposition of) the $N \times N$ kernel matrix $\mathbf{K}$. A variety of approximation methods have been devised which take $O(M^2 N)$ time, where M is a user-specifiable parameter. For details, see (Quinonero-Candela et al. 2007).

Exercises

Exercise 15.1 Reproducing property

Prove Equation 15.109.