Gaussian processes

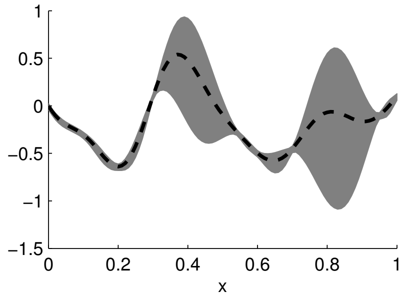
Gaussian processes (GPs) are distributions over functions from an input $\mathbf{x}$, which could be high-dimensional and could be a complex object like a string or graph, to a scalar or 1-dimensional output $f(\mathbf{x})$. We will use a Gaussian process prior over functions in a Bayesian approach to regression. Gaussian processes can also be used as a part of larger models.

Goals: 1) model complex functions: not just straight lines/planes, or a combination of a fixed number of basis functions. 2) Represent our uncertainty about the function, for example with error bars. 3) Do as much of the mathematics as possible analytically.

1 The value of uncertainty

Using Gaussian processes, like Bayesian linear regression, we can compute our uncertainty about a function given data. Error bars are useful when making decisions, or optimizing expensive functions. Examples of expensive functions include 'efficacy of a drug', or 'efficacy of a training procedure for a large-scale neural network'.

In the cartoon below, a black dashed line shows an estimate of a function, with a gray region indicating uncertainty. The function could be performance of a system as a function of its settings. Often the performance of a system doesn't vary up and down rapidly as a function of one setting, but in high-dimensions there could easily be multiple modes — different combinations of inputs that work quite well.



The largest value appears to be around $x = 0.4$. If we can afford to run a couple of experiments, we might also test the system at $x = 0.8$, which might actually perform better. According to our point estimates, the performance at $x = 0$ is similar to $x = 0.8$. However, we aren't uncertain at $x = 0$, so it isn't worth running experiments there.

Using uncertainty in a probabilistic model to guide search is called *Bayesian optimization*. We could do Bayesian optimization with Bayesian linear regression. Although we would have to be careful: Bayesian linear regression is often too certain if the model is too simple (think of our 'underfitting' example in Bayesian linear regression), or doesn't include enough basis functions.

2 The multivariate Gaussian distribution

We'll see that Gaussian processes are really just high-dimensional multivariate Gaussian distributions. This section has a quick review of some of the things we can do with Gaussians. It's perhaps surprising that these manipulations can answer interesting statistical and machine learning questions!

A Gaussian distribution is completely described by its parameters $\boldsymbol{\mu}$ and Σ:

$$p(\mathbf{f} \,|\, \Sigma, \boldsymbol{\mu}) \;=\; |2\pi\Sigma|^{-\frac{1}{2}} \exp\big(-\tfrac{1}{2}(\mathbf{f}-\boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{f}-\boldsymbol{\mu})\big), \qquad (1)$$

where $\boldsymbol{\mu}$ and Σ are the mean and covariance:

$$\mu_i = \mathbb{E}[f_i] \tag{2}$$

$$\Sigma_{ij} = \mathbb{E}[f_i f_j] - \mu_i \mu_j. \tag{3}$$

The covariance matrix Σ must be positive definite. (Or positive semi-definite if we allow some of the elements of $\mathbf{f}$ to depend deterministically on each other.) If we know a distribution is Gaussian and know its mean and covariances, we have completely defined the distribution.

Any marginal distribution of a Gaussian is also Gaussian. So given a joint distribution:

$$p(\mathbf{f}, \mathbf{g}) \;=\; \mathcal{N}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}; \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix}\right), \tag{4}$$

then as soon as you convince yourself that the marginal

$$p(\mathbf{f}) \;=\; \int p(\mathbf{f}, \mathbf{g})\; \mathrm{d}\mathbf{g} \tag{5}$$

is Gaussian, you already know the means and covariances:

$$p(\mathbf{f}) \;=\; \mathcal{N}(\mathbf{f}; \mathbf{a}, A). \tag{6}$$

Any conditional distribution of a Gaussian is also Gaussian:

$$p(\mathbf{f} \,|\, \mathbf{g}) = \mathcal{N}(\mathbf{f};\; \mathbf{a} + CB^{-1}(\mathbf{g} - \mathbf{b}),\; A - CB^{-1}C^\top) \tag{7}$$

Showing this result from scratch requires quite a few lines of linear algebra. However, this is a standard result that is easily looked up (and we wouldn't make you show it in a question!).

3 Representing function values with a Gaussian

We can think of functions as infinite-dimensional vectors. Dealing with the mathematics of infinities and real numbers rigorously is possible but involved. We will side-step these difficulties with a low-tech description.

We could imagine a huge finite discretization of our input space. For example, we could consider only the input vectors $\mathbf{x}$ that can be realized with IEEE floating point numbers. After all, those are the only values we'll ever consider in our code. In theory (but not in practice) we could evaluate a function f at every discrete location, $\tilde{f}_i = f(\mathbf{x}^{(i)})$. We here use the tilde symbol to emphasize the difference between the function f and function values $\tilde{f}_i$. If we put all the $\tilde{f}_i$'s into a vector $\tilde{\mathbf{f}}$, that vector specifies the whole function (up to an arbitrarily fine discretization).

While we can't store a vector containing the function values for every possible input, it is possible to define a multivariate Gaussian prior on it. Given noisy observations of some of the elements of this vector, it will turn out that we can infer other elements of the vector without explicitly representing the whole object.

If our model is going to be useful, the prior needs to be sensible. If we don't have any specific domain knowledge, we'll set the mean vector to zero. If we drew "function vectors" from our prior, an element $\tilde{f}_i = f(\mathbf{x}^{(i)})$, will be positive just as often as it is negative. Defining the covariance is harder. We certainly can't use a diagonal covariance matrix: if our beliefs about the function values are independent, then observing the function in one location will tell us nothing about its values in other locations. We usually want to model continuous functions: function values for nearby inputs will be nearly the same as each other, so function values for nearby inputs should have large covariances.

We will define the covariance between two function values using a covariance or *kernel* function:

$$\mathrm{cov}[\tilde{f}_i, \tilde{f}_j] = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}). \tag{8}$$

There are families of positive definite kernel functions ("Mercer kernels"), which always produce a positive definite matrix K if each element is set to $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. If the kernel wasn't positive definite, our prior wouldn't define a valid Gaussian distribution. One kernel that is positive definite is proportional to a Gaussian:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2), \tag{9}$$

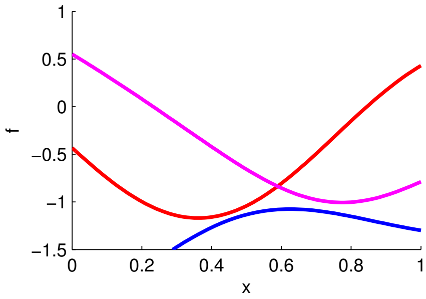
but there are many other choices. For example, another valid kernel is:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (1 + \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|) \exp(-\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|). \tag{10}$$

Gaussian processes get their name because they define a Gaussian distribution over a vector of function values. *Not* because they sometimes use a Gaussian kernel to set the covariances.

4 Using the Gaussian process for regression

The plot below shows three functions drawn from a Gaussian process prior with zero mean and a Gaussian kernel function.



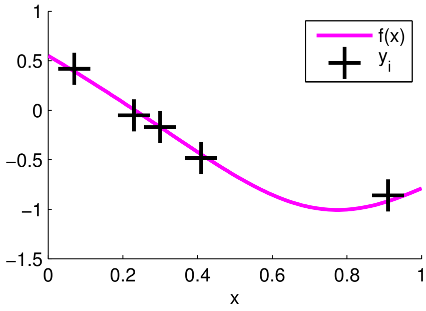
The 'functions' plotted here are actually samples from a 100-dimensional Gaussian distribution. Each sample gave the height of a curve at 100 locations along the x-axis, which were joined up with straight lines. If we chose a different kernel we could get rough functions, straighter functions, or periodic functions.

Our prior is that the function comes from a Gaussian process, $f \sim \mathcal{GP}$, where an $N \times 1$ vector $\mathbf{f}$ of function values at training locations $X = \{\mathbf{x}^{(i)}\}$ has prior $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}; \mathbf{0}, K)$, where $f_i = f(\mathbf{x}^{(i)})$ and $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.

Given noisy observations of the function values at the training locations:

$$y_i \sim \mathcal{N}(f_i, \sigma_y^2), \tag{11}$$

we can update our beliefs about the function. For now we assume the noise variance σ_y^2 is fixed and known. According to the model, the observations are centred around one underlying true function as follows:



But we don't know where the function is, we only see the $\mathbf{y}$ observations.

In 1D we can represent the whole function fairly accurately (as in the plot above) with $\sim$100 values. In high-dimensions however, we can't grid up the input space and explicitly estimate the function everywhere. Instead, we only consider the function at places we have observed, and at test locations $X_* = \{\mathbf{x}^{(*,i)}\}$ where we will make predictions. We call the vector of function values at the test locations $\mathbf{f}_*$. Importantly, the elements of $\mathbf{f}_*$ form a subset of the elements of $\tilde{\mathbf{f}}$ because we assume that $\tilde{\mathbf{f}}$ specifies our function at every possible location.

We can write down the model's joint distribution of the observations and the $\mathbf{f}_*$ function values that we're interested in. This joint distribution is Gaussian, as it's simply a marginal of our prior over the whole function, with noise added to some of the values:

$$p\left(\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix}; \mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_y^2\mathbb{I} & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{12}$$

where the compact notation $K(X,Y)$ follows the Rasmussen and Williams GP textbook (Murphy uses an even more compact notation). Given an $N\times D$ matrix[1] X and an $M\times D$ matrix Y, $K(X,Y)$ is an $N\times M$ matrix of kernel values:

$$K(X,Y)_{ij} = k(\mathbf{x}^{(i)}, \mathbf{y}^{(j)}), \tag{13}$$

where $\mathbf{x}^{(i)}$ is the ith row of X and $\mathbf{y}^{(j)}$ is the jth row of Y.

The covariance of the function values $\mathbf{f}_*$ at the test locations, $K(X_*,X_*)$ is given directly by the prior. The covariance of the observations, $K(X,X)+\sigma_y^2\mathbb{I}$, is given by the prior, plus the independent noise variance. The cross covariances, $K(X,X_*)$ come from the prior on functions, the noise has no effect:

$$\begin{aligned} \mathrm{cov}(y_i, f_{*,j}) &= \mathbb{E}[y_i f_{*,j}] - \mathbb{E}[y_i]\,\mathbb{E}[f_{*,j}] && (14)\\ &= \mathbb{E}[(f_i + \nu_i) f_{*,j}], \quad \nu_i \text{ is noise from } \mathcal{N}(0, \sigma_y^2), && (15)\\ &= \mathbb{E}[f_i f_{*,j}] + \mathbb{E}[\nu_i]\,\mathbb{E}[f_{*,j}], && (16)\\ &= \mathbb{E}[f_i f_{*,j}] = k(\mathbf{x}^{(i)}, \mathbf{x}^{(*,j)}). && (17) \end{aligned}$$

Because Gaussians marginalize so easily, we can ignore the enormous (or infinite) prior covariance matrix over all of the function values that we're not interested in.

Using the rule for conditional Gaussians (for the conditional distribution over a subset of values in a multivariate joint Gaussian given the others), we can immediately identify that the posterior over function values $p(\mathbf{f}_* \,|\, \mathbf{y})$ is Gaussian with:

$$\begin{aligned} \text{mean, } \bar{\mathbf{f}}_* &= K(X_*,X)(K(X,X) + \sigma_y^2\mathbb{I})^{-1}\mathbf{y} && (18)\\ \mathrm{cov}(\mathbf{f}_*) &= K(X_*,X_*) - K(X_*,X)(K(X,X) + \sigma_y^2\mathbb{I})^{-1}K(X,X_*) && (19) \end{aligned}$$

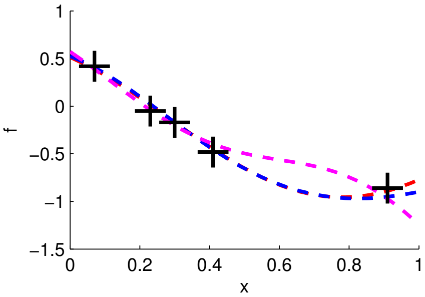
This posterior distribution over a few function values is a marginal of the joint posterior distribution over the whole function. The posterior distribution over possible functions is itself a Gaussian Process (a large or infinite Gaussian distribution).

[The website version of this note has a question here.]

5 Visualizing the posterior

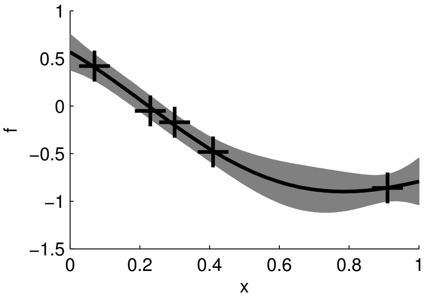
One way to visualize what we believe about the function is to sample plausible realizations from the posterior distribution. The figure below shows three functions sampled from the posterior, two of them are very close to each other.

[1] Actually the inputs don't have to be D-dimensional vectors. There are kernel functions for graphs and strings. Once we have computed the covariances, none of the remaining mathematics looks at the feature vectors themselves.



Really we plotted three samples from a 100-dimensional Gaussian, $p(\mathbf{f}_* \,|\, \text{data})$, where 100 test locations were chosen on a grid. We can see from these samples that we're fairly sure about the function for $x \in [0, 0.4]$, but less sure of what it's doing for $x \in [0.5, 0.8]$.

We can summarize the uncertainty at each input location by plotting the mean of the posterior distribution and error bars. The figure below shows the mean of the posterior plotted as a black line, with a grey band indicating ±2 standard deviations.



This figure doesn't show how the functions might vary within that band, for example whether they are smooth or rough.[2] However, it might be easier to read, and is cheaper to compute. We don't need to compute the posterior covariances between test locations to plot the mean and error band. We just need the 1-dimensional posterior at each test point:

$$p(f(\mathbf{x}^{(*)}) \,|\, \text{data}) \;=\; \mathcal{N}(\tilde{f};\, m,\, s^2). \tag{20}$$

To identify the mean m and variance s^2, we need the covariances:

$$K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}), \quad (\mathbf{k}^{(*)})_i = k(\mathbf{x}^{(*)}, \mathbf{x}^{(i)}). \tag{21}$$

Then the computation is a special case of the posterior expression already provided, but with only one test point:

$$M = K + \sigma_y^2 \mathbb{I}, \tag{22}$$
$$m = \mathbf{k}^{(*)\top} M^{-1} \mathbf{y}, \tag{23}$$
$$s^2 = k(\mathbf{x}^{(*)}, \mathbf{x}^{(*)}) - \underbrace{\mathbf{k}^{(*)\top} M^{-1} \mathbf{k}^{(*)}}_{\text{positive}}. \tag{24}$$

The mean prediction m is just a linear combination of observed function values $\mathbf{y}$.

After observing data, our beliefs are always more confident than under the prior. The amount that the variance improves, $\mathbf{k}^{(*)\top} M^{-1} \mathbf{k}^{(*)}$, doesn't actually depend on the $\mathbf{y}$ values we observe! These properties of inference with Gaussians are computationally convenient: we can pick experiments, knowing how certain we will be after getting its result. However, these properties are somewhat unrealistic. When we see really surprising results, we usually

[2] Sometimes we care about dependencies in predictions: for example, we might care whether a model says that several stock-prices are likely to fall together when they fall.

become less certain, because we realize our model is probably wrong. In the Gaussian process framework we can adjust our kernel function and noise model σ_y^2 in response to the observations, and then our uncertainties will depend on the observations again.

[The website version of this note has a question here.]

6 What you should know

- The marginal and conditional of a joint Gaussian are Gaussian. Be able to write down a marginal distribution given a joint. Given the standard result for Gaussian conditionals, be able to apply it to a particular situation.
- GP idea: explain how a smooth curve or surface relates to a draw from a multivariate Gaussian distribution. How would you plot one?

7 Check your understanding

- Given a GP model (with specified kernel, noise model etc.) and N observations in a regression task, what will you compute to predict the function value (with an error bar) at a test location? What is the computational cost of these operations? Would you actually be able to do it? Or is there anything else you need to know?

8 Reading

The core recommended reading for Gaussian processes is Bishop section 6.4 to section 6.4.3 inclusive.

Alternatives are: Murphy, Chapter 15, to section 15.2.4 inclusive, Barber Chapter 19 to section 19.3 inclusive, or the dedicated Rasmussen and Williams book[3] up to section 2.5. There is also a chapter on GPs in MacKay's book.

Gaussian processes are Bayesian kernel methods. Bishop Chapter 6 to section 6.2 inclusive and Murphy Chapter 14 have more information about kernels in general if you want to read about the wider picture.

The idea of *Bayesian optimization* is old. However, a relatively recent paper rekindled interest in the idea as a way to tune machine learning methods: `https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms` These authors then formed a startup around the idea, which was acquired by Twitter. Other companies like Netflix have also used their software.

[3] `http://gaussianprocess.org/gpml/`