Introduction to Gaussian Processes

Iain Murray
School of Informatics, University of Edinburgh
The problem

Learn scalar function of vector values $f(x)$

We have (possibly noisy) observations $\{x_i, y_i\}_{i=1}^n$
Example Applications

Real-valued regression:

— Robotics: target state $\rightarrow$ required torque
— Process engineering: predicting yield
— Surrogate surfaces for optimization or simulation

Many problems are not regression:
Classification, rating/ranking, discovery, embedding, clustering, . . .
But unknown functions may be part of larger model
Model complexity

The world is often complicated:

- Simple fit
- Complex fit
- Truth

Problems:

- Don’t want to underfit, and be too certain
- Don’t want to overfit, and generalize poorly
- Bayesian model comparison is often hard
Predicting yield

Factory settings $x_1 \rightarrow$ profit of $32 \pm 5$ monetary units

Factory settings $x_2 \rightarrow$ profit of $100 \pm 200$ monetary units

Which are the best settings $x_1$ or $x_2$?

Knowing the error bars can be important
Optimization

In high dimensions it takes many function evaluations to be certain everywhere. Costly if experiments are involved.

Error bars are needed to see if a region is still promising.
Bayesian modelling

If we come up with a parametric family of functions, $f(x; \theta)$ and define a prior over $\theta$, probability theory tells us how to make predictions given data. For flexible models, this usually involves intractable integrals over $\theta$.

We’re really good at integrating Gaussians though

Can we really solve significant machine learning problems with a simple multivariate Gaussian distribution?
Gaussian distributions

Completely described by parameters $\mu$ and $\Sigma$:

$$p(f | \Sigma, \mu) = |2\pi\Sigma|^{-\frac{1}{2}} \exp \left(-\frac{1}{2}(f - \mu)^T \Sigma^{-1} (f - \mu)\right)$$

$\mu$ and $\Sigma$ are the mean and covariance:

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

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

If we know a distribution is Gaussian and know its mean and covariances, we know its density function.
The marginal of a Gaussian distribution is Gaussian.

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

As soon as you convince yourself that the marginal

\[ p(f) = \int p(f, g) \, dg \]

is Gaussian, you already know the means and covariances:

\[ p(f) = \mathcal{N}(f; a, A) \]
Conditional of Gaussian

Any conditional of a Gaussian distribution is also Gaussian:

\[
p(f, g) = \mathcal{N}
\begin{bmatrix}
  f \\
  g
\end{bmatrix};
\begin{bmatrix}
a \\
b
\end{bmatrix},
\begin{bmatrix}
  A & C \\
  C^\top & B
\end{bmatrix}
\]

\[
p(f \mid g) = \mathcal{N}(f; \ a + CB^{-1}(g - b), \ A - CB^{-1}C^\top)
\]

Showing this result requires some grunt work.
But it is standard, and easily looked up up.
Noisy observations

Previously we inferred $f$ given $g$. What if we only saw a noisy observation, $y \sim \mathcal{N}(g, S)$?

$$p(f, g, y) = p(f, g) p(y | g)$$ is Gaussian distributed

a quadratic form inside the exponential after multiplying

Posterior over $f$ is still Gaussian:

$$p(f | y) \propto \int p(f, g, y) \, dg$$

RHS is Gaussian after marginalizing, so still a quadratic form in $f$ inside an exponential.
Laying out Gaussians

A way of visualizing draws from a 2D Gaussian:

Now it’s easy to show three draws from a 6D Gaussian:
Building large Gaussians

Three draws from a 25D Gaussian:

To produce this, we needed a mean: I used zeros(25,1)
The covariances were set using a kernel function: $\Sigma_{ij} = k(x_i, x_j)$.
The $x$’s are the positions that I planted the tics on the axis.
Later we’ll find $k$’s that ensure $\Sigma$ is always positive semi-definite.
GP regression model

\[ f \sim GP \]

\[ f \sim \mathcal{N}(0, K), \quad K_{ij} = k(x_i, x_j) \]

where \( f_i = f(x_i) \)

Noisy observations:

\[ y_i \mid f_i \sim \mathcal{N}(f_i, \sigma_n^2) \]
Our prior over observations and targets is Gaussian:

\[
P \left( \begin{bmatrix} y \\ f_* \end{bmatrix} \right) = \mathcal{N} \left( \begin{bmatrix} y \\ f_* \end{bmatrix}; 0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right)
\]

Using the rule for conditionals, \( p(f_* | y) \) is Gaussian with:

- mean, \( \bar{f}_* = K(X_*, X)(K(X, X) + \sigma_n^2 I)^{-1} y \)
- \( \text{cov}(f_*) = K(X_*, X_*) - K(X_*, X)(K(X, X) + \sigma_n^2 I)^{-1} K(X, X_*) \)

The posterior over functions is a Gaussian Process.
Two incomplete ways of visualizing what we know:

Draws $\sim p(f \mid \text{data})$

Mean and error bars
Point predictions

Conditional at one point $x_*$ is a simple Gaussian:

$$p(f(x_*) | \text{data}) = \mathcal{N}(f; m, s^2)$$

Need covariances:

$$K_{ij} = k(x_i, x_j), \quad (k_*)_i = k(x_*, x_i)$$

Special case of joint posterior:

$$M = K + \sigma_n^2 I$$

$$m = k_\top M^{-1} y$$

$$s^2 = k(x_*, x_*) - \underbrace{k_\top M^{-1} k_*}_{\text{positive}}$$
$p(f_* | \text{data}) = \mathcal{N}(f_*; m, s^2)$
says what we know about the noiseless function.

$p(y_* | \text{data}) = \mathcal{N}(y_*; m, s^2 + \sigma_n^2)$
predicts what we’ll see next.
Review so far

We can represent a function as a *big* vector $f$

We assume that this unknown vector was drawn from a *big* correlated Gaussian distribution, a *Gaussian process*.

(This might upset some mathematicians, but for all practical machine learning and statistical problems, this is fine.)

Observing elements of the vector (optionally corrupted by Gaussian noise) creates a Gaussian posterior distribution. The posterior over functions is still a Gaussian process.

Marginalization in Gaussians is trivial: just ignore all of the positions $x_i$ that are neither observed nor queried.
Covariance functions

The main part that has been missing so far is where the covariance function $k(x_i, x_j)$ comes from.

What else can it say, other than nearby points are similar?
Covariance functions

We can construct covariance functions from parametric models

Simplest example: **Bayesian linear regression:**

\[ f(x_i) = w^\top x_i + b, \quad w \sim \mathcal{N}(0, \sigma_w^2 I), \quad b \sim \mathcal{N}(0, \sigma_b^2) \]

\[
\text{cov}(f_i, f_j) = \mathbb{E}[f_i f_j] - \mathbb{E}[f_i] \mathbb{E}[f_j] = 0
\]

\[
= \mathbb{E}[(w^\top x_i + b)^\top (w^\top x_j + b)]
\]

\[ = \sigma_w^2 x_i^\top x_j + \sigma_b^2 = k(x_i, x_j) \]

Kernel parameters \( \sigma_w^2 \) and \( \sigma_b^2 \) are hyper-parameters in the Bayesian hierarchical model.

More interesting kernels come from models with a large or infinite feature space: \( k(x_i, x_j) = \sigma_w^2 \Phi(x_i)^\top \Phi(x_j) + \sigma_b^2 \), the ‘kernel trick’. 
What’s a valid kernel?

We could ‘make up’ a kernel function $k(x_i, x_j)$

But any ‘Gram matrix’ must be positive semi-definite:

$$K = \begin{pmatrix}
    k(x_1, x_1) & \cdots & k(x_1, x_N) \\
    \vdots & & \vdots \\
    k(x_N, x_1) & \cdots & k(x_N, x_N)
\end{pmatrix}, \quad z^\top K z \geq 0 \text{ for all } z$$

Achieved by positive semi-definite kernel, or Mercer kernel

$K$ +ve eigenvalues $\Rightarrow K^{-1}$ +ve eigenvalues $\Rightarrow$ Gaussian normalizable

Mercer kernels give inner-products of some feature vectors $\Phi(x)$
But these $\Phi(x)$ vectors may be infinite.
Squared-exponential kernel

An infinite number of radial-basis functions can give

\[ k(x_i, x_j) = \sigma_f^2 \exp \left( -\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{d,i} - x_{d,j})^2}{\ell_d^2} \right), \]

the most commonly-used kernel in machine learning.

It looks like an (unnormalized) Gaussian, so is sometimes called the Gaussian kernel.

A Gaussian process need not use the “Gaussian” kernel. In fact, other choices will often be better.
Meaning of hyper-parameters

Many kernels have similar types of parameters:

\[ k(x_i, x_j) = \sigma_f^2 \exp \left( -\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{d,i} - x_{d,j})^2}{\ell_d^2} \right), \]

Consider \( x_i = x_j \), \( \Rightarrow \) marginal function variance is \( \sigma_f^2 \)
Meaning of hyper-parameters

\( \ell_d \) parameters give the length-scale in dimension-\( d \)

\[
k(x_i, x_j) = \sigma_f^2 \exp\left( -\frac{1}{2} \sum_{d=1}^{D} (x_{d,i} - x_{d,j})^2 / \ell_d^2 \right),
\]

Typical distance between peaks \( \approx \ell \)
Effect of hyper-parameters

Different (SE) kernel parameters give different explanations of the data:

\[ \ell = 0.5, \quad \sigma_n = 0.05 \]
\[ \ell = 1.5, \quad \sigma_n = 0.15 \]
Other kernels

SE kernel produce very smooth and ‘boring’ functions

Kernels are available for rough data, periodic data, strings, graphs, images, models, . . .

Different kernels can be combined:

\[ k(x_i, x_j) = \alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j) \]

Positive semi-definite if \( k_1 \) and \( k_2 \) are.
Example of combinations

Combination of kernels for long-term trend, (approximate) periodicity, and short term artifacts:

Figure credit: Carl Rasmussen
http://learning.eng.cam.ac.uk/carl/mauna/
The (marginal) likelihood

The probability of the data is just a Gaussian:

\[
\log p(y \mid X, \theta) = -\frac{1}{2} y^\top M^{-1} y - \frac{1}{2} \log |M| - \frac{n}{2} \log 2\pi
\]

— likelihood of the kernel parameters, \( \theta = \{\ell, \sigma_n, \ldots\} \)
— used to choose amongst kernels

Gradients of the log-likelihood wrt the hyper-parameters can be computed to find (local) maximum likelihood fits.

Because the GP can be viewed as having an infinite number of weight parameters that have been integrated out, \( \log p(y \mid X, \theta) \) is often called the marginal likelihood.
Learning hyper-parameters

The fully Bayesian solution computes the function posterior:

\[ p(f_* | y, X) = \int p(f_* | y, X, \theta) p(\theta | y, X) \, d\theta \]

The first term in the integrand is tractable.

The second term is the posterior over hyper-parameters. This can be sampled using Markov chain Monte Carlo to average predictions over plausible hyper-parameters.
Log-transform +ve inputs

(Wisconsin breast cancer data from the UCI repository)

Positive quantities are often highly skewed
The log-domain is often a much more natural space

A better transformation could be learned:
Log-transform +ve outputs

Warped Gaussian processes, Snelson et al. (2003)

Learned transformations for positive data were log-like. Always consider log transforming positive data.

However, other transformations (or none at all) are sometimes the best option.
Mean function

Using $\mathbf{f} \sim \mathcal{N}(0, K)$ is common

Poor model if data has mean far from zero!
Center your data, or use a parametric mean function $m(x)$. 
Other tricks

To set initial hyper-parameters, use domain knowledge wherever possible. Otherwise...

Standardize input data and set lengthscales to $\sim 1$.

Standardize targets and set function variance to $\sim 1$.

Often useful: set initial noise level high, even if you think your data have low noise. The optimization surface for your other parameters will be easier to move in.

If optimizing hyper-parameters, (as always) random restarts or other tricks to avoid local optima are advised.
Real data can be nasty

A projection of a robot arm problem:

Common artifacts: thresholding, jumps, clumps, \(k^n\)k

How might we fix these problems?
Non-Gaussian likelihoods

GP regression is tractable because both the prior and likelihood are Gaussian.

There are many reasons to want to use non-Gaussian likelihoods, although we can no longer marginalize out the unknown function values at the observations. We can use approximate inference methods such as MCMC, Laplace, or variational methods.

A common application of a non-Gaussian likelihood is a model of heavy-tailed noise to account for large outliers.
Classification

Special case of a non-Gaussian noise model

Assume $y_i \sim \text{Bernoulli}(\text{sigmoid}(f_i))$

MCMC can sum over the latent function values.
Variational methods also work well.

Figures from Bishop textbook
Regressing on the labels

If we give up on a Bayesian modelling interpretation, we could just apply standard GP regression code on binary classification data with $y \in \{-1, +1\}$.

The sign of the mean function is a reasonable hard classifier. Asymptotically the posterior function will be peaked around $f(x) = 2p(x) - 1$.

**Multiway classification:** regressing $y \in \{1, 2, \ldots, C\}$ would be a bad idea. Instead, train $C$ “one-against all” classifiers and pick class with largest mean function.

**Not really Gaussian process modelling any more:** this is just regularized least squares fitting
Exploding costs

GPs scale poorly with large datasets

$O(n^3)$ computation usually takes the blame:

$M^{-1}$ or $M^{-1}y$, $M^{-1}k_*$ and $\text{det}(M)$

Not the only story:

$K_{ij} = k(x_i, x_j)$  
$O(dn^2)$ computation

$O(n^2)$ memory

Large literature on GP approximations
Subset of Data

Trivial, obvious solution:

*randomly throw away most of the data*

There are also methods to choose greedily, cheaply choose which points to keep.
Take-home messages

● **Simple to use:**
  – Just matrix operations (if likelihoods are Gaussian)
  – Few parameters: *relatively* easy to set or sample over
  – Predictions are often very good

● **No magic bullet:** best results need (at least) careful
data scaling, which could be modelled or done by hand.

● **The need for approximate inference:**
  – Sometimes Gaussian likelihoods aren’t enough
  – $O(n^3)$ and $O(n^2)$ costs are bad news for big problems
Further reading

Many more topics and code:
http://www.gaussianprocess.org/gpml/

More software:
http://becs.aalto.fi/en/research/bayes/gpstuff/
http://sheffieldml.github.io/GPy/