PEN Emulator Workshop
Lab 1: Brief recap of GPs

Richard Wilkinson

University of Sheffield

Gaussian processes

We are given data $D = \{x_i, y_i\}_{i=1}^n$ which is pairs of inputs and outputs. We want to learn the functional relationship between inputs x and outputs y. We will use a GP to model this relationship.

Gaussian processes

We are given data $D = \{x_i, y_i\}_{i=1}^n$ which is pairs of inputs and outputs. We want to learn the functional relationship between inputs x and outputs y. We will use a GP to model this relationship.

Our GP model consists of the following:

- A mean function
  - Zero or a constant can often be a good choice.

Gaussian processes

We are given data $D = \{x_i, y_i\}_{i=1}^n$ which is pairs of inputs and outputs. We want to learn the functional relationship between inputs x and outputs y. We will use a GP to model this relationship.

Our GP model consists of the following:

- A mean function
  - Zero or a constant can often be a good choice.
- A covariance function. This (and the mean function) will often depend on unknown parameters ψ.

Gaussian processes

We are given data $D = \{x_i, y_i\}_{i=1}^n$ which is pairs of inputs and outputs. We want to learn the functional relationship between inputs x and outputs y. We will use a GP to model this relationship.

Our GP model consists of the following:

- A mean function
  - Zero or a constant can often be a good choice.
- A covariance function. This (and the mean function) will often depend on unknown parameters ψ.
- A likelihood relating the unknown function $f(x)$ to the data y, e.g.

$$y_i = f(x_i) + \epsilon_i \qquad \text{where} \qquad \epsilon_i \sim N(0, \sigma^2)$$

  with $\epsilon_i \perp\!\!\!\perp \epsilon_j$ for $i \neq j$.

  Here f is the continuous latent function underlying the relationship - it is this we will model by a GP.

Gaussian processes

We are given data $D = \{x_i, y_i\}_{i=1}^n$ which is pairs of inputs and outputs. We want to learn the functional relationship between inputs x and outputs y. We will use a GP to model this relationship.

Our GP model consists of the following:

- A mean function
  - Zero or a constant can often be a good choice.
- A covariance function. This (and the mean function) will often depend on unknown parameters ψ.
- A likelihood relating the unknown function $f(x)$ to the data y, e.g.

  $$y_i = f(x_i) + \epsilon_i \qquad \text{where} \qquad \epsilon_i \sim N(0, \sigma^2)$$

  with $\epsilon_i \perp\!\!\!\perp \epsilon_j$ for $i \neq j$.

  Here f is the continuous latent function underlying the relationship - it is this we will model by a GP.

Then inference consists of

- A Bayesian updating scheme to infer $\pi(f(x)|D, \psi)$.
- An approach to estimate the hyper-parameters, ψ.

Mean functions

The mean function

$$m(x) = \mathbb{E}f(x)$$

is

- the most important/unimportant part of a GP, and
- is a key aspect of your inference/an unnecessary parametric bottleneck.

Mean functions

The mean function

$$m(x) = \mathbb{E}f(x)$$

is

- the most important/unimportant part of a GP, and
- is a key aspect of your inference/an unnecessary parametric bottleneck.

The machine learning world tends to set $m(x) = 0$ or sometimes a non-zero constant, but they are often working with large amounts of closely packed data.

Mean functions

The mean function

$$m(x) = \mathbb{E}f(x)$$

is

- the most important/unimportant part of a GP, and
- is a key aspect of your inference/an unnecessary parametric bottleneck.

The machine learning world tends to set $m(x) = 0$ or sometimes a non-zero constant, but they are often working with large amounts of closely packed data.

The UQ/emulator world tends to put more effort into choosing a good parametric form for $m(x) = \beta x$, as their data is often sparse, and so a strong parametric component can help (or sometimes hinder...) predictions in regions where we lack data.

Covariance functions/kernels

The covariance function **is** the most important aspect of a GP

$$k(x, x') := \mathbb{C}\mathrm{ov}(f(x), f(x'))$$

The space of functions that samples from a given GP live in, is determined by the choice of k.

Covariance functions/kernels

The covariance function **is** the most important aspect of a GP

$$k(x, x') := \mathbb{C}\text{ov}(f(x), f(x'))$$

The space of functions that samples from a given GP live in, is determined by the choice of k.

- If k is stationary, then we can write $k(x, x') = k(x - x')$. The differentiability of the sample paths matches the differentiability of $k(r)$ at $r = 0$, i.e., if $k(r)$ is twice differentiable, then the sample paths will be twice differentiable.

Covariance functions/kernels

The covariance function **is** the most important aspect of a GP

$$k(x, x') := \mathbb{C}\text{ov}(f(x), f(x'))$$

The space of functions that samples from a given GP live in, is determined by the choice of k.

- If k is stationary, then we can write $k(x, x') = k(x - x')$. The differentiability of the sample paths matches the differentiability of $k(r)$ at $r = 0$, i.e., if $k(r)$ is twice differentiable, then the sample paths will be twice differentiable.

Covariance functions are often parameterized in terms of hyperparameters, e.g.,

$$k(x, x') = \sigma^2 \exp(-(x - x')^2/\lambda^2)$$

Here σ^2 controls the variance of the sample paths, and λ determines their length-scale.

- Rough rule of thumb: if $|x - x'| > 2\lambda$ then $f(x)$ and $f(x')$ are close to being uncorrelated.

Posterior inference

The main reason for using GPs is that if $f(\cdot) \sim GP(m(\cdot), k(\cdot))$, then if we observe data (x_i, y_i)

$$y_i = f(x_i) + \epsilon_i \qquad \text{where } \epsilon_i \sim N(0, \sigma^2)$$

then

$$f(\cdot)|D \sim GP(m^*(\cdot), k^*(\cdot))$$

ie they form a closed class of models under Bayesian conditioning.

Posterior inference

The main reason for using GPs is that if $f(\cdot) \sim GP(m(\cdot), k(\cdot))$, then if we observe data (x_i, y_i)

$$y_i = f(x_i) + \epsilon_i \qquad \text{where } \epsilon_i \sim N(0, \sigma^2)$$

then

$$f(\cdot)|D \sim GP(m^*(\cdot), k^*(\cdot))$$

ie they form a closed class of models under Bayesian conditioning.
In GPy, the `likelihood` refers to the distribution of the random error ϵ.

If $\epsilon \sim N(0, \sigma^2)$ then usually we just do the Bayesian updating to train the model.

Posterior inference

The main reason for using GPs is that if $f(\cdot) \sim GP(m(\cdot), k(\cdot))$, then if we observe data (x_i, y_i)

$$y_i = f(x_i) + \epsilon_i \qquad \text{where } \epsilon_i \sim N(0, \sigma^2)$$

then

$$f(\cdot)|D \sim GP(m^*(\cdot), k^*(\cdot))$$

ie they form a closed class of models under Bayesian conditioning.
In GPy, the `likelihood` refers to the distribution of the random error ϵ.

If $\epsilon \sim N(0, \sigma^2)$ then usually we just do the Bayesian updating to train the model.
Some exceptions:

- If ϵ has a non-Gaussian distribution, then we need to use an alternative updating scheme.
- If n is large, then the full Bayesian updating can be prohibitively expensive, as we have to invert a $n \times n$ matrix (cost $O(n^3)$). There are sparse methods to reduce this cost (included in GPy) that allow us to work with larger datasets.

Estimating hyper-parameters

- If we know the hyper-parameters, working with GPs is a pleasant experience, and the theory is beautiful and simple.
- But if we need to estimate hyper-parameters, then everything gets much trickier.

Estimating hyper-parameters

- If we know the hyper-parameters, working with GPs is a pleasant experience, and the theory is beautiful and simple.
- But if we need to estimate hyper-parameters, then everything gets much trickier.

There are various approaches for estimating the hyper-parameters.

- Maximum likelihood (and related) approaches are the default choice
- Cross-validation
- Sampling them from their posterior distribution (e.g. using MCMC)

Estimating hyper-parameters

- If we know the hyper-parameters, working with GPs is a pleasant experience, and the theory is beautiful and simple.
- But if we need to estimate hyper-parameters, then everything gets much trickier.

There are various approaches for estimating the hyper-parameters.

- Maximum likelihood (and related) approaches are the default choice
- Cross-validation
- Sampling them from their posterior distribution (e.g. using MCMC)

The estimation is often tricky, as the likelihood surface can be flat with multiple local maxima.

Estimating hyper-parameters

- If we know the hyper-parameters, working with GPs is a pleasant experience, and the theory is beautiful and simple.
- But if we need to estimate hyper-parameters, then everything gets much trickier.

There are various approaches for estimating the hyper-parameters.

- Maximum likelihood (and related) approaches are the default choice
- Cross-validation
- Sampling them from their posterior distribution (e.g. using MCMC)

The estimation is often tricky, as the likelihood surface can be flat with multiple local maxima.

Sometimes we need to resort to tricks such as constraining values to lie in some interval (or placing a prior distribution on them), or fixing them at sensible values by hand.

GPy

GPs are not usually robust models that we can treat as magic black-boxes.

- GPy is probably the most complete GP implementation, written by people deeply involved in the GP world.

GP fitting can often go wrong.

- Hyper-parameter optimization is usually the problem.
- Try constraining the hyper parameters using your judgement.

You must check your model in some way.

- Plot your fitted model
- Test its predictive skill (held out data, or cross-validation)
- The optimized value of the log-likelihood tells you how good a fit you've found - the larger the better, but it does not indicate predictive skill.