Surrogates 7020

Chapter 5(Part 1): Gaussian Process Regression

Dr. Alex Bledar Konomi

Department of Mathematical Sciences
University of Cincinnati

UNIVERSITY OF Cincinnati

Gaussian Processes

- Our aim is to understand the Gaussian process (GP) as a prior over random functions, a posterior over functions given observed data, as a tool for spatial data modeling and surrogate modeling for computer experiments, and simply as a flexible nonparametric regression.
- We'll see that, almost in spite of a technical over-analysis of its properties, and sometimes strange vocabulary used to describe its features.
- GP regression is a simple extension of linear modeling. Knowing that is all it takes to make use of it as a nearly unbeatable regression tool when input–output relationships are relatively smooth, and signal-to-noise ratios relatively high.

Gaussian Process

- Gaussian process is a stochastic process (a collection of random variables indexed by input), such that every finite collection of those random variables has a multivariate normal (MVN) distribution.
- That, in turn, means that characteristics of those realizations are completely described by their mean n-vector μ and $n \times n$ covariance matrix Σ.
- With interest in modeling functions, we'll sometimes use the term mean function, thinking of $\mu(\boldsymbol{x})$, and covariance function, thinking of $\Sigma(\boldsymbol{x}, \boldsymbol{x}')$.
- But ultimately we'll end up with vectors μ and matrices Σ after evaluating those functions at specific input locations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

Gaussian Process Prior

- The action, at least the part that's interesting, in a GP treatment of functions is all in the covariance.
- Consider a covariance function defined by inverse exponentiated squared Euclidean distance:

  $$\Sigma(\boldsymbol{x}, \boldsymbol{x}') = \exp\{-||\boldsymbol{x} - \boldsymbol{x}'||^2\}$$

  Here covariance decays exponentially fast as $\boldsymbol{x}$ and $\boldsymbol{x}'$ become farther apart in the input, or x-space. In this specification, observe that $\Sigma(\boldsymbol{x}, \boldsymbol{x}) = 1$ and $\Sigma(\boldsymbol{x}, \boldsymbol{x}') < 1$ for $\boldsymbol{x}' \neq \boldsymbol{x}$.
- The function $\Sigma(\boldsymbol{x}, \boldsymbol{x})$ must be positive definite. For us this means that if we define a covariance matrix $\boldsymbol{\Sigma}_n$, based on evaluating $\Sigma(\boldsymbol{x}, \boldsymbol{x})$ at pairs of n x-values $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, we must have that $\boldsymbol{x}^T \boldsymbol{\Sigma} \boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq 0$.

Generating a Sample from a GP

- Let's first see how GPs can be used to generate random data following a smooth functional relationship.
- Suppose we take a bunch of x-values: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, define $\boldsymbol{\Sigma}_n$ via $\boldsymbol{\Sigma}_n^{ij} = \Sigma(\boldsymbol{x}_i, \boldsymbol{x}_j)$, for $i, j = 1, \ldots, n$, then draw an n-variate realization

$$\boldsymbol{Y} \sim \mathcal{N}_n(0, \boldsymbol{\Sigma}_n)$$

- One simple option for design is random uniform. That is, fill a design matrix $\boldsymbol{X}_n$ in $[0,1]^m$ with n runs via *runif*.
- Note that the mean of this MVN is zero; this need not be but it's quite surprising how well things work even in this special case.
- Location invariant zero-mean GP modeling, sometimes after subtracting off a middle value of the response (e.g., $\bar{y}$), is the default in computer surrogate modeling and (ML) literature.
- More later for a general mean.

Algorithm for Generating a GP Sample

```
install.packages("plgp")
library(plgp)
n <- 100
X <- matrix(seq(0, 10, length=n), ncol=1)
D <- distance(X)  # Distance Matrix

eps <- sqrt(.Machine$double.eps)

Sigma <- exp(-D) + diag(eps, n) # Covaraince matrix

library(mvtnorm)
Y <- rmvnorm(1, sigma=Sigma) # Generating from GP

plot(X, Y, type="l")

$
```

Some Properties

- The code augments the diagonal with a small number eps $\equiv \epsilon$. Although inverse exponentiated distances guarantee a positive definite matrix in theory, sometimes in practice the matrix is numerically ill-conditioned. Augmenting the diagonal a tiny bit prevents that. Neal (1998), a GP vanguard in the statistical/ML literature, calls ϵ the jitter in this context.
- Because the Y-values are random, you'll get a different curve when you try this on your own.
- What are the properties of this function, or more precisely of a random function generated in this way?
  - Several are easy to deduce from the form of the covariance structure.
  - We'll get a range of about $[-2, 2]$, with 95% probability, because the scale of the covariance is 1, ignoring the jitter ϵ added to the diagonal.
  - We'll get several bumps in the x-range of $[0, 10]$ because short distances are highly correlated.

Smooth Function

- The function plotted above is only a finite realization, meaning that we really only have 100 pairs of points.
- Those points look smooth, in a tactile sense, because they're close together and because the plot function is "connecting the dots" with lines.
- The full surface, which you might conceptually extend to an infinite realization over a compact domain, is extremely smooth in a calculus sense **because the covariance function is infinitely differentiable**
- It has to do with the covariance function! We will discussion this a little bit later.

Three Different Realizations

Besides those three things – scale of two, several bumps, smooth look – we won't be able to anticipate much else about the nature of a particular realization. Example below:

```
Y <- rmvnorm(3, sigma=Sigma)
matplot(X, t(Y), type="l", ylab="Y")
```

Gaussian process posterior

- Given examples of a function in pairs $(x_1, y_1), \ldots, (x_n, y_n)$, comprising data $\boldsymbol{D}_n = (\boldsymbol{X}_n, \boldsymbol{Y}_n)$: what random function realizations could explain – could have generated – those observed values?
- That is, we want to know about the conditional distribution of $Y(x)|\boldsymbol{D}_n$. If we call $Y(x) \sim GP$ the prior, then $Y(x)|\boldsymbol{D}_n$ must be the posterior.
- You do not really have to know Bayesian Statistics at this point: the only thing you have to know is the conditional distribution, $Y(x)|\boldsymbol{D}_n$, which one might more simply call a predictive distribution, and everybody is familiar with this quantity in regression analysis.

Conditional Distribution

Deriving that predictive distribution is a simple application of deducing a conditional from a (joint) MVN. If an N-dimensional random vector $\boldsymbol{X}$ is partitioned as $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{pmatrix}$ with sizes $\begin{pmatrix} q \times 1 \\ (N-q) \times 1 \end{pmatrix}$. Accordingly the mean $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are partitioned:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}$$

then the distribution of $\boldsymbol{X}_1$ conditional on $\boldsymbol{X}_2 = \boldsymbol{x}_2$ is MVN $\boldsymbol{X}_1 | \boldsymbol{x}_2 \sim N_q(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}})$, where

$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)$$

$$\bar{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$$

Conditional Distribution Features

- An interesting feature of this result is that conditioning upon $\boldsymbol{x}_2$ alters the variance of $\boldsymbol{X}_1$. Observe that $\bar{\boldsymbol{\Sigma}}$ above is reduced compared to its marginal analog Σ_{11}.
    - Reduction in variance when conditioning on data is a hallmark of statistical learning. We know more – have less uncertainty – after incorporating data.
    - The amount by which variance is decreased doesn't depend (directly) on the value of $\boldsymbol{x}_2$.
- Observe that the mean is also altered, comparing $\boldsymbol{\mu}_1$ to $\bar{\boldsymbol{\mu}}$. In fact, the equation for $\bar{\boldsymbol{\mu}}$ is a linear mapping, i.e., of the form $ax + b$ for vectors a and b.
- Finally, note that $\boldsymbol{\Sigma}_{12} = \boldsymbol{\Sigma}_{21}^T$ so that $\bar{\boldsymbol{\Sigma}}$ is symmetric.

Prediction Distribution

How do we deploy that fundamental MVN result towards deriving the GP predictive distribution $Y(\boldsymbol{x})|\boldsymbol{D}_n$? Consider an $n+1$st observation $Y(\boldsymbol{x})$. Allow $Y(\boldsymbol{x})$ and $\boldsymbol{Y}_n$ to have a joint MVN distribution with mean zero and covariance function $\Sigma(\boldsymbol{x}, \boldsymbol{x}')$. That is, stack:

$$\boldsymbol{Y}_{n+1} = \begin{pmatrix} Y(\boldsymbol{x}) \\ \boldsymbol{Y}_n \end{pmatrix}$$

Apply the conditional representation above the predictive distribution is $Y(\boldsymbol{x})|\boldsymbol{D}_n \sim N(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$ where:

$$\boldsymbol{\mu}(\boldsymbol{x}) = \Sigma(\boldsymbol{x}, \boldsymbol{X}_n)\boldsymbol{\Sigma}_n^{-1}\boldsymbol{Y}_n$$

$$\sigma^2(\boldsymbol{x}) = \Sigma(\boldsymbol{x}, \boldsymbol{x}) - \Sigma(\boldsymbol{x}, \boldsymbol{X}_n)\boldsymbol{\Sigma}_n^{-1}\Sigma(\boldsymbol{x}, \boldsymbol{X}_n)^T$$

Prediction Distribution Features

- Observe that $\mu(\boldsymbol{x})$ is linear in observations $\boldsymbol{Y}_n$, so we have a linear predictor! In fact it's the best linear unbiased predictor (BLUP), an argument we'll leave to other texts (e.g., Santner, Williams, and Notz 2018).
- Also notice that $\sigma^2(\boldsymbol{x})$ is lower than the marginal variance. So we learn something from data $\boldsymbol{Y}_n$; **in fact the amount that variance goes down is a quadratic function of distance between x and $\boldsymbol{X}_n$.**
- Learning is most efficient for $\boldsymbol{x}$ that are close to training data locations $\boldsymbol{X}_n$. However the amount learned doesn't depend upon $\boldsymbol{Y}_n$.

Joint Prediction Distribution

The derivation above is for "pointwise" GP predictive calculations. These are sometimes called the kriging equations, especially in geospatial contexts. We can apply them, separately, for many predictive/testing locations $\boldsymbol{x}$, one $\boldsymbol{x}$ at a time, but that would ignore the obvious correlation they'd experience in a big MVN analysis. Alternatively, we may consider a bunch of $\boldsymbol{x}$ locations jointly, in a testing design $\boldsymbol{X}$ of n' rows, say, all at once:

$$\boldsymbol{Y}(\mathcal{X})|\boldsymbol{D}_n \sim \mathcal{N}_{n'}(\boldsymbol{\mu}(\mathcal{X}), \boldsymbol{\Sigma}(\mathcal{X}))$$

$$\boldsymbol{\mu}(\mathcal{X}) = \Sigma(\mathcal{X}, \boldsymbol{X}_n)\boldsymbol{\Sigma}_n^{-1}\boldsymbol{Y}_n$$

$$\Sigma(\mathcal{X}) = \Sigma(\mathcal{X}, \mathcal{X}) - \Sigma(\mathcal{X}, \boldsymbol{X}_n)\boldsymbol{\Sigma}_n^{-1}\Sigma(\mathcal{X}, \boldsymbol{X}_n)^T$$

Joint Prediction Distribution

where $\Sigma(\mathcal{X}, \boldsymbol{X}_n)$ is an $n' \times n$ matrix. Having a full covariance structure offers a more complete picture of the random functions which explain data under a GP posterior, but also more computation. The $n' \times n'$ matrix $\Sigma(\mathcal{X})$ could be enormous even for seemingly moderate n'.

Simple 1d GP prediction example

Consider a toy example in 1d where the response is a simple sinusoid measured at eight equally spaced $\boldsymbol{x}$ -locations in the span of a single period of oscillation. R code below provides relevant data quantities, including pairwise squared distances between the input locations collected in the matrix D, and its inverse exponentiation in Sigma

```
n <- 8
X <- matrix(seq(0, 2*pi, length=n), ncol=1)
y <- sin(X)
D <- distance(X)
Sigma <- exp(-D) + diag(eps, ncol(D))
```

Simple 1d GP prediction example

Now this is where the example diverges from our earlier one, where we used such quantities to generate data from a GP prior. Applying MVN conditioning equations requires similar calculations on a testing design X, coded as XX below. We need inverse exponentiated squared distances between those XX locations

```
XX <- matrix(seq(-0.5, 2*pi + 0.5, length=100), ncol=1)
DXX <- distance(XX)
SXX <- exp(-DXX) + diag(eps, ncol(DXX))

## between testing locations  and training data locations X_n.
DX <- distance(XX, X)
SX <- exp(-DX)
```

Simple 1d GP prediction example

```
XX <- matrix(seq(-0.5, 2*pi + 0.5, length=100), ncol=1)
DXX <- distance(XX)
SXX <- exp(-DXX) + diag(eps, ncol(DXX))

## between testing locations  and training data locations X_n.
DX <- distance(XX, X)
SX <- exp(-DX)
```

Simple 1d GP prediction example

Mean vector and covariance matrix in hand, we may generate Y-values from the posterior/predictive distribution $Y(\boldsymbol{X})|\boldsymbol{D}_n$ in the same manner as we did from the prior. (Also calculate the PI)

```
YY <- rmvnorm(100, mup, Sigmap)

q1 <- mup + qnorm(0.05, 0, sqrt(diag(Sigmap)))
q2 <- mup + qnorm(0.95, 0, sqrt(diag(Sigmap)))

##################

matplot(XX, t(YY), type="l", col="gray", lty=1, xlab="x", ylab="y")
points(X, y, pch=20, cex=2)
lines(XX, sin(XX), col="blue")
lines(XX, mup, lwd=2)
lines(XX, q1, lwd=2, lty=2, col=2)
lines(XX, q2, lwd=2, lty=2, col=2)
```

What do we observe?

- Notice how the predictive surface interpolates the data. That's because $\Sigma(\boldsymbol{x}, \boldsymbol{x}) = 1$ and $\Sigma(\boldsymbol{x}, \boldsymbol{x}') \longrightarrow 1$ as $x' \longrightarrow x$.
- Error-bars take on a "football" (American) shape, or some say a "sausage" shape, being widest at locations farthest from x_i-values in the data. Error-bars get really big outside the range of the data, a typical feature in ordinary linear regression settings.
- But the predictive mean behaves rather differently than under an ordinary linear model. For GPs it's mean-reverting, eventually leveling off to zero as $x \in \mathcal{X}$ gets far away from $\boldsymbol{X}_n$. Predictive variance, as exemplified by those error-bars, is also reverting to something: a prior variance of 1.
- Variance won't continue to increase as $\boldsymbol{x}$ gets farther and farther from $\boldsymbol{X}_n$. Together those two "reversions" imply that although we can't trust extrapolations too far outside of the data range, at least their behavior isn't unpredictable, as can sometimes happen in linear regression contexts, for example when based upon feature-expanded (e.g., polynomial basis) covariates.

Higher dimension?

There's nothing particularly special about the presentation above that would preclude application in higher input dimension. Except perhaps that visualization is a lot simpler in $1d$ or $2d$. We'll get to even higher dimensions with some of our later examples. For now, consider a random function in $2d$ sampled from a GP prior. The plan is to go back through the process above: first prior, then (posterior) predictive, etc.

Drawing from 2d GP prior

Begin by creating an input set, $\boldsymbol{X}_n$, in two dimensions. Here we'll use a regular 20×20 grid.

```
nx <- 20; x <- seq(0, 2, length=nx); X <- expand.grid(x, x)

D <- distance(X)
Sigma <- exp(-D) + diag(eps, nrow(X))
Y <- rmvnorm(2, sigma=Sigma)

par(mfrow=c(1,2))
persp(x, x, matrix(Y[1,], ncol=nx), theta=-30, phi=30, xlab="x1",
  ylab="x2", zlab="y")
persp(x, x, matrix(Y[2,], ncol=nx), theta=-30, phi=30, xlab="x1",
  ylab="x2", zlab="y")
```

2d simple function

Consider the 2d function $y(x) = x_1 \exp\{-x_1^2 - x_2^2\}$ which is highly nonlinear near the origin, but flat (zero) as inputs get large. This function has become a benchmark 2d problem in the literature for reasons that we'll get more into in Chapter 9.

```
library(lhs)
X <- randomLHS(40, 2)
X[,1] <- (X[,1] - 0.5)*6 + 1
X[,2] <- (X[,2] - 0.5)*6 + 1
y <- X[,1]*exp(-X[,1]^2 - X[,2]^2)
######
xx <- seq(-2, 4, length=40)
XX <- expand.grid(xx, xx)

D <- distance(X)
Sigma <- exp(-D)
```

2d simple function

```
DXX <- distance(XX)
SXX <- exp(-DXX) + diag(eps, ncol(DXX))
DX <- distance(XX, X)
SX <- exp(-DX)

Si <- solve(Sigma)
mup <- SX %*% Si %*% y
Sigmap <- SXX - SX %*% Si %*% t(SX)
sdp <- sqrt(diag(Sigmap))

par(mfrow=c(1,2))
cols <- heat.colors(128)
image(xx, xx, matrix(mup, ncol=length(xx)), xlab="x1", ylab="x2", col=cols)
points(X[,1], X[,2])
image(xx, xx, matrix(sdp, ncol=length(xx)), xlab="x1", ylab="x2", col=cols)
points(X[,1], X[,2])

persp(xx, xx, matrix(mup, ncol=40), theta=-30, phi=30, xlab="x1",
  ylab="x2", zlab="y")
```

GP hyperparameters

- The GP is called (most of the times) a non-parametric model or regression.
- All this business about nonparametric regression and here we are introducing parameters. (but with a different name: hyperparamters)
- How can one have hyperparameters without parameters to start with, or at least to somehow distinguish from?
- To make things even more confusing, we go about learning those hyperparameters in the usual way, by optimizing something, just like parameters. These hyperparameters are more of a fine tuning.
- The fact is that you can express the GPs with latent variables (which we are not going to cover) and then model the latent variable covariance and not the mean.