Gaussian Processes : Introduction, application and challenges

Rahul Rahaman

28 March 2019

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process

3 Challenges of Gaussian Process

4 Summary

Table of Contents

1 Introduction to Gaussian Process
- Definition & Properties
- Kernel Functions
- Motivation of using GP

2 Application of Gaussian Process

3 Challenges of Gaussian Process

4 Summary

Definition of GP

A Gaussian Process with a mean function m, covariance kernel K, is a collection of random variable $\{f_t\}_{t \in T}$ (where T can be an uncountable index set) such that for any finite collection $t_1, t_2, .., t_k$

$$f(t_1), f(t_2), .., f(t_k) \sim \mathcal{N}(\mu_k, \Sigma_{k \times k})$$

where $\mu := m(t_1, t_2, ..., t_k)$, $\Sigma_{ij} := K(t_i, t_j)$
This Gaussian marginalization makes it easy to use and interpret GP confined to a finite set of points.

Properties of GP

- A GP can be viewed as a distribution over functions f on the index set T
- When we are confining ourselves to a finite set of points $t_1, t_2, .., t_k$ we are actually looking at the distribution of the function to the finite sets, which follows Gaussian.
- The mean function controls the general trend of the GP, while the covariance kernel controls the smoothness of the GP function.

Simulating Gaussian Process

To simulate a GP with a known mean function m and covariance kernel K, we do the following steps

- Choose an interval for simulation e.g [0,1]
- Get a set of equally spaced points in the interval $x_1, x_2, ..., x_k$ e.g. $\{0.01, 0.02, ..., 1\}$
- Calculate $\mu := (m(x_1), m(x_2), ..., m(x_k))$ and $\Sigma := (K(x_i, x_j))_{ij}$
- Simulate from $\mathcal{N}(\mu_k, \Sigma_{k\times k})$
- Connect the points to get a sample of the GP. Here by connecting we are actually choosing one particular GP function out of all those which has the same values at $x_1, x_2, ..., x_k$

Simulating Gaussian Process

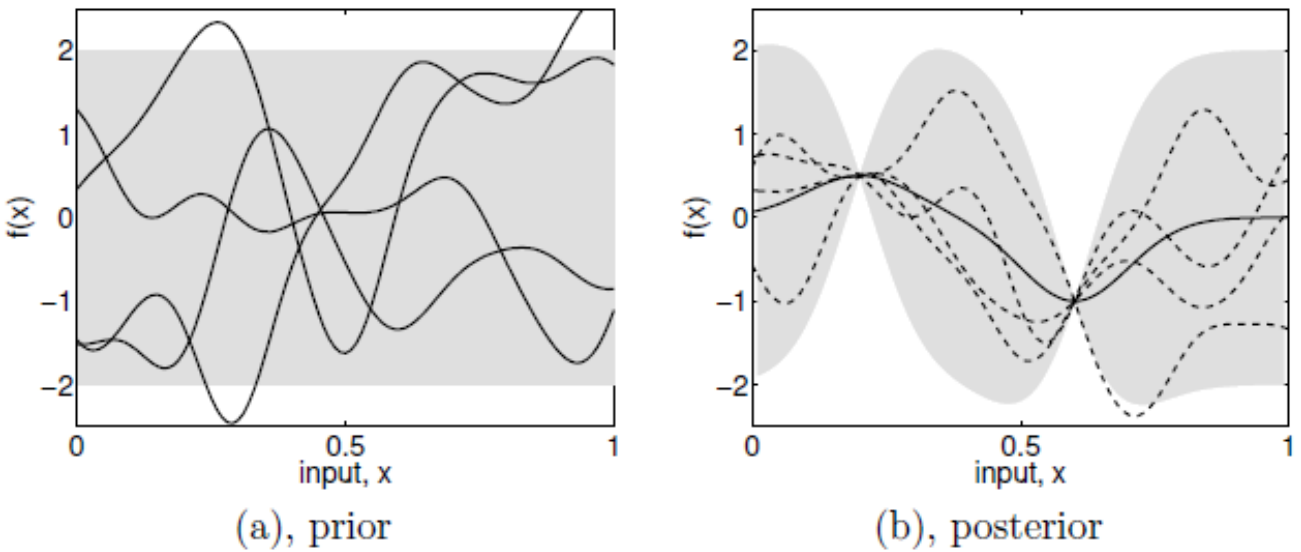

Figure: *(a) GP simulation on unit interval with a mean and covariance kernel (b) GP simulation after pivoting values at two points (posterior distribution). The shaded region in both the graphs shows the interval of length twice the variance around the mean at each point*

Table of Contents

1 Introduction to Gaussian Process
- Definition & Properties
- Kernel Functions
- Motivation of using GP

2 Application of Gaussian Process

3 Challenges of Gaussian Process

4 Summary

Role of Kernel function

- The main importance of Kernel function is to control the variance of the function around the mean. Note that the covariance between x_i, x_j is $Cov(f(x_i), f(x_j)) = K(x_i, x_j)$ That means the kernel also controls how much dependent the function values will be at two distinct points.
- If the covariance kernel is of the form $K(x_i, x_j) = g(d(x_i, x_j))$ where g is a decreasing function and $d(x_i, x_j)$ is some kind of distance function between the two datapoints, then the GP will have the property that the covariance is highest in a close neighborhood around a point implying continuity.
- A kernel is stationary if the kernel function depends on x_i, x_j only by $\|x_i - x_j\|$

Examples

Some covariance kernels are-

- Squared exponential kernel $K(x, y) = s \exp(-\frac{\|x-y\|^2}{2l^2})$
- Dot product kernel $K(x, y) = \sigma^2 + x^\intercal \Sigma y$
- Neural network kernel $K(x, y) = \frac{2}{\pi} \sin^{-1}\left(\frac{2x^\intercal \Sigma y}{\sqrt{(1+2x^\intercal \Sigma x)(1+2y^\intercal \Sigma y)}}\right)$

Table of Contents

1 Introduction to Gaussian Process
- Definition & Properties
- Kernel Functions
- Motivation of using GP

2 Application of Gaussian Process

3 Challenges of Gaussian Process

4 Summary

Bayesian Linear Regression

We have the model

$$y = X\beta + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I),\ \beta \sim \mathcal{N}(0, \Sigma)$$
$$\implies X\beta \sim \mathcal{N}(0, X\Sigma X^\intercal)$$

Instead of using the model above, we can use a feature transformation of X, $\phi(X)$

$$y = \phi(X)\beta + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I),\ \beta \sim \mathcal{N}(0, \Sigma)$$
$$\implies X\beta \sim \mathcal{N}(0, \phi(X)\Sigma\phi(X)^\intercal)$$

Where the dimension of $\phi(X)$, X can differ. Notice that both $X\beta, \phi(X)\beta$ are functions of X

From Bayesian Regression to GP

Question: Can we take all possible ϕ into our consideration, and perform linear regression with such feature transforms at the same time also getting rid of the model parameters?
Answer: Gaussian Process functions!
Notice that in both the cases, under gaussian prior over the parameters, the final function follows Gaussian. So what we do is assume the model to be-

$$y = f(X) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I),\ f \sim GP(0, K)$$
$$\implies f(X) \sim \mathcal{N}(0, K_{XX})$$

The kernels also have the property that for any finite dimensional X, we always have K_{XX} non-singular. The nice marginal gaussianity ensures that we can forget about the functions behaviour at other points than the datapoints we have.

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process
- Regression
- Classification
- Hyperparameter Tuning

3 Challenges of Gaussian Process

4 Summary

GP Regression

We assume the following model in absence of noise (i.e. the case when we observe exact functions f)

$$y = f(X),\ f \sim GP(0, K)$$
$$\implies f(X) \sim \mathcal{N}(0, K_{XX})$$

Let us have a training data (X, f), and some test data X_*, where the corresponding f_* is not observed. Now,

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K_{XX} & K_{XX_*} \\ K_{X_*X} & K_{X_*X_*} \end{bmatrix}\right)$$
$$\implies f_* | X_*, X, f \sim \mathcal{N}(\bar{f}_*, Cov(f_*))$$
$$\bar{f}_* = K_{X_*X} K_{XX}^{-1} f,\quad Cov(f_*) = K_{X_*X_*} - K_{X_*X} K_{XX}^{-1} K_{XX_*}$$

GP Regression

In the presence of noisy observation

$$y = f(X) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I),\ f \sim GP(0, K)$$
$$\implies f(X) \sim \mathcal{N}(0, K_{XX})$$

Now we have a training data (X, y), and some test data X_*, where the corresponding y_* is not observed. In this case we get

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K_{XX} + \sigma^2 I & K_{XX_*} \\ K_{X_*X} & K_{X_*X_*} \end{bmatrix}\right)$$
$$\implies f_* | X_*, X, y \sim \mathcal{N}(\bar{f}_*, Cov(f_*))$$
$$\bar{f}_* = K_{X_*X}(K_{XX} + \sigma^2 I)^{-1} y,\ \ Cov(f_*) = K_{X_*X_*} - K_{X_*X}(K_{XX} + \sigma^2 I)^{-1} K_{XX_*}$$

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process
- Regression
- Classification
- Hyperparameter Tuning

3 Challenges of Gaussian Process

4 Summary

The model

Let us consider binary classification. Much like Bayesian logistic regression we have the model

$$y = Bin(\sigma(f(X))), \quad f \sim GP(0, K)$$
$$\implies f(X) \sim \mathcal{N}(0, K_{XX})$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$, the softmax function.

Inference

To perform inference for a test dataset X_*, we have to follow a set of operation,

- Compute $p(f|X, y)$ i.e. the posterior distribution of latent function f given the dataset. Where

$$\begin{aligned} p(f|X, y) &\propto p(y|f)\ p(f|X) \\ &= \sigma(f)^y (1 - \sigma(f))^{1-y} p(f|X) \end{aligned}$$

- Compute $p(f_*|X_*, X, y) = \int p(f_*|X_*, X, y)\ p(f|X, y)\ df$
- Compute $p(y_* = 1|X_*, X, y) = \int p(y_* = 1|f_*)\ p(f_*|X_*, X, y)\ df_*$

Inference

- The main problem in the process is finding $p(f|X, y)$ as it has two parts among which one part is Gaussian and the other part is not.
- If by any method we can approximate this posterior by a Gaussian one, then from that we can perform the subsequent steps easily. The last step can be overcome by simulating from $p(f_*|X_*, X, y)$ and performing average of $p(y_* = 1|f_*)$ over this simulated f_*
- There are a variety of approximation method. Some of which are Laplace approximation, Variational inference.

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process
- Regression
- Classification
- Hyperparameter Tuning

3 Challenges of Gaussian Process

4 Summary

Marginal log-likelihood

Remember that the covariance kernel has some hyperparameters which we might want to choose properly. Let us now go back to the GP regression setup. For a training data (X, y) we can calculate the marginal log-likelihood by marginalizing over the latent function f

$$p(y|X) = \int p(y|f, X)p(f|X)df$$

$$\textit{where} \;\; f|X \sim \mathcal{N}(0, K), \;\; y|f \sim \mathcal{N}(f, \sigma^2 I)$$

$$\implies \log p(y|X) = -\frac{1}{2}y^\intercal(K + \sigma^2 I)^{-1}y - \frac{1}{2}log|K + \sigma^2 I| - \frac{n}{2}log(2\pi)$$

Where we call the first part **data fit**, and the second part is called **model complexity**.

Optimizing marginal log-likelihood

To find the optimal value of the hyperparameters by maximizing the log-likelihood, we use the gradient descent algorithm on the negetive log-likelihood. For that we calculate the gradient of the loss. Denoting $\bar{K} := K + \sigma^2 I$

$$\mathcal{L} := -\log p(y|X) \propto y^\intercal \bar{K}^{-1} y + log|\bar{K}| + n\log(2\pi)$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = -y^\intercal \bar{K}^{-1} \left(\frac{\partial \bar{K}}{\partial \theta}\right) \bar{K}^{-1} y + Tr\left(\bar{K}^{-1} \frac{\partial \bar{K}}{\partial \theta}\right)$$

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process

3 Challenges of Gaussian Process
- The Marginal log-likelihood revisited
- BlackBox Matrix Multiplication (GPyTorch)

4 Summary

Complexity of calculation

Remember the gradient of loss function

$$\frac{\partial \mathcal{L}}{\partial \theta} = -y^{\intercal} \bar{K}^{-1} \left(\frac{\partial \bar{K}}{\partial \theta} \right) \bar{K}^{-1} y + Tr\left(\bar{K}^{-1} \frac{\partial \bar{K}}{\partial \theta} \right)$$

Also, to monitor trajectory of training, we generally also calculate the loss function. So there are 3 major things that we need to calculate.

- $\bar{K}^{-1}y$ inverse of the kernel matrix which has dimension $N \times N$, N being the size of dataset.
- $|\bar{K}|$ the determinant of the kernel matrix

Cholesky decomposition

Till now people had been using Cholesky decomposition for this task. For any given positive definite matrix A the decomposition gives us a lower triangular matrix L which satisfies the property that $A = LL^{\intercal}$

Pros:

- The solution for L can be found by solving a set of easily numerically solvable equations giving one by one each of the elements of the lower triangular matrix.
- After finding L it is easy to calculate the inverse of A by solving another set of easily solvable equations.
- Determinant of A also becomes easy to calculate.

Cons:

- The naive implementation of Cholesky decomposition has $\mathcal{O}(N^3)$ time complexity.
- The sequence of equation can not be parallelized.

Table of Contents

1 Introduction to Gaussian Process

2 Application of Gaussian Process

3 Challenges of Gaussian Process
- The Marginal log-likelihood revisited
- BlackBox Matrix Multiplication (GPyTorch)

4 Summary

BCG algorithm (The inverse)

Gardner, Pleiss, Wilson *et al.*[2019] used a number of tricks to solve these computational challenges. If we need to calculate $\bar{K}^{-1}y$, then instead of calculating inverse and then doing the multiplication with y we can get $u \approx \bar{K}^{-1}y$ (with arbitrary precision) by solving the convex optimization problem

$$u = \underset{v}{\operatorname{argmin}} \frac{1}{2} v^\intercal \bar{K} v - v^\intercal y$$

The algorithm only needs a BlackBox Matrix Multiplication module that does all the matrix multiplication in a heavily optimized and parallelized way.

BCG algorithm (The inverse)

- The solver now performs gradient descent algorithm to find the minima which can be parallelized (also using GPU).
- To achieve a certain level of reasonable accuracy, the runtime of the algorithm is much less compare to computing Cholesky.
- It only requires some basic Matrix multiplication for which heavily optimized distributed algorithm exists.

BCG algorithm (The Trace)

To find the trace term in the log-likelihood, we notice the fact that if z is a random variable with $\mathbb{E}[z] = 0$ and $\mathbb{E}[zz^\intercal] = I$ then

$$Tr(A) = \mathbb{E}(z^\intercal Az)$$

So one can simulate $z_1, z_2, ..., z_t$ (maybe from $\mathcal{N}(0, I)$) and compute

$$Tr(A) \approx \frac{1}{t} \sum_{i=1}^{t} z_i{}^\intercal Az_i$$

BCG algorithm (The Trace)

So in our case we can compute

$$Tr\left(\bar{K}^{-1}\frac{\partial \bar{K}}{\partial \theta}\right) \approx \frac{1}{t}\sum_{i=1}^{t}\left(z_i{}^{\intercal}\bar{K}^{-1}\right)\left(\frac{\partial \bar{K}}{\partial \theta}z_i\right)$$

Where the first term can be computed using BCG and transposing the result, and the second term is just a matrix multiplication. Thus the final algorithm becomes

- Get $z_1, z_2, ..., z_t$ by simulation
- Compute $[u_0, u_1, u_2, ..., u_t] = BCG(y, z_1, z_2, ..., z_t)$
- calculate the trace and inverse multiplications by using u_i

BCG algorithm (The Trace)

- Gaussian Process gives us a non-parametric model that at the same time is powerful to fit any dataset, as well as of Bayesian nature.
- Both Linear regression and Neural Network fully connected layers can be regarded as a special case of GP, with asymptotic properties of GP. So, we can use GP over all of these kind of models.
- While the computation cost had been a hinderence for using GP on practical datasets of reasonable size, recent research has enabled us to apply GP on datasets of size over 1M

GPyTorch is an extension of PyTorch which lets you fit Gaussian process regression and classification on huge datasets and also uses PyTorch models (even Neural networks) very easily.

Thank you!