Final Project -STAT431-

Taiga Hasegawa

2019/4/29

INTRODUCTION

Unlike the classical machine learning algorithm, bayesian methods do not attempt to get the best fit by convex optimization. Instead they make the posterior inference by incorporating prior knowledge and come with uncertainty quantification. We focused on Gaussian Process, which was kernel-based fully Bayesian algorithm. It is thought to be useful and flexible because it does not overfit and can produce uncertainty quantification.

GAUSSIAN PROCESS

What is gaussian process

Assume stochastic process $x_1, x_2, \cdots x_N : x_n \in X$, indexed by elements from some set X. A Gaussian Process defines a **distribution over function** $p(f)$, where $f : X \to \mathbb{R}$ is a function mapping some input X to $\mathbb{R}$ and $p(f)$ is a Gaussian process only if for any finite subset $\{x_1 \cdots x_N \in X\}$, the marginal distribution over the finite subspace $p(f)$ with f:=$(f(x_1), \cdots, f(x_N))$ is multivariate Gaussian.

Multivariate gaussian distribution of $f(x_1), \cdots, f(x_N)$:$p(f)$ has the form like below.

$$p(f) = \begin{bmatrix} f(x_1) \\ \cdot \\ \cdot \\ f(x_N) \end{bmatrix} \sim N \left(\begin{pmatrix} m(x_1) \\ \cdot \\ \cdot \\ m(x_N) \end{pmatrix}, \begin{pmatrix} k(x_1, x_2) & \cdot & \cdot & k(x_1, x_N) \\ \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot \\ k(x_N, x_1) & \cdot & \cdot & k(x_N, x_N) \end{pmatrix} \right)$$

Gaussian process is parameterized by a mean function , $\mu(x)$ and a covariance function, or kernel $K(x, x')$. Using notation, it can be written like

$$f(\cdot) \sim GP(m(\cdot), k(\cdot, \cdot))$$

Intuitively, one can think of a function $f(\cdot)$ drawn from Gaussian process as an extremely high-dimensional vector from an extremely high-dimensional multivariate Gaussian. Using the marginalization property for multivariate Gaussians, we can obtain the marginal multivariate Gaussian density corresponding to any finite subcollection of variables. This is why, Gaussian process can be thought as probability distribution over functions with infinite domains.

In general , any real-valued function $m(\cdot)$ is acceptable and

$$K = \begin{pmatrix} k(x_1, x_2) & \cdot & \cdot & k(x_1, x_N) \\ \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot \\ k(x_N, x_1) & \cdot & \cdot & k(x_N, x_N) \end{pmatrix}$$

must be positive semi definite because this condition is necessary for covariance matrix of Gaussian distribution.

Kernel function

What kind of kernel density is used for Gaussian process. The most popular one would be squared exponential kernel function, defined as

$$k(x, x') = exp(-\kappa^2 \frac{1}{2\tau^2} ||x - x'||_F^2)$$

where τ, κ^2 is hyper parameter. Here, tau controls the correlation length and κ accounts for the height of oscillations. In this case, for any pair of $x, x' \in X$, $f(x), f(x')$ tend to have high covariance when x and x' are nearby in the input space(i.e. $||x - x'|| \approx 0, exp(-\frac{1}{2\tau^2}||x - x'||^2 \approx 1)$). On the other hand, when x and x' are far apart, $f(x)$ and $f(x')$ tend to have low covariance.

There are other kernel functions such as Matern process, Ornstein-Uhlenbeck (OU) process and Wiener process. It is also possible to make new kernel functions by adding or multiplying several kernel functions.

Gaussian process regression model

Let $X, Y = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ be a training data of i.i.d. examples from some unknown distribution. In the Gaussian process regression model, we use the following model.

$$Y = f(X) + \epsilon \qquad \epsilon \sim i.i.d. N(0, \sigma^2)$$

Like in Bayesian regression, we assume a prior distribution over function, or Gaussian process prior is

$$f(X) \sim N(0, K)$$

, where K is some kernel function.

Then posterior distribution of $f(X)$ is

$$f(X)|X, Y \sim N(\mu_n, K_n)$$

where

$$\mu_n = (K(X, X)^{-1} + \sigma^2 I_N)^{-1}(\sigma^{-2} y)$$
$$K_n = (K(X, X)^{-1} + \sigma^{-2} I_N)^{-1}$$

I_N is the N dimentional identity matrix.

Now let $X^*, Y^* = \{(x_*^{(i)}, y_*^{(i)})\}_{i=1}^{N_*}$ be test data drawn from the same unknown distribution as training data. For any function $f(X^*)$ drawn from our zero-mean Gaussian process prior with covariance function k($\cdot$, $\cdot$), the marginal distribution over any set of input points belonging to X must have a joint multivariate Gaussian distribution. So, we can write

$$\begin{bmatrix} f(X) \\ f^*(X^*) \end{bmatrix} |X, X^* \sim N(0, \begin{pmatrix} K(X, X) & K(X, X^*) \\ K(X^*, X) & K(X^*, X^*) \end{pmatrix})$$

For i.i.d. noise,

$$\begin{bmatrix} \epsilon \\ \epsilon^* \end{bmatrix} \sim N(0, \begin{pmatrix} \sigma^2 I_N & 0 \\ 0 & \sigma^2 I_N \end{pmatrix})$$

Because the sum of independent Gaussian random variables are also Gaussian, the conditional distribution of Y and Y^* is

$$\begin{bmatrix} Y \\ Y^* \end{bmatrix} |X, X^* = \begin{bmatrix} f(X) \\ f(X^*) \end{bmatrix} + \begin{bmatrix} \epsilon \\ \epsilon^* \end{bmatrix} \sim N(0, \begin{pmatrix} K(X, X) + \sigma^2 I_N & K(X, X^*) \\ K(X^*, X) & K(X^*, X^*) + \sigma^2 I_{N_*} \end{pmatrix})$$

From this multivariate gaussian distribution, we can know that

$$Y^*|X, X^*, f \sim N(0, K(X^*, X^*) + \sigma^2 I_{N_*})$$

Consequently, posterior predictive distribution is

$$p(Y^*|Y,X,X^*) = \int p(Y^*|X,X^*,f)p(f(X)|X,Y) = N(\mu^*,\Sigma^*) \quad \cdot\cdot \quad (1)$$

where

$$\mu^* = K(X,X^*)(K(X,X)+\sigma^2 I)^{-1}Y$$
$$\Sigma^* = K(X^*,X^*)+\sigma^2 I_{N_*} - K(X^*,X)(K(X,X)+\sigma^2 I_N)^{-1}K(X,X^*)$$

Gaussian process classification model

Connection between Gaussian process regression and Bayesian regression

We use the same notations as above for train data and test data. First we look at Bayesian regression. We assume D degree polynomial function is $\phi()$ and coefficient parameter is β in this section. So regression equation is $Y = \beta^T X + \epsilon$, where $\epsilon \sim N(0, \frac{1}{\tau})$

$$Y|X,\beta \sim N(\beta^T\phi(X), \lambda^{-1})$$

The prior distribution for β is assumed to be

$$\beta \sim N(0, \Lambda^{-1})$$

where λ and Λ is precision. In this case, the posterior predictive distribution is

$$y^*|y,X^*,X \sim N(\mu_*, \lambda_*^{-1})$$

where

$$\mu_* = \phi(X^*)^T\Lambda^{-1}\phi(X)(\lambda^{-1}I + \phi(X)\Lambda^{-1}\phi(X)^T)Y$$

and

$$\lambda_*^{-1} = \lambda^{-1} + \phi(X^*)^T(\lambda^{-1}I + \phi(X)\Lambda^{-1}\phi(X)^T)\phi(X^*)$$

If we define $\phi(X)\Lambda^{-1}\phi(X)^T = K(X,X)$ and $\phi(X^*)\Lambda^{-1}\phi(X) = K(X^*,X)$, the above μ_* and λ_*^{-1} can be written by using this $K(X_i,X_j)$.

This means that we can do feature engineering not by using ϕ but by using $K(X_i,X_j)$. ϕ corresponding to the squared exponential kernel function is infinite degree polynomial function.

Connection between Gaussian process and neural network

As we have seen the connection of polynomial function and kernel function, there are also kernel functions which are connected with nonlinear function with infinite number of hidden units.

$$K_{ReLU}(X_i,X_j) = \frac{1}{\pi}||X_i||||X_j||\{sin\theta + (\pi-\theta)cos\theta)\} \quad \cdot\cdot \quad (2)$$

where

$$\theta = cos^{-1}(\frac{X_i^T X_j}{||X_i||||X_j||})$$

First let's consider the neural network with only single hidden layer. The ith component of network output z_i is computed as

$$h_j(X) = \sigma(\sum_{k=1}^{D} W_{j,k}^0 X_k + b_j^0)$$

$$z_i(X) = \sum_{j=1}^{N} W_{ij}^1 h_j(X) + b_i^1$$

W and b is the parameter for neural network and is assumed to follow the i.i.d normal distribution. The prior of W and b is $p(W) \sim N(0, \lambda_w^{-1}), p(b) \sim N(0, \lambda_b^{-1})$. Because each parameter is independent, h_j is also independent. As $N \to \infty$, $h_j(X)$ asymtomically follows the normal distribution by central limit theorem. Becaues b^1 also follows the normal distribution, conseqently z also follows normal distribution. Therefore, we can write z in Gaussian process with mean μ^1 and covariance K^1. Because the parameters have zero mean, we have $\mu^1 = 0$ and kernel function is

$$K(X, X') = E[z_i(X), z_i(X')] = \lambda_b^{-1} + \lambda_w^{-1} E[h_i(X) h_i(X')] = \lambda_b^{-1} + \lambda_w^{-1} C(X, X')$$

where we have introduced $C(X, X')$; it is obtained by integraing against the distribution of W^0, b^0. Note that, as any two z_i, z_j for $i \neq j$ are joint Gaussian and have zero covariance.

The argument of single hidden layer can be extended to deeper layers by induction. We proceed by taking the hidden layer width to be infinite succession ($N_1 \to \infty, N_2 \to \infty$etc.) as we continue with the induction.

Suppose that z_j^{l-1} is a GP, identical and independent for every j and hence $h_j^l(X)$ are also independent and identically distributed. After $l-1$ hidden units , the network computes

$$z_i^l(X) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l h_j^l(X), \qquad h_j^l = \sigma(z_j^{l-1}(X))$$

As before, z_i^l will have joint multivariate Gaussian distribution and $z_i^l \sim GP(0, K^l(X, X'))$ where

$$K^l(X, X') = E(z_i^l(X), z_i^l(X')) = \lambda_b^{-1} + \lambda_w^{-1} E[\sigma(z_i^{l-1}(X))\sigma(z_i^{l-1}(X'))]$$

In shorthand, we can rewrite this formula using F whose form only depends on the nonlinearity σ.

$$K^l(X, X') = \lambda_b^{-1} + \lambda_w^{-1} F_\sigma(K^{l-1}(X, X\prime), K^{l-1}(X, X), K^{l-1}(X', X'))$$

If we choose ReLU function as nonlinearity, the kernel function is

$$K^l(X, X') = \lambda_b^{-1} + \frac{\lambda_w^{-1}}{2\pi} \sqrt{K^{l-1}(X, X) K^{l-1}(X'X')} \{sin\theta^{l-1} + (\pi - \theta^{l-1}) cos\theta^{l-1}\}$$

from equation (2).

$$\theta^l = cos^{-1}(\frac{K^l(X, X\prime)}{\sqrt{K^l(X, X) K(X'X')}})$$

So only we have to do is to calculate this kernel function and use equation (1). Then we can get the prediction of deep learning only by calculating matrix operation.

Connection with time series

Gaussian Process in JAGS

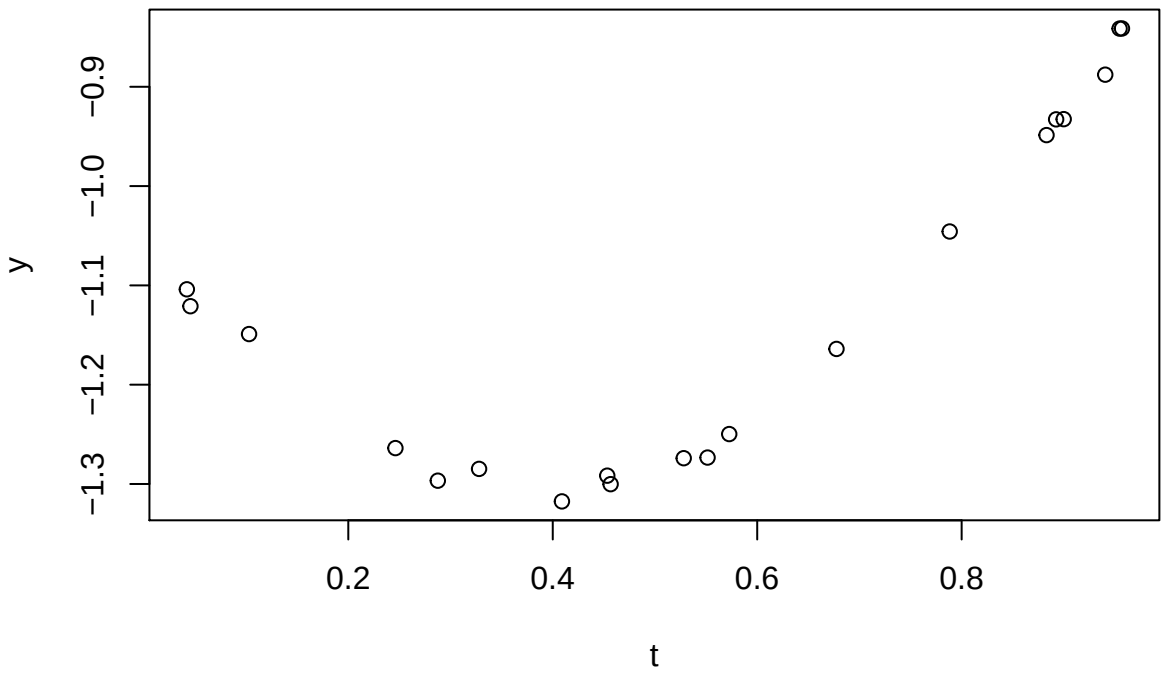
Finally, let's see how to implement Gaussian Process in JAGS. We simulated the multivariate normal distribution and predicted it using Gaussian process.

```r
library(rockchalk)
# Notation:
# y(t): Response variable at time t, defined on continuous time
# y: vector of all observations
# alpha: Overall mean parameter
# sigma: residual standard deviation parameter (sometimes known in the GP world as the nugget)
# rho: decay parameter for the GP autocorrelation function
# tau: GP standard deviation parameter

# Likelihood:
# y ~ MVN(Mu, Sigma)
# where MVN is the multivariate normal distribution and
# Mu[t] = alpha
# Sigma is a covariance matrix with:
# Sigma_{ij} = tau^2 * exp( -rho * (t_i - t_j)^2 ) if i != j
# Sigma_{ij} = tau^2 + sigma^2 if i=j
# The part exp( -rho * (t_i - t_j)^2 ) is known as the autocorrelation function


# Simulate data ------------------------------------------------------------

# Some R code to simulate data from the above model
T = 20 # can take to T = 100 but fitting gets really slow ...
alpha = 0
sigma = 0.01
tau = 1
rho = 1
set.seed(123)
t = sort(runif(T))
Sigma = sigma^2 * diag(T) + tau^2 * exp( - rho * outer(t,t,'-')^2 )
y = mvrnorm(1,rep(alpha,T), Sigma)
plot(t,y)
```



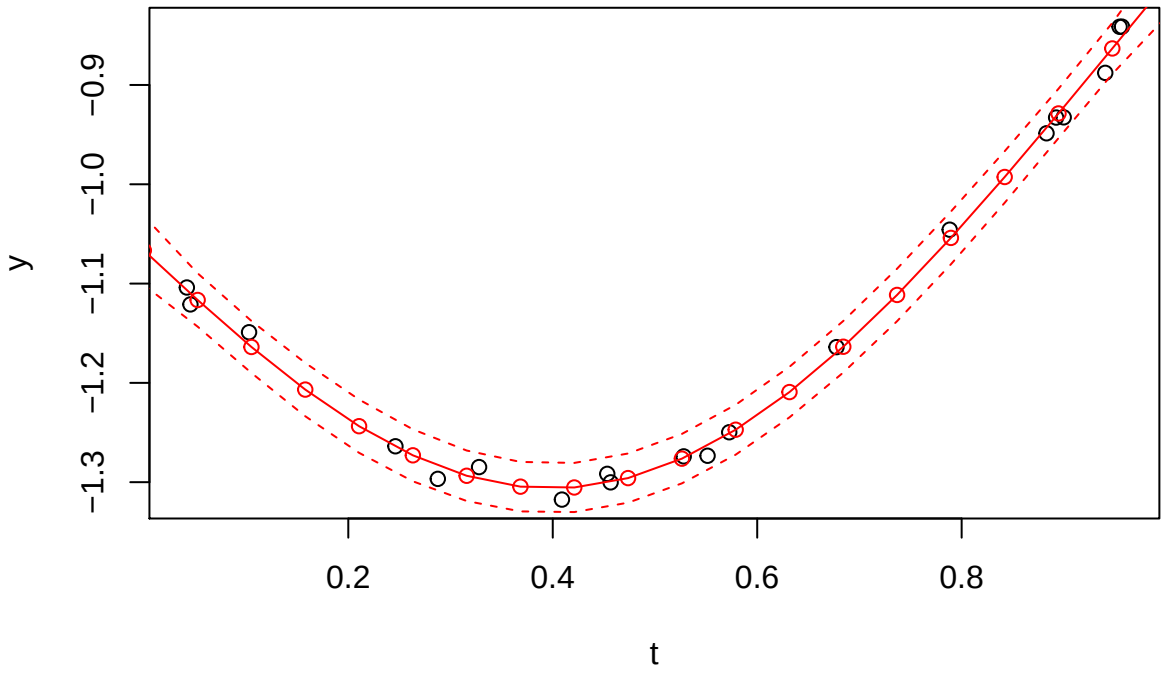
```
## Loading required package: coda
## Linked to JAGS 4.3.0
## Loaded modules: basemod,bugs
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 4
##    Total graph size: 989
##
## Initializing model
```

```
alpha=x[[1]][,1]
rho=x[[1]][,2]
sigma=x[[1]][,3]
tau=x[[1]][,4]
# Now create predictions
T_new = 20
t_new = seq(0,1,length=T_new)
Mu = rep(mean(alpha), T)
Mu_new = rep(mean(alpha), T_new)
Sigma_new = mean(tau)^2 * exp( -mean(rho) * outer(t, t_new, '-')^2 )
Sigma_star = mean(sigma)^2 * diag(T_new) + mean(tau)^2 * exp( - mean(rho) * outer(t_new,t_new,'-')^2 )
Sigma = mean(sigma)^2 * diag(T) + mean(tau)^2 * exp( - mean(rho) * outer(t,t,'-')^2 )

# Use fancy equation to get predictions
pred_mean = Mu_new + t(Sigma_new)%*%solve(Sigma, y - Mu)
pred_var = Sigma_star - t(Sigma_new)%*%solve(Sigma, Sigma_new)

# Plot output
plot(t,y)
points(t_new, pred_mean, col='red')
lines(t_new, pred_mean, col='red')

pred_low = pred_mean - 1.95 * sqrt(diag(pred_var))
pred_high = pred_mean + 1.95 * sqrt(diag(pred_var))
lines(t_new, pred_low, col = 'red', lty = 2)
lines(t_new, pred_high, col = 'red', lty = 2)
```


y
−1.3
−1.2
−1.1
−1.0
−0.9
0.2
0.4
0.6
0.8
t