

# Final Project -STAT431-

*Taiga Hasegawa (taigah2) Seung ji Nam (sjnam2) Yinghao Li (yinghao3)*

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

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

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

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

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

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  $\tau, \kappa^2$  is hyper parameter. Here,  $\tau$  controls the correlation length and  $\kappa$  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.

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

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

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

$I_N$  is the  $N$  dimensional 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

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

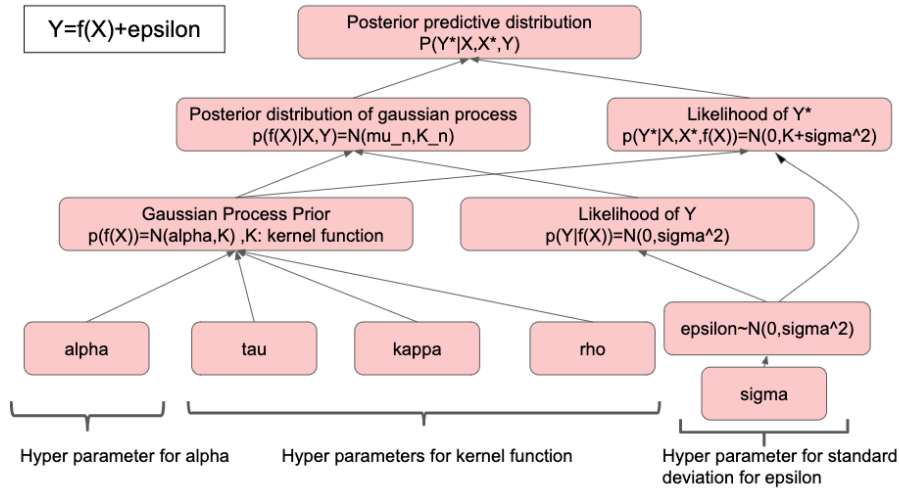
Consequently, posterior predictive distribution is

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

where

$$\begin{aligned} \mu^* &= K(X, X^*)(K(X, X) + \sigma^2 I)^{-1}Y \\ \Sigma^* &= K(X^*, X^*) + \sigma^2 I_{N^*} - K(X^*, X)(K(X, X) + \sigma^2 I_N)^{-1}K(X, X^*) \end{aligned}$$

The model architecture is now Hierarchical Models like below.



## Gaussian process classification model

Let's assume binary classification case.  $X \in \mathbb{R}^p$ ,  $Y \in 0, 1$  is train data. We can assume that the underlying model for the probability of  $Y = 1$  at each target point  $X$  is  $\eta(X) = p(Y = 1|X)$ . So we can model  $Y$  using a Bernoulli distribution  $Y \sim \text{Ber}(\eta(X))$ .

Although we can choose many functional forms for  $\eta(x)$ ,  $u(x) = \phi(\eta(x)) = X^T \beta$  is chosen in linear classification model.  $u(X) = \phi(\eta(x))$  can be logit ( $\log(\frac{\eta(x)}{1-\eta(x)})$ ) or other functions like probit. From some simple derivations, it can be rewritten as

$$p(Y|X) = \eta(x) = \frac{\exp(X^T \beta)}{1 + \exp(X^T \beta)} = \frac{\exp(u(X))}{1 + \exp(u(X))} \quad \cdot \cdot (1)$$

In the gaussian classification model, we assume the Gaussian process prior over a continuous latent function  $u(x)$ .

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

The posterior of  $u$  can be written as

$$p(u(X)|X, Y) \propto p(u(X))p(Y|X)$$

Combining the GP prior over  $u$  and the conditional model in (1) yields the more specific expression

$$\log p(u|X, Y) = \sum_{i=1}^n [u(x_i) - \log \sum_y \exp(u(x_i))] - \frac{1}{2} u(X)^T K^{-1} u(X) + \text{const}$$

Following a Bayesian approach, the posterior predictive distribution of a label for a new observation  $X^*$  is

$$p(y|X, y, x) = \int p(y * |u(X))p(u(X)|X, Y)du$$

## 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  $\beta$  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  $\beta$  is assumed to be

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

where  $\lambda$  and  $\Lambda$  is precision. In this case, the posterior predictive distribution is

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

where

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

and

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

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  $\mu_*$  and  $\lambda_*^{-1}$  can be written by using this  $K(X_i, X_j)$ .

This means that we can do feature engineering not by using  $\phi$  but by using  $K(X_i, X_j)$ .  $\phi$  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

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

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

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

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

$W$  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  $N \rightarrow \infty$ ,  $h_j(X)$  asymptotically follows the normal distribution by central limit theorem. Because  $b^1$  also follows the normal distribution, consequently  $z$  also follows normal distribution. Therefore, we

can write  $z$  in Gaussian process with mean  $\mu^1$  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 integrating 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 ( $N_1 \rightarrow \infty, N_2 \rightarrow \infty$  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

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

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  $\sigma$ .

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

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).

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

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.

## Linear-Gaussian Time Series Model

We did not include the topic about time series in our proposal, but we decided to research about time series after Professor Park lecturing it as an extra topic. We think that time series is an important topic which has connection to bayesian analysis and gaussian process.

Time series is a series of data points listed in time order. It is used in many fields, and some are directly related to our daily life, such as weather prediction and earthquake prediction. It tends to exhibit correlations related to its temporal structure, so many models that can be used to analyze time series have been developed. Models provide better understanding of what was happening when the data was being collected, and models are also useful in making predictions about the future. When doing time series analysis, We can find the best available model that represents the system, in which the data is collected, by “optimising a cost function such as the likelihood” (Roger, 1).

Linear-Gaussian Time Series Model:

$$f(t) \sim GP(m(t), k(t, t'))$$

It is a Gaussian Process with a specific covariance function  $k(\cdot, \cdot)$  and time as the index set of the Gaussian Process. Some popular examples are linear auto-regressive (AR) model, linear auto-regressive moving-average (ARMA) model and linear state-space model.

Nonlinear AR(n) Model With Gaussian Process:

$$f(Y) \sim GP(m(Y), k(Y, Y')),$$

$$g_t = f(Y_{t-1}),$$

$$y_t|g_t \sim p(y_t|g_t, \theta)$$

$Y_{t-1} = \{y_{t-1}, \dots, y_{t-n}\}$ .  $\theta$  are the parameters of AR(n) model.

$y_t$  depends on  $g_t$  and  $g_t$  depends on  $\{y_{t-n}, \dots, y_{t-1}\}$ . Therefore, we first draw a function from  $f(Y)$  and draw the first  $n$  observations to get the distribution of  $g_n$ .  $Y_{n-1} \sim p(Y_{n-1}) = p(y_0, \dots, y_{n-1})$ ,  $g_n = f(Y_{n-1})$ , then the rest can be drawn sequentially. Exact inference is possible for this model if we choose a conjugate likelihood, for instance  $y_t|g_t \sim N(y_t|g_t, \theta)$ .

## 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.

```
# 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

#JAGS model
model
{
  # Likelihood
  y ~ dmnorm(Mu, Sigma.inv)
  Sigma.inv <- inverse(Sigma)
  # Set up mean and covariance matrix
  for(i in 1:T) {
    Mu[i] <- alpha
    Sigma[i,i] <- pow(sigma, 2) + pow(tau, 2)
    for(j in (i+1):T) {
      Sigma[i,j] <- pow(tau, 2) * exp( - rho * pow(t[i] - t[j], 2) )
      Sigma[j,i] <- Sigma[i,j]
    }
  }
  alpha ~ dnorm(0, 0.01)
  sigma ~ dunif(0, 10)
```

```

tau ~ dunif(0, 10)
rho ~ dunif(0.1, 5)
}

# Simulate data -----
library(rockchalk)
# 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)

## 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)

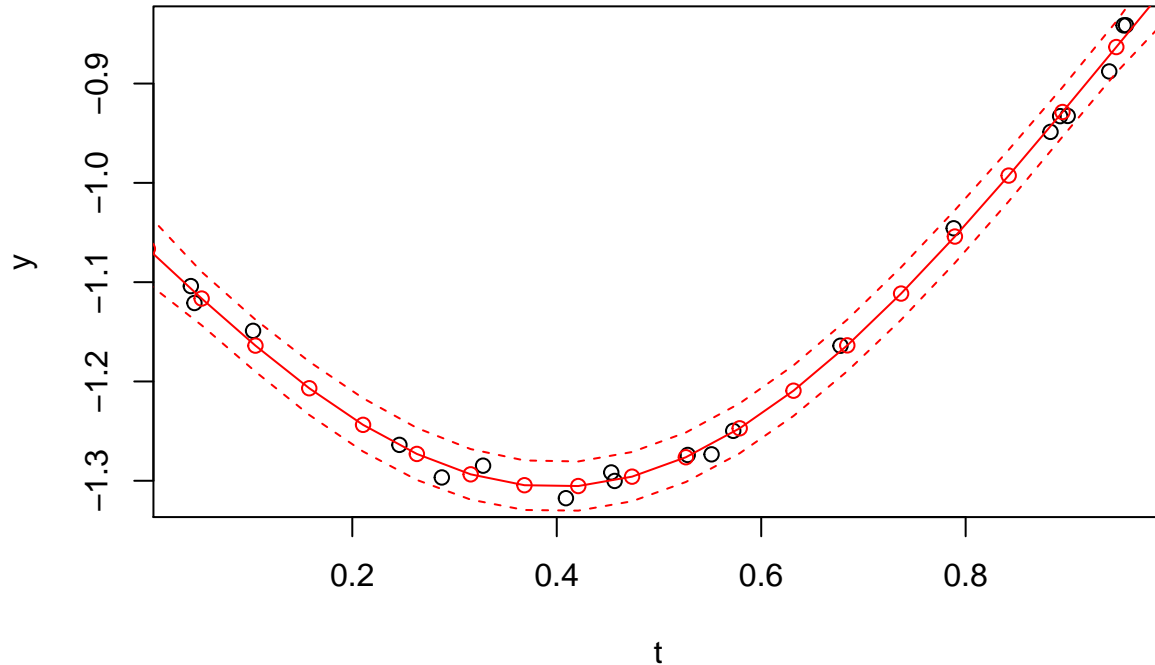
# Plot output
plot(t,y)
points(t_new, pred_mean, col='red')
lines(t_new, pred_mean, col='red')

```

```

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)

```



## 4. Relevant references

Carl Edward Rasmussen and Chris Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006.

Titsias, M. K., & Lawrence, N. D. (n.d.). Bayesian Gaussian process latent variable model. Journal of Machine Learning Research, 9, 844–851.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, Jascha Sohl-Dickstein, Deep Neural Networks as Gaussian Processes, Cornell University, Mar 2018.

Shiwei Lan, CS101C: Bayesian Analysis Bayesian Non-parametric Modeling Gaussian Process, CMS, Cal-Tech, 2017

Chuong B. Do, Gaussian processes, 2008

Roger Frigola-Alcalde, Bayesian Time Series Learning with Gaussian Processes, Cambridge, Aug. 2015.

Roberts, Osborne, et al., Gaussian Processes for Timeseries Modelling, Oxford, 22 Jul. 2012.

Hannes Nickish, Carl Edward Rasmussen, Approximations for Binary Gaussian Process Classification, Journal of Machine Learning Research 9, 2008

Randford M. Neal, Regression and Classification using gaussian process priors, Bayesian statistics 6.