Random Fourier Features for Gaussian Process Model

Tetsuya Ishikawa
tiskw111@gmail.com

Abstract

This article describes the procedure for applying random Fourier features [1] to Gaussian process model [2]. This makes it possible to speed up the training and inference of Gaussian process model, and to apply the model to large-scale data.

Gaussian process model [2] is one of the supervised machine learning frameworks designed on a probability space, and is widely used for regression and classification tasks, as well as support vector machine and random forest. The major difference between Gaussian process model and other machine learning models is that Gaussian process model is a *stochastic* model. In other words, since Gaussian process model is formulated as a stochastic model, it can provide not only the predicted value but also a measure of uncertainty for the prediction. This is a very useful property that can improve the explainability of machine learning model.

On the other hand, Gaussian process model is also known for its high computational cost of training and inference. If the total number of training data is $N \in \mathbb{Z}^+$, the computational cost required for training is $O(N^3)$, and the computational cost required for inference is $O(N^2)$, where O is *Bachmann–Landau notation*. The problem is that the computational cost is given by a power of the total number of training data N, which can be an obstacle when applying the model to large-scale data. This comes from the fact that Gaussian process model has the same mathematical structure as kernel methods, in other words, the kernel support vector machine also has the same problem.

One of the methods to speed up kernel methods is random Fourier features [1] (hereinafter abbreviated as RFF). This method can significantly reduce the computational cost while keeping the flexibility of kernel methods by approximating a kernel function as an inner product of finite dimensional vectors. Specifically, the computational cost required for training can be reduced to $O(ND^2)$, and the amount of calculation required for inference can be reduced to $O(D^2)$, where $D \in \mathbb{Z}^+$ is a hyperparameter of RFF and can be specified independently of the total number of training data N. Since Gaussian process model has the same mathematical structure as kernel methods, RFF can be applied to Gaussian process model as well. This evolves Gaussian process model into a more powerful, easy-to-use, and highly reliable ML tool.

However, when applying RFF to Gaussian process model, some mathematical techniques are required that are not straightforward. Unfortunately, there seems to be no article in the world that mentions its difficulties and solutions, so I decided to leave this document. If you prefer the Japanese version of this document, see the repository [3].

1 Gaussian Process Model Revisited

This section gives an overview of Gaussian process model. Unfortunately, this section can not cover details such as the formulation and derivation of Gaussian process models due to the limitation of pulp, so if you are interested in the details, please refer [2].

Let $\mathscr{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ be training data, and $\sigma \in \mathbb{R}^+$ be a standard deviation of the label observation error, where $\boldsymbol{x}_n \in \mathbb{R}^M$, $y_n \in \mathbb{R}$. Gaussian process model describes the prediction as a probability variable that follows the normal distribution. If the test date is $\boldsymbol{\xi} \in \mathbb{R}^M$, the expectation of the prediction is given by:

$$m(\boldsymbol{\xi}) = \widehat{m}(\boldsymbol{\xi}) + \left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T} \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}(\boldsymbol{\xi}), \tag{1}$$

and the covariance of the test data $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2 \in \mathbb{R}^M$ is given by:

$$v(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) = k(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) - \boldsymbol{k}(\boldsymbol{\xi}_1)^\mathsf{T} \left(\boldsymbol{K} - \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}(\boldsymbol{\xi}_2), \tag{2}$$

where the function $k : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ is a kernel function, the matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ is a kernel matrix defined as

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}, \tag{3}$$

and the vector function $\boldsymbol{k}(\boldsymbol{\xi}) : \mathbb{R}^M \to \mathbb{R}^N$ and the vector $\boldsymbol{y} \in \mathbb{R}^N$ is defined as

$$\boldsymbol{k}(\boldsymbol{\xi}) = \begin{pmatrix} k(\boldsymbol{\xi}, \boldsymbol{x}_1) \\ \vdots \\ k(\boldsymbol{\xi}, \boldsymbol{x}_N) \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \tag{4}$$

respectively. Also, $\widehat{m}(\boldsymbol{\xi})$ is the prior distribution of the prediction, and $\widehat{\boldsymbol{m}} = (\widehat{m}(\boldsymbol{x}_1), \ldots, \widehat{m}(\boldsymbol{x}_N))^\mathsf{T}$ is the prior distribution of the predicted values of the training data. If you don't need to set prior distribution, it's common to set $\widehat{m}(\cdot) = 0$ and $\widehat{\boldsymbol{m}} = \boldsymbol{0}$.

You can compute the variance of the prediction of the test data $\boldsymbol{\xi}$ by substituting $\boldsymbol{\xi}_1 = \boldsymbol{\xi}_2 = \boldsymbol{\xi}$ into the equation (2),

$$v(\boldsymbol{\xi}, \boldsymbol{\xi}) = k(\boldsymbol{\xi}, \boldsymbol{\xi}) - \boldsymbol{k}(\boldsymbol{\xi})^\mathsf{T} \left(\boldsymbol{K} - \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}(\boldsymbol{\xi}). \tag{5}$$

2 RFF Revisited

In this section, we revisit random Fourier features. Unfortunately, this article don't have enough space to explain the details as the same as the previous section, therefore if you would like to know more details, please refer to the original paper [1].

Let the function $k : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ be a kernel function. In RFF, the kernel function can be approximated as

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) \simeq \boldsymbol{\phi}(\boldsymbol{x}_1)^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{x}_2), \tag{6}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^D$ is a feature vector extracted from the data $\boldsymbol{x}$ and $D \in \mathbb{Z}^+$ is the dimension of $\boldsymbol{\phi}(\boldsymbol{x})$. Note that the approximation (6) is equivalent with

$$\boldsymbol{K} \simeq \boldsymbol{\Phi}^\mathsf{T} \boldsymbol{\Phi}, \tag{7}$$

where $\boldsymbol{K}$ is defined as the equation (3), and $\boldsymbol{\Phi}$ is defined as $\boldsymbol{\Phi} = (\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N))$. The larger the dimension D, the higher the

approximation accuracy of the equation (6), while the larger the dimension D, the greater compurational cost.

The actual function form of the feature vector $\boldsymbol{\phi}(\boldsymbol{x})$ depends on the kernel function. For example, in the case of the RBF kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\gamma \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2\right), \tag{8}$$

which is the most famous kernel function, the vector $\boldsymbol{\phi}(\boldsymbol{x})$ is given by

$$\boldsymbol{\phi}(\boldsymbol{x}) = \cos\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{u}\right), \tag{9}$$

where, the matrix $\boldsymbol{W} \in \mathbb{R}^{D\times M}$ is a random matrix in which each element is sampled from the normal distribution $\mathcal{N}(0, \frac{1}{4\gamma})$:

$$\boldsymbol{W} \sim \mathcal{N}\left(\boldsymbol{0}, \frac{1}{4\gamma}\boldsymbol{I}\right), \tag{10}$$

and the vector $\boldsymbol{u} \in \mathbb{R}^M$ is a random vector sampled from the uniform distribution over the range $[0, 2\pi)$:

$$\boldsymbol{u} \sim \mathcal{U}[0, 2\pi). \tag{11}$$

This approximation method is called random Fourier features because the feature extraction vector $\boldsymbol{\phi}(\boldsymbol{x})$ is derived from the Fourier transform of the kernel matrix $\mathcal{F}[k](\omega)$. Please refer to the original paper [1] for more details.

3 Gaussian Process Model and RFF

In this section, we apply RFF to Gaussian process model and theoretically confirm the effect of speeding up.

3.1 Computational complexity of Gaussian process model before applying RFF

First, let's check the computational cost required for training and inferring a normal Gaussian process model. As a premise, it is assumed that the number of training data $N \in \mathbb{Z}^+$ is sufficiently larger than the dimension $D \in \mathbb{Z}^+$ which is a hyperparameter of RFF. Here, the bottleneck of training computational cost is the calculation of the inverse matrix $\left(\boldsymbol{K} + \sigma^2\boldsymbol{I}\right)^{-1}$ in the formulas (1) and (2). Since the size of this matrix is $N \times N$, the computational cost for the training is $O(N^3)$. Next, the bottleneck of the inference is the matrix multiplication $\left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T}\left(\boldsymbol{K} + \sigma^2\boldsymbol{I}\right)^{-1}$ for the expectation prediction and $\boldsymbol{k}(\boldsymbol{\xi}_1)^\mathsf{T}\left(\boldsymbol{K} - \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{k}(\boldsymbol{\xi}_2)$, for the covariance prediction whose computational costs is $O(N)$ and $O(N^2)$ respectively.

3.2 Applying RFF to expectation of prediction

Now, let's apply RFF to Gaussian process model. First of all, if you substitute the RFF approximation formula (7) into the formula of expectation of the prediction in Gaussian process (1), you'll get

$$m(\boldsymbol{\xi}) = \widehat{m}(\boldsymbol{\xi}) + \left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T}\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{\xi}), \tag{12}$$

where the matrix $\boldsymbol{\Phi} \in \mathbb{R}^{D\times N}$ is defined as $\boldsymbol{\Phi} = (\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N))$. However, this has not yet speeded up. The complexity bottleneck of the above expression (12) is still the inverse of the $N \times N$ matrix $\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\right)^{-1}$.

Now we will add a bit of contrivance to the equation (12).

Lemma 1 (inversion of the matrix $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta\boldsymbol{I}$)

Let $\boldsymbol{\Phi} \in \mathbb{R}^{D\times N}$ be a real matrix and $\sigma \in \mathbb{R}^+$ be a positive real number. Then the matrices $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}_N$ and $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}_D$ are regular matrices and the equation

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}_N\right)^{-1}\boldsymbol{\Phi}^\mathsf{T} = \boldsymbol{\Phi}^\mathsf{T}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}_D\right)^{-1}, \tag{13}$$

holds, where the matrices $\boldsymbol{I}_D \in \mathbb{R}^{D\times D}$ and $\boldsymbol{I}_N \in \mathbb{R}^{N\times N}$ are D and N dimensional identity matrices, respectively.

The proof of the above lemma is given at the end of this article, and let us move on to the utilization of the lemma to the equation (12). By the lemma 1, the equation (12) can be transformed as below:

$$\begin{aligned} m(\boldsymbol{\xi}) &= \widehat{m}(\boldsymbol{\xi}) + \left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T}\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{\xi}) \\ &= \widehat{m}(\boldsymbol{\xi}) + \left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}), \end{aligned} \tag{14}$$

provided that the two $\boldsymbol{I}$'s appearing in the above equation are of different sizes.

Clever readers would have already noticed that the bottleneck has been resolved. The inverse matrix $\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\right)^{-1}$, which was the bottleneck of the expression (12), became $\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}\right)^{-1}$ in the expressions (14) where the size of the inverse matrix is $D \times D$. Normally, the dimension D is set sufficiently smaller than the number of training data N, therefore the inverse matrix $\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}\right)^{-1}$ is no longer a bottleneck of computational cost. The new bottleneck of the expression (14) is the matrix product $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$, whose computational cost is $O(ND^2)$. Therefore we've achieved a considerable speedup of the training of Gaussian process model by applying RFF because the calculational cost before RFF is $O(N^3)$.

3.3 Applying RFF to covariance of prediction

Next, we apply RFF to the covariance of the prediction (2). By substituting RFF approximation (7) into the formula of covariance prediction (2), we obtain

$$\begin{aligned} v(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) &= k(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) - \boldsymbol{k}(\boldsymbol{\xi}_1)^\mathsf{T}\left(\boldsymbol{K} - \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{k}(\boldsymbol{\xi}_2) \\ &= \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T}\left\{\boldsymbol{I} - \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^\mathsf{T}\right\}\boldsymbol{\phi}(\boldsymbol{\xi}_2), \end{aligned} \tag{15}$$

and by applying the lemma 1, we'll get

$$v(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) = \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T}\left\{\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}\right)^{-1}\right\}\boldsymbol{\phi}(\boldsymbol{\xi}_2). \tag{16}$$

The bottleneck of the expression (16) is, as well as the expectation prediction, the matrix inverse $\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\boldsymbol{I}\right)^{-1}$ is no longer a bottleneck of the computational cost, and the new bottleneck is the matrix product $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$ whose calculation cost is $O(ND^2)$.

3.4 Training and inference process

The theoretical essence of the application of RFF to Gaussian process was mentioned in the previous subsections. In this section, let me review the procedure of training and inference of Gaussian process model after applying RFF. Each procedure is described in the algorithms 1 as pseudo-code, where $\mathscr{D}$ is defined

as $\mathscr{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\sigma \in \mathbb{R}^+$ is a standard deviation of the measurement error, and $\boldsymbol{\phi}$ is a feature extraction function used in the RFF approximation.

In the algorithm 1, the function `TRAINING_GP_WITH_RFF` is designed to take the training data $\mathscr{D}$ and the standard deviation σ as arguments, and return a vector $\boldsymbol{a}$ and a matrix $\boldsymbol{C}$ that will be used in the inference of expectation and covariance, respectively. Also, the function `INFERENCE_GP_WITH_RFF` is designed to take the inference target data $\boldsymbol{\xi}$ and the training result $\boldsymbol{a}$ and $\boldsymbol{C}$, and return predicted expectation μ and predicted variance v. In the algorithm 1, we computed only the variance of prediction, however, if you want to compute the covariance of two input data $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$, then please compute $\boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T} \boldsymbol{C} \boldsymbol{\phi}(\boldsymbol{\xi}_2)$. Also, note that the prior distribution of Gaussian process model is set to 0 for the sake of simplicity in the algorithm 1.

Algorithm 1 Training of the GP model after RFF

function TRAINING_GP_WITH_RFF($\mathscr{D}$, σ)
  $\boldsymbol{y} \leftarrow \left(y_1, y_2, \ldots, y_N\right)^\mathsf{T}$
  $\boldsymbol{\Phi} \leftarrow \left(\boldsymbol{\phi}(\boldsymbol{x}_1), \boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N)\right)$
  $\boldsymbol{P} \leftarrow \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$
  $\boldsymbol{Q} \leftarrow (\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1}$
  $\boldsymbol{a} \leftarrow \boldsymbol{y}^\mathsf{T} \boldsymbol{\Phi}^\mathsf{T} Q$
  $\boldsymbol{C} \leftarrow \boldsymbol{I} - \boldsymbol{P}\boldsymbol{Q}$
  return ($\boldsymbol{a}$, $\boldsymbol{C}$)
end function

function INFERENCE_GP_WITH_RFF($\boldsymbol{\xi}$, $\boldsymbol{a}$, $\boldsymbol{C}$)
  $\boldsymbol{z} \leftarrow \boldsymbol{\phi}(\boldsymbol{\xi})$
  $\mu \leftarrow \boldsymbol{a}^\mathsf{T} \boldsymbol{z}$
  $v \leftarrow \boldsymbol{z}^\mathsf{T} \boldsymbol{C} \boldsymbol{z}$
  return (μ, v)
end function

3.5 Computational cost of training and inderence

Finally, let me summarize the computational cost of the training and inference of Gaussian process model with random Fourier features. In this section, we assume that the number of training data N is sufficiently larger than the RFF dimension D. Also the computational cost of the feature extraction function $\boldsymbol{\phi}$ is ignored because it depends on the kernel function itself.

As you can see the algorithm 1, the bottleneck of the training process is the matrix multiplication $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$ whoes computational cost is $O(ND^2)$. Note that the matrix inversion $(\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1}$ where $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$ is not a bottleneck of the training process under the assumption $N \gg D$, because its computatinal cost is $O(D^3)$.

As for the inference cost, the bottleneck is $\boldsymbol{a}^\mathsf{T} \boldsymbol{z}$ for the expectation prediction, and $\boldsymbol{z}^\mathsf{T} \boldsymbol{C} \boldsymbol{z}$ for the covariance prediction that have the computational cost $O(D)$ and $O(D^2)$ respectively. The computational cost before and after applying RFF is summarized in the table 1.

Table 1: Computational cost of the GP model before/after RFF

	Training	Inference
Before RFF	$O(N^3)$	$O(N^2)$
After RFF	$O(ND^2)$	$O(D^2)$

A Appendix 1: Another Approach

This section provides another approach to reduce the computational cost of the equation (12). This approach has almost the same computational cost as the previous approach, however, the previous approach is a bit more simple and beautiful [†1] than this approach. Therefore the readers don't have to pay much attention to this section.

A.1 Applying RFF to expectation of prediction

First, let us introduce the *matrix inversion lemma* (it's also referred to as the *binominal inverse lemma*) which is a useful formula for the expansion of a matrix inverse.

Lemma 2 (Matrix Inversion Lemma)

Let $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, $\boldsymbol{B} \in \mathbb{R}^{N\times M}$, $\boldsymbol{C} \in \mathbb{R}^{M\times N}$, and $\boldsymbol{D} \in \mathbb{R}^{M\times M}$ be real matrices. Then the equation

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{D}^{-1} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} \tag{17}$$

holds, where the matrices $\boldsymbol{A}$ and $\boldsymbol{D}$ are regular matrices.

The proof of the matrix inversion lemma is given at the end of this article, and let us move on to the utilization of the lemma to the equation (12).

By replacing $\boldsymbol{A} = \sigma^2 \boldsymbol{I}$, $\boldsymbol{B} = \boldsymbol{\Phi}^\mathsf{T}$, $\boldsymbol{C} = \boldsymbol{\Phi}$, and $\boldsymbol{D} = \boldsymbol{I}$ on the equation (17), we obtain the following equation:

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1} = \frac{1}{\sigma^2}\left(\boldsymbol{I} - \boldsymbol{\Phi}^\mathsf{T}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2 \boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}\right), \tag{18}$$

where $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} \in \mathbb{R}^{D\times D}$. Then multiply $\boldsymbol{\Phi}$ from the right to the above equation (18), and we get

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^\mathsf{T} = \frac{1}{\sigma^2}\boldsymbol{\Phi}^\mathsf{T}\left(\boldsymbol{I} - \left(\boldsymbol{P} + \sigma^2 \boldsymbol{I}\right)^{-1}\boldsymbol{P}\right). \tag{19}$$

Therefore, the expression (12) can be written as

$$m(\boldsymbol{\xi}) = \widehat{m}(\boldsymbol{\xi}) + \frac{1}{\sigma^2}\left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{S}\boldsymbol{\phi}(\boldsymbol{\xi}), \tag{20}$$

where

$$\boldsymbol{S} = \boldsymbol{I} - \left(\boldsymbol{P} + \sigma^2 \boldsymbol{I}\right)^{-1}\boldsymbol{P}. \tag{21}$$

The same as the previous approach, the bottleneck has been resolved now. The inverse matrix $\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1}$, which was the bottleneck of the expression (12), became $(\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1}$ in the expressions (20) and (21) where the size of the inverse matrix is $D \times D$. Normally, the RFF dimension D is set sufficiently smaller than the number of training data N, therefore the inverse matrix $(\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1}$ is no longer a bottleneck of computational cost. The bottleneck of the expressions (20) and (21) is the matrix product $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$, whose computational cost is $O(ND^2)$. Therefore we've achieved a considerable speedup of the training of Gaussian process model by applying RFF because the calculational cost before RFF is $O(N^3)$.

A.2 Applying RFF to covariance of prediction

Next, we apply RFF to the covariance of the prediction (2). By substituting RFF approximation (19) to the expression (2), we

[†1] The previous approach has high similarity with the linear regression, however, the approach mentioned in this section doesn't have.

obtain

$$v(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) = \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{\xi}_2) - \frac{1}{\sigma^2} \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T} \boldsymbol{P} \boldsymbol{S} \boldsymbol{\phi}(\boldsymbol{\xi}_2)$$
$$= \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\mathsf{T} \left(\boldsymbol{I} - \frac{1}{\sigma^2} \boldsymbol{P} \boldsymbol{S} \right) \boldsymbol{\phi}(\boldsymbol{\xi}_2), \tag{22}$$

The bottleneck of the expression (16) is, as well as the expectation of the prediction, the matrix product $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$ whose calculation cost is $O(ND^2)$.

The procedure of training and inference of Gaussian process model after applying RFF is described in the algorithms 2 as pseudo-code. Note that the prior distribution of Gaussian process model is set to 0 for the sake of simplicity in the algorithms 2.

Algorithm 2 Training of the GP model after RFF (2)

function TRAINING_GP_WITH_RFF($\mathscr{D}$, σ)
  $\boldsymbol{y} \leftarrow (y_1, \ldots, y_N)^\mathsf{T}$
  $\boldsymbol{\Phi} \leftarrow (\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N))$
  $\boldsymbol{P} \leftarrow \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$
  $\boldsymbol{S} \leftarrow \boldsymbol{I} - \left(\boldsymbol{P} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{P}$
  $\boldsymbol{a} \leftarrow \frac{1}{\sigma^2} \boldsymbol{y}^\mathsf{T} \boldsymbol{\Phi}^\mathsf{T} \boldsymbol{S}$
  $\boldsymbol{C} \leftarrow \boldsymbol{I} - \frac{1}{\sigma^2} \boldsymbol{P} \boldsymbol{S}$
  return ($\boldsymbol{a}$, $\boldsymbol{C}$)
end function

function INFERENCE_GP_WITH_RFF($\mathscr{D}$, σ)
  $\boldsymbol{z} \leftarrow \boldsymbol{\phi}(\boldsymbol{\xi})$
  $\mu \leftarrow \boldsymbol{a}^\mathsf{T} \boldsymbol{z}$
  $v \leftarrow \boldsymbol{z}^\mathsf{T} \boldsymbol{C} \boldsymbol{z}$
  return (μ, v)
end function

B Appendix 2: Proofs

B.1 Proof of the lemma 1

The lemma 1 is reprinted and proved.

Lemma 1 (inversion of the matrix $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}$)

Let $\boldsymbol{\Phi} \in \mathbb{R}^{D \times N}$ be a real matrix and η be a positive real number. Then the matrices $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N$ and $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D$ are regular matrices and the equation

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right)^{-1} \boldsymbol{\Phi}^\mathsf{T} = \boldsymbol{\Phi}^\mathsf{T} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right)^{-1} \tag{23}$$

holds, where the matrices $\boldsymbol{I}_D$ and $\boldsymbol{I}_N$ are D dimensional and N dimensional identity matrices, respectively.

Proof: First, the matrices $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N$ and $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D$ are positive definite, because

$$\boldsymbol{x}^\mathsf{T} \left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right) \boldsymbol{x} = \|\boldsymbol{\Phi}\boldsymbol{x}\|^2 + \eta \|\boldsymbol{x}\|^2 > 0,$$
$$\widehat{\boldsymbol{x}}^\mathsf{T} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right) \widehat{\boldsymbol{x}} = \left\|\boldsymbol{\Phi}^\mathsf{T}\widehat{\boldsymbol{x}}\right\|^2 + \eta \|\widehat{\boldsymbol{x}}\|^2 > 0,$$

holds for any non-zero real vectors $\boldsymbol{x} \in \mathbb{R}^N$ and $\widehat{\boldsymbol{x}} \in \mathbb{R}^D$. In general, the determinant of a matrix is equal to the product of its eigenvalues. Also, the eigenvalues of a positive definite matrix are always greater than zero. Therefore, the determinant of a positive definite matrix is greater than zero. Hence the matrices $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N$ and $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D$ are regular matrices, that is, these matrices are invertible.

Next, it is obvious that the equation

$$\boldsymbol{\Phi}^\mathsf{T} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right) = \left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right) \boldsymbol{\Phi}^\mathsf{T},$$

holds, because both are equivalent to $\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{\Phi}^\mathsf{T}$. You'll get the following equation by multiplying the inverse matrix $\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right)^{-1}$ to the above equation from the left:

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right)^{-1} \boldsymbol{\Phi}^\mathsf{T} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right) = \boldsymbol{\Phi}^\mathsf{T}.$$

Similarly, by multiplying the inverse matrix $\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right)^{-1}$ to the above equation from the right, you'll get

$$\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \eta \boldsymbol{I}_N\right)^{-1} \boldsymbol{\Phi}^\mathsf{T} = \boldsymbol{\Phi}^\mathsf{T} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \eta \boldsymbol{I}_D\right)^{-1}.$$

■

B.2 Proof of the matrix inversion lemma

The matrix inversion lemma is reprinted and proved.

Lemma 2 (Matrix Inversion Lemma)

Let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, $\boldsymbol{B} \in \mathbb{R}^{N \times M}$, $\boldsymbol{C} \in \mathbb{R}^{M \times N}$, and $\boldsymbol{D} \in \mathbb{R}^{M \times M}$ be real matrices. Then the equation

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{D}^{-1} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1} \boldsymbol{C}\boldsymbol{A}^{-1} \tag{24}$$

holds, where the matrices $\boldsymbol{A}$ and $\boldsymbol{D}$ are regular matrices.

Proof: The following equation holds:

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S} \\ -\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1} & \boldsymbol{S} \end{pmatrix}$$
$$= \begin{pmatrix} \boldsymbol{T} & -\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T} & \boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1} \end{pmatrix},$$

where

$$\boldsymbol{T} = \left(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}, \tag{25}$$
$$\boldsymbol{S} = \left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1}. \tag{26}$$

It is easy to verify the above equation from a direct calculation. By comparing the corresponding parts of the above block matrix, we get

$$\boldsymbol{T} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1}, \tag{27}$$
$$\boldsymbol{S} = \boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1}, \tag{28}$$
$$-\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S} = -\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1}, \tag{29}$$
$$-\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1} = -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T}, \tag{30}$$

By replacing with

$$\boldsymbol{A} \to \boldsymbol{D}^{-1}, \quad \boldsymbol{B} \to -\boldsymbol{C}, \quad \boldsymbol{C} \to \boldsymbol{B}, \quad \boldsymbol{D} \to \boldsymbol{A},$$

in the equation (27), we get the formula to be proved. ■

References

[1] A. Rahimi and B. Recht, “Random Features for Large-Scale Kernel Machines”, Neural Information Processing Systems, 2007.

[2] C. Rasmussen and C. Williams, “Gaussian Processes for Machine Learning”, MIT Press, 2006.

[3] `https://github.com/tiskw/mathematical-articles`