Random Fourier Features for Gaussian Process Model

Tetsuya Ishikawa
`tiskw111@gmail.com`

Abstract

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

The Gaussian process model [1] is one of the supervised machine learning frameworks designed on a probability space, and is widely used for regression and classification tasks, like support vector machine and random forest. The major difference between Gaussian process models and other machine learning models is that the Gaussian process model is a *stochastic* model. In other words, since the 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, the Gaussian process model is also known for its high computational cost of training and inrefence. If the total number of training data is $N \in \mathbb{Z}^+$, the computational cost required for training the Gaussian process model is $O(N^3)$, and computational cost required for inference is $O(N^2)$, where O is the *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 appliying the model to large-scale data. This comes from the fact that the Gaussian process model has the same mathematical structure as the kernel method, in other words, the kernel support vector machine also has the same problem.

One of the methods to speed up the kernel method is random Fourier features [2] (hereinafter abbreviated as RFF). This method can significantly reduces the computational cost while keeping the flexibility of the kernel method by approximating the kernel function as the inner product of finite dimensional vectors. Specifically, the compurational cost required for training can be reduced to $O(ND^2)$, and the amount of calculation required for inference can be reduced to $O(D^2)$. However, $D \in \mathbb{Z}^+$ is a hyperparameter of RFF and can be specified independently of the total number of training data N.

Since the Gaussian process model has the same mathematical structure as the kernel method, RFF can be applied to the Gaussian process model as well. This evolves the Gaussian process model into a more powerful, easy-to-use, and highly reliable ML tool.

However, when applying RFF to a Gaussian process model, some mathematical techniques are required that are not straightforward. However, unfortunately, there seems to be no articles in the world that mentions it's difficulties and solutions, so I left an explanation of the procedure.

If you preffer the Japanese version of this document, see this repository [1].

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

1 Gaussian Process Model Revisited

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

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

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

and the covariance of the test data $\boldsymbol{\xi}_1$, $\boldsymbol{\xi}_2$ is given by:

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

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 $\boldsymbol{k}(\boldsymbol{\xi}) \in \mathbb{R}^N$ 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 $\widehat{\boldsymbol{m}} = (\widehat{m}(\boldsymbol{x}_1), \ldots, \widehat{m}(\boldsymbol{x}_N))^\top$ 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),

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

2 RFF Revisited

This section, we revisit random Fourier features. Unfortunately, this article don't have enough space to explain the details, 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 the kernel function. In RFF, the kernel function can be approximated as

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

where the dimension D of the vector $\boldsymbol{\phi}(\boldsymbol{x}_1)$ is a hyperparamter of RFF. The larger the dimension D, the higher the approximation accuracy of the equation (6), while the larger the dimension D, the greater compurational cost.

For example, in the case of the RBF kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\gamma \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2\right), \tag{7}$$

which is famous as one of the kernel functions, the vector $\boldsymbol{\phi}(\boldsymbol{x})$ is given by

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \cos \boldsymbol{W}\boldsymbol{x} \\ \sin \boldsymbol{W}\boldsymbol{x} \end{pmatrix}, \tag{8}$$

where, the matrix $\boldsymbol{W} \in \mathbb{R}^{D/2 \times M}$ is a random matrix in which each element is sampled from the normal distribution $\mathcal{N}(0, \frac{1}{4\gamma})$.

3 Gaussian process model and RFF

In this section, we apply RFF to the 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 $M \in \mathbb{Z}^+$ of the input vector and dimention $D \in \mathbb{Z}^+$ which is a hyperparameter of RFF. Here, the bottleneck of training computational cost is obviously 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 training is $O(N^3)$.

Next, the bottleneck of the inference is matrix multiplications $\left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1}$ or $\boldsymbol{k}(\boldsymbol{\xi}_1)^\top \left(\boldsymbol{K} - \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}(\boldsymbol{\xi}_2)$, and either of these computational cost is $O(N)$.

3.2 Applying RFF to expectation of prediction

Now, let's apply RFF to the Gaussian process model. First of all, if you substitute the RFF approximation formula (2) into the formula of expectation of the prediction in the Gaussian process (1), you'll get

$$m(\boldsymbol{\xi}) = \widehat{m}(\boldsymbol{\xi}) + \left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\top \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\phi}(\boldsymbol{\xi}), \tag{9}$$

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 (9) is still the inverse of the $N \times N$ matrix $\left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1}$.

Now we will add a bit of contrivance to the equation (9). At first, let us introduce the *matrix inversion lemma* (it's also referred as *binominal inverse lemma*) which is a useful formula for expansion of matrix inverse.

Theorem 3.1 (*Matrix Inversion Lemma*)

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

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{D}^{-1} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} \tag{10}$$

holds, where the matrix $\boldsymbol{A}$ *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 (9).

By replacing $\boldsymbol{A} = \sigma^2 \boldsymbol{I}$, $\boldsymbol{B} = \boldsymbol{\Phi}^\top$, $\boldsymbol{C} = \boldsymbol{\Phi}$, and $\boldsymbol{D} = \boldsymbol{I}$ on the equation (10), we obtain the following equation:

$$\left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1} = \frac{1}{\sigma^2}\left(\boldsymbol{I} - \boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}\right), \tag{11}$$

where $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \in \mathbb{R}^{D \times D}$. Then multiply $\boldsymbol{\Phi}$ from the right the to the above equation (11), we get

$$\left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi} = \frac{1}{\sigma^2} \boldsymbol{\Phi}^\top \left(\boldsymbol{I} - \left(\boldsymbol{P} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{P}\right). \tag{12}$$

Therefore, the expression (9) can be written as

$$m(\boldsymbol{\xi}) = \widehat{m}(\boldsymbol{\xi}) + \frac{1}{\sigma^2}\left(\boldsymbol{y} - \widehat{\boldsymbol{m}}\right)^\top \boldsymbol{\Phi}^\top \boldsymbol{S}, \tag{13}$$

where

$$\boldsymbol{S} = \boldsymbol{I} - \left(\boldsymbol{P} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{P}. \tag{14}$$

Clever readers would have already noticed that the bottleneck has been resolved. The inverse matrix $(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1}$, which was the bottleneck of the expression (9), became $(\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1}$ in the expressions (13) and (14) 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 (13) and (14) is the matrix product $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$, whoes computational cost is $O(ND^2)$. Therefore we've achieved a considerable speedup of the training of the 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 substitute RFF approximation (12) to the expression (2), we obtain

$$\begin{aligned} v(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) &= \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\top \boldsymbol{\phi}(\boldsymbol{\xi}_2) - \frac{1}{\sigma^2} \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\top \boldsymbol{P}\boldsymbol{S}\boldsymbol{\phi}(\boldsymbol{\xi}_2) \\ &= \boldsymbol{\phi}(\boldsymbol{\xi}_1)^\top \left(\boldsymbol{I} - \frac{1}{\sigma^2}\boldsymbol{P}\boldsymbol{S}\right) \boldsymbol{\phi}(\boldsymbol{\xi}_2), \end{aligned} \tag{15}$$

The bottleneck of the expression (15) is, as the same as the expectation of the prediction, the matrix product $\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ whoes calculation cost is $O(ND^2)$.

The procedure of training and inference of the Gaussian process model after applying RFF is descrived in algorithm 1 and 2 as pseudo code. Note that the prior distribution of the Gaussian process model is set to 0 for the sake of simplicity in Algorithm 1 and 2.

Finally, the calculational cost after applying RFF is summarized in the table 1, where $N \in \mathbb{Z}^+$ is the number of training data and $D \in \mathbb{Z}^+$ is the dimension of RFF.

A Appendices

Algorithm 1: Training of the GP model after RFF

```
Data: $\mathscr{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$,  $\sigma \in \mathbb{R}^+$
Result: $\boldsymbol{c}_\mathrm{m} \in \mathbb{R}^D$,  $\boldsymbol{C}_\mathrm{v} \in \mathbb{R}^{D\times D}$
$\boldsymbol{y} \leftarrow (y_1, \ldots, y_N)^\top$
$\boldsymbol{\Phi} \leftarrow (\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N))$
$\boldsymbol{P} \leftarrow \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$
$\boldsymbol{S} \leftarrow \boldsymbol{I} - (\boldsymbol{P} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{P}$
$\boldsymbol{c}_\mathrm{m} \leftarrow \frac{1}{\sigma^2} \boldsymbol{y}^\top \boldsymbol{\Phi}^\top \boldsymbol{S}$          /* Cache for expectation */
$\boldsymbol{C}_\mathrm{v} \leftarrow \boldsymbol{I} - \frac{1}{\sigma^2} \boldsymbol{P}\boldsymbol{S}$          /* Cache for covariance  */
```

Algorithm 2: Inference of the GP model after RFF

```
Data: $\boldsymbol{\xi} \in \mathbb{R}^M$,  $\boldsymbol{c}_\mathrm{m} \in \mathbb{R}^D$,  $\boldsymbol{C}_\mathrm{v} \in \mathbb{R}^{D\times D}$
Result: $\mu \in \mathbb{R}$, $\eta \in \mathbb{R}$
$\boldsymbol{z} \leftarrow \boldsymbol{\phi}(\boldsymbol{\xi})$
$\mu \leftarrow \boldsymbol{c}_\mathrm{m} \boldsymbol{z}$          /* Inference of expectation */
$\eta \leftarrow \boldsymbol{z}^\top \boldsymbol{C}_\mathrm{v} \boldsymbol{z}$          /* Inference of covariance  */
```

A.1 Proof of matrix inversion lemma

The matrix inversion lemma is reprinted and proved.

Theorem A.1 (*Matrix Inversion Lemma*)

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

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{D}^{-1} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} \tag{16}$$

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

Proof: The following equation holds:

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

where

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

$$\boldsymbol{S} = \left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1}. \tag{18}$$

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

$$\boldsymbol{T} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1}, \tag{19}$$

$$\boldsymbol{S} = \boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1}, \tag{20}$$

$$-\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{S} = -\boldsymbol{T}\boldsymbol{B}\boldsymbol{D}^{-1}, \tag{21}$$

$$-\boldsymbol{S}\boldsymbol{C}\boldsymbol{A}^{-1} = -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{T}, \tag{22}$$

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 (19), we get the formula to be proved. ■

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

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

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.