Lab Course SS2024: Data Science in Astrophysics Gaussian Processes

Sudeshna Boro Saikia[*]

April 20, 2024

[*]sudeshna.boro.saikia@univie.ac.at

1 Introduction

1.1 Supervised machine learning

Supervised machine learning makes predictions on a given dataset (test dataset) based on knowledge gained on a previous dataset (also known as training dataset). A Guassian process (GP) is a form of supervised machine learning.

Let us take the example of a dataset $\mathcal{D}$ with n number of observations, where $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., n\}$. Here $\mathbf{x}$ refers to the input vector and y is the output (continuous or discrete). For a given training dataset $\mathcal{D}$, supervised machine learning learns the input-output mapping (a function describing the mapping from $\mathbf{x}$ to y) and applies this function f to make predictions for new inputs $\mathbf{x}_*$ (inputs not seen in the training set).

To find the best mapping function that describes the data, supervised machine learning algorithms usually take two common approaches. The first approach is to consider only a certain class of functions (e.g., linear functions of the input). *This approach has an obvious problem. What is it?* The second approach is to consider all possible functions and assigning a prior probability to all functions, with a higher probability to functions that we consider to be more likely. If one takes the second approach there is an infinite set of possible functions to be computed within a finite time period. This is where Gaussian processes come in handy.

1.2 A short recap

1.2.1 Gaussian distributions

The underlying building block of a Gaussian process is the Gaussian distribution, specifically the multi-variate Gaussian distribution. The multi-variate Gaussian distribution is defined by a mean $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. The mean describes the expected value of the distribution and the variance describes the correlation between the different random variables. The diagonal of the covariance matrix is the variance of each random variable whereas the off-diagonal elements describe the correlation/covariance between the variables. The covariance matrix $\boldsymbol{\Sigma}$ determines the shape of the distribution, as shown in the bivariate example below.

$$\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11}\Sigma_{12} \\ \Sigma_{21}\Sigma_{22} \end{bmatrix}\right) \tag{2}$$

Using NUMPY we can easily draw a bivariate Gaussian distribution by specifying the mean and the covariance matrix, as shown in Figure 1.

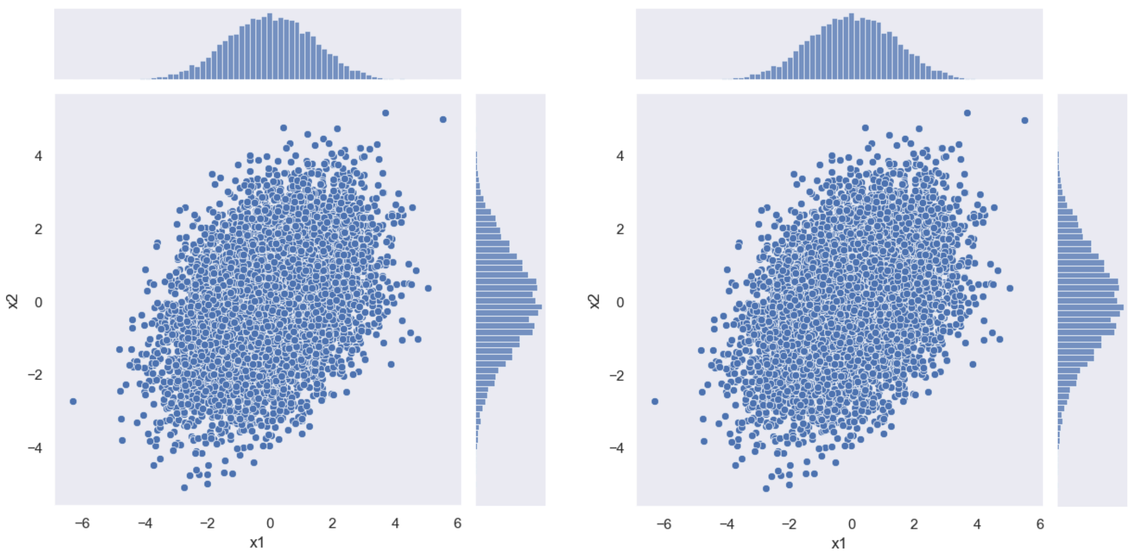

Figure 1: Two examples of bivariate Gaussian distribution.

```python
import numpy as np

rng = np.random.RandomState(0)
X1, X2 = rng.multivariate_normal(mean, cov, n).T
```

Two important operations that will be useful for later are *marginalization* and *conditioning*. In general terms, marginalization means ignoring and conditioning means incorporating information. Figure 1 already demonstrates how marginalization works for a bivariate distribution. While the scatter plot shows the joint distribution of $\mathbf{X}$, the two histograms show the marginal distribution of X_1 and X_2. In other words we can write the marginal distributions as

$$X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \tag{3}$$

$$X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2) \tag{4}$$

This shows that if we marginalize variables out of a Gaussian distribution, the resulting distribution is still Gaussian, known as *closed under marginalization*. In the bivariate case marginalization leads to a univariate Gaussian distribution.

Conditioning means inferring knowledge of a certain variable (lets say X_1) based on new information gained on another variable ($X_2 = x_2$). Gaussian distributions are also *closed under conditiong*. This means that if new information is incorporated in a Gaussian distribution the resulting (conditional) distribution is still Gaussian.

In the bivariate case conditioning can be defined as,

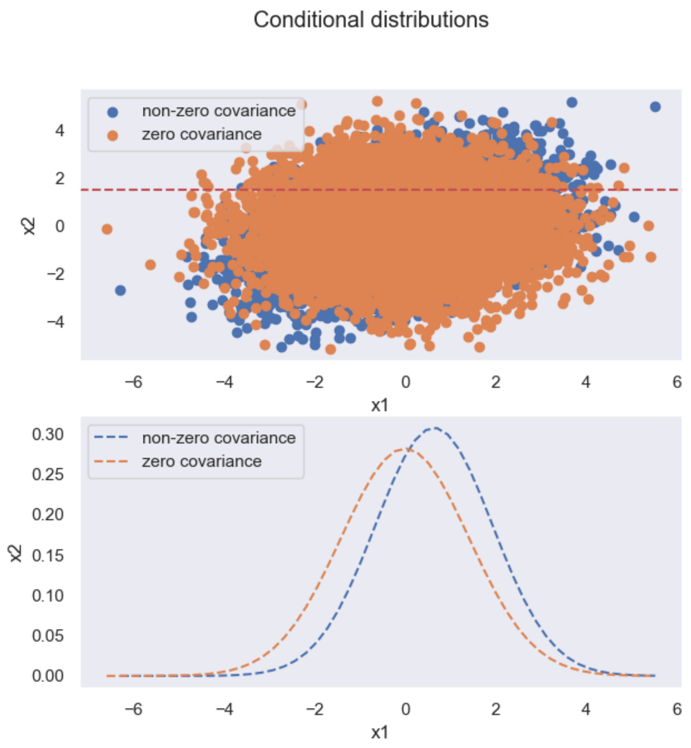

Figure 2: Conditional distributions for the two examples shown in Figure 1. The red dashed line at the top panel marks $X_2 = x_2$

$$X_1|X_2 \sim \mathcal{N}(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(X_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \tag{5}$$

In the above equation X_2 is the conditioned variable. The new mean depends on this conditioned variable where as the covariance matrix does not. The conditional covariance matrix $\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ can be computed by inverting the covariance matrix $\mathbf{\Sigma}^{-1}$, dropping the rows and columns corresponding to the conditioned variable, and inverting it back to get the conditional covariance matrix. Figure 2 shows an example of the conditional distribution of X_1 given X_2 for two separate bivariate distributions.

Until now we have looked into some of the fundamental properties of a Gaussian distribution. What if we have a distribution and we want to draw random samples? Or in other words, how do we convert a uniform random sample to a random sample of a Gaussian distribution (or any distribution)? We can do that using the cumulative distribution and the technique is known as inverse cumulative mapping or inverse transform sampling.

In previous lectures we learnt that the cumulative probability at some value x_1 is the area under the probability density function (PDF) to the left of x_1 (left side area) and

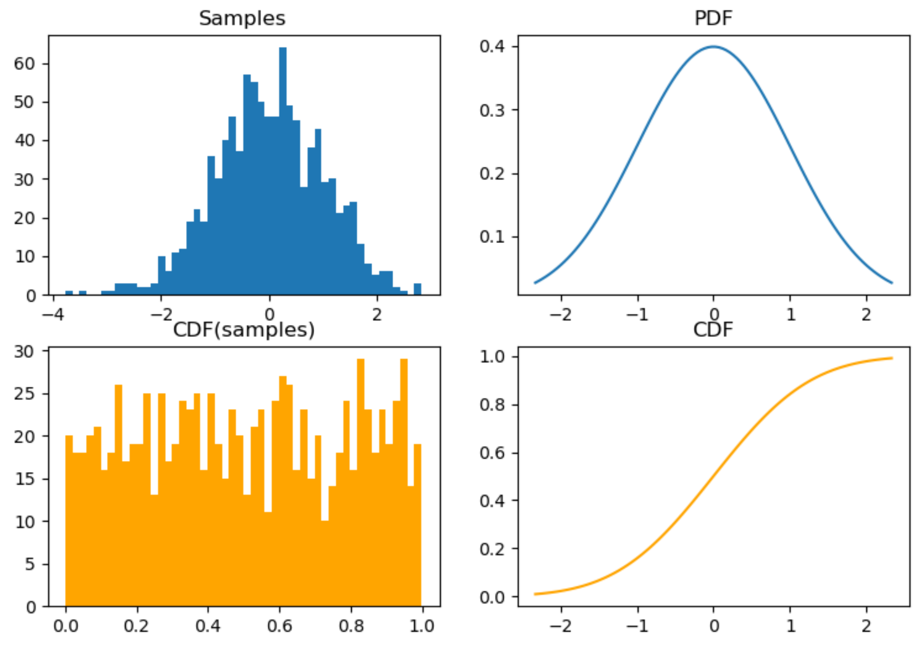

Figure 3: Relationship between a Gaussian PDF and CDF.

the function that returns the cumulative probability of x_1 is called a cumulative probability function (CDF). Figure 3 shows an example of a sample drawn from a standard Gaussian distribution $X \sim \mathcal{N}(0, 1)$ and its associated PDF on the top row, followed by the corresponding CDF and the samples drawn from the CDF at the bottom.

Figure 3 demonstrates an important principle: *For any distribution the cumulative probability is always uniformly distributed.* This means that even if the sample (Top row in Figure 3) has a Gaussian distribution the samples drawn from the corresponding cumulative distribution are uniformly sampled. This principle allows us to transform a sample from a Gaussian (or any) distribution to a uniform sample, or in the reverse direction, we can transform a uniform sample to a sample from any distribution using the inverse function of the distribution's CDF, hence called inverse transform sampling. The sampling in the reverse direction comes very useful when we look into how Gaussian processes work.

1.2.2 Bayesian linear regression

Inference in the Bayesian linear model is based on Bayes' rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}} \tag{6}$$

Unlike the *frequentist* approach where we search for constant values of the parameter $\boldsymbol{\theta}$, with $\boldsymbol{\theta} \in \mathbb{R}$, in the Bayesian approach $\boldsymbol{\theta}$ is drawn from a probability distribution. Equation 6 can be also written as

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})} \tag{7}$$

The *posterior* distribution describes the probability of $\boldsymbol{\theta}$ given a prior distribution, and observed and target data. The *prior* describes our intital knowledge of $\boldsymbol{\theta}$ and can be assumed to be drawn from a Gaussian distribution $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$. The likelihood is the probability of the target given the observed data and parameters. The *marginal likelihood*, also known as the evidence, can be considered as the likelihood averaged over all parameter settings. It is a normalizing constant and is independent of $\boldsymbol{\theta}$, as shown below.

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{8}$$

Figure 4 shows an example dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ with added noise, generated using NUMPY. To apply Bayesian linear regression we assume that the noise is drawn from a Gaussian distribution and $\mathbf{y}$ is also drawn from a Gaussian distribution with a mean μ and a variance σ^2, $\mathbf{y} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\theta}, \sigma^2)$. Additionally, $\boldsymbol{\theta}$ (the slope and the intercept in this case) come from a Gaussian distribution. The variance σ^2 can be any distribution that gives exclusively positive values.

```python
import numpy as np
import pandas as pd

np.random.seed(1)

# Parameters
size = 500
theta_0, theta_1 = 1, 4

# stores x and y as pandas dataframe
x = np.random.random(size)
y = theta_0 + theta_1 * x + np.random.randn(size)
data = pd.DataFrame(dict(x=x, y=y))
```

Using PYMC3 we can obtain the posterior distribution using a few lines of code. The code snippet below assigns prior distributions for the parameters and calculates the likelihood. To get the posterior distribution it uses a NUTS sampler (default in PYMC3). Figure 5

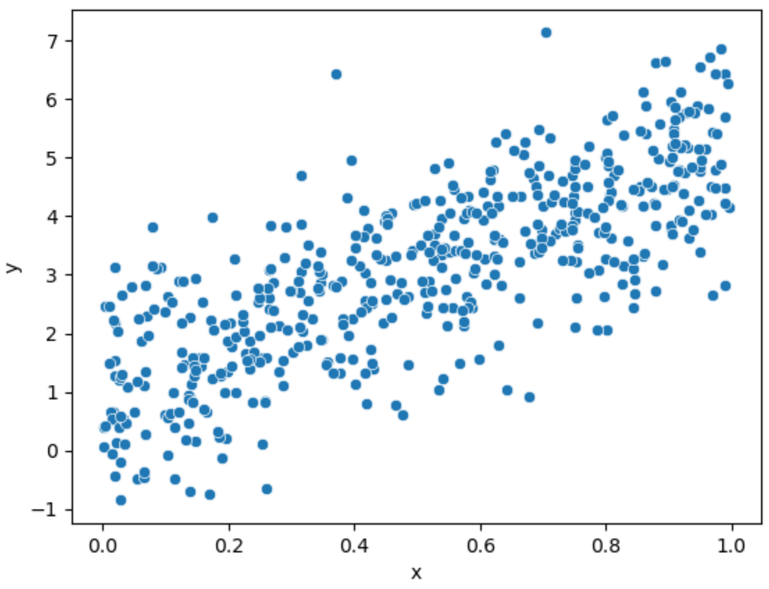

Figure 4: Example dataset used to demonstrate Bayesian linear regression.

shows the posterior distribution of $\boldsymbol{\theta}$ for the dataset in Figure 4. Instead of getting constant values for $\boldsymbol{\theta}$ we now have a distribution of likely parameters. The mean of the slope and the intercept in Figure 5 is very close to the true value of the mean and the intercept. Additionally, the model also tells us how sure it is of the parameters, as shown by the highest density interval of 94% in Figure 5. Thus in Bayesian linear regression we are not limited to one best fitting regression line and can draw many samples from the posterior distributions.

The specific example shown here has a size of 500, which has an impact on the posterior distribution in Figure 5. Using the PYMC3 model estimates of $\boldsymbol{\theta}$ on this dataset we have plotted a fit to the data. In this case the mean of the parameters are used to plot the regression line, but in reality multiple different lines can be drawn from the sample. For comparison the true regression line is also shown in black. *Exercise: If the sample size is low then the model is less sure. Show this using a pymc3 example for a sample size of 50.*

```
#bayesian regression model
with pm.Model() as model:
    # define priors
    slope = pm.Normal('slope', 0, 10)
    intercept = pm.Normal('intercept', 0, 16)
    sigma = pm.Exponential('error', 1)

    # predictions
    y_ = slope*x + intercept
```

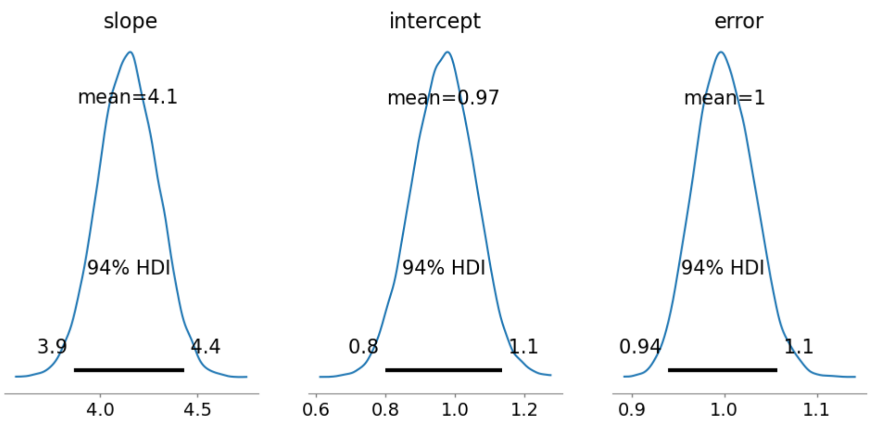

Figure 5: Distributions of the model parameters

```
    likelihood = pm.Normal('observation', y_, sigma=sigma, observed=y)

    # Sampler
    step = pm.NUTS()

    # Posterior distribution
    trace = pm.sample(100000, step,cores=1)

print(pm.summary(trace).round(2))
pm.plot_posterior(trace)
```

Now that we know how Bayesian linear regression estimates model parameters, how does it *predict* the value of $\mathbf{y}$ for new values of $\mathbf{X}$? To achieve that we have to make a small adjustment to the PYMC3 regression model by introducing a placeholder for the data $\mathbf{X}$.

```
x_ = pm.Data('features', x)
y_ = slope*x_ + intercept
```

If we want to predict $\mathbf{y}$ for new $\mathbf{X}$ then we can use the model predictions on the new values. Figure 7 shows the predictions made by our trained model. For easy visualisation the sample size was reduced to 100 instead of the 500 used in the previous figures.

```
#new values of x
x_new = np.linspace(0, 1.0, 10)

# y predictions for new values of x
with model:
    pm.set_data({'features': x_new})
    posterior_predictive = pm.sample_posterior_predictive(trace,var_names=["observation"])
```

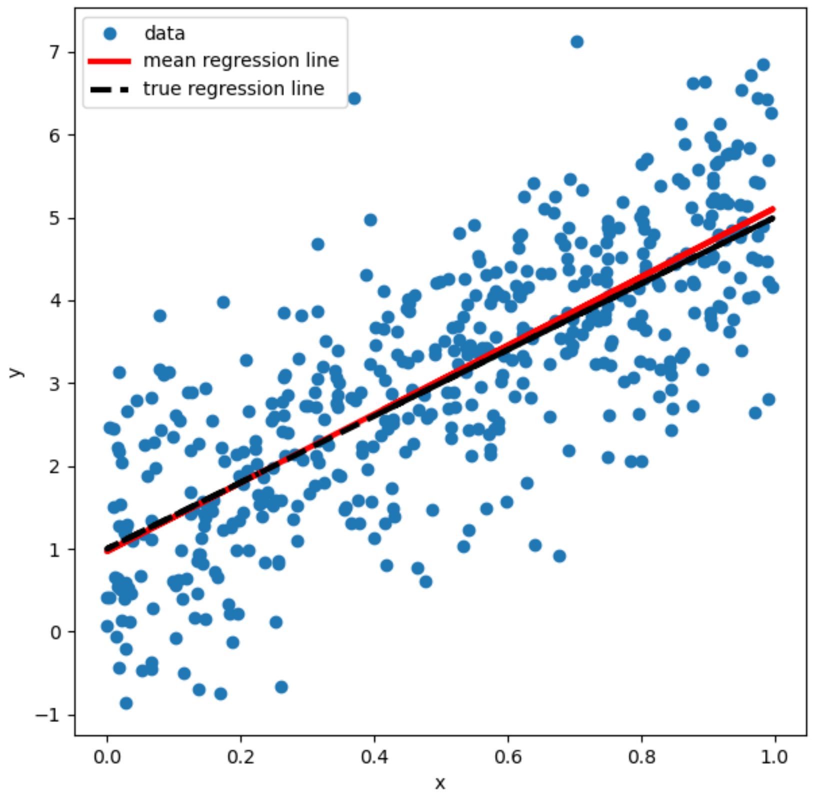

Figure 6: Bayesian regression fit to the dataset in Figure 4. The mean of the posterior distribution was used to plot the regression line, shown in red. The dashed line in black marks the true regression.

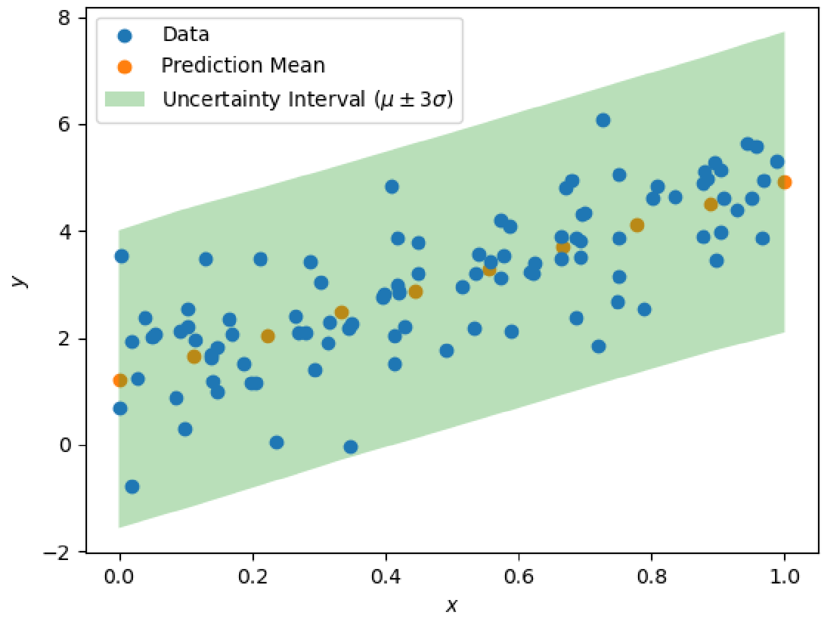

Figure 7: Predictions made on unseen new values of X

```

    y_pred = posterior_predictive['observation']
```

1.3 What is a Gaussian Process?

A Gaussian process (GP) is a stochastic process of random variables with a Gaussian distribution. GPs are non-parametric models that model the underlying function directly, providing a distribution (plus uncertainty) instead of a single value for a prediction. A GP can be completely specified by its mean function and a covariance function, *function-space view*. Another way of conceptualising GPs is the *weight-space* view, i.e., a kernelised Bayesian linear regression, where the kernel parameterization is made by the choice of the covariance/kernel function.

Parametric vs non-parametric model In previous lectures we learnt about linear regressions. As an example, predictions for the a linear function $y = f(x)$ are independent of the observed data but depend on the parameter vector θ, where $\theta \in \mathbb{R}^K$. In a non-parametric model the number of parameters is infinite, ie., θ has an infinite dimension.

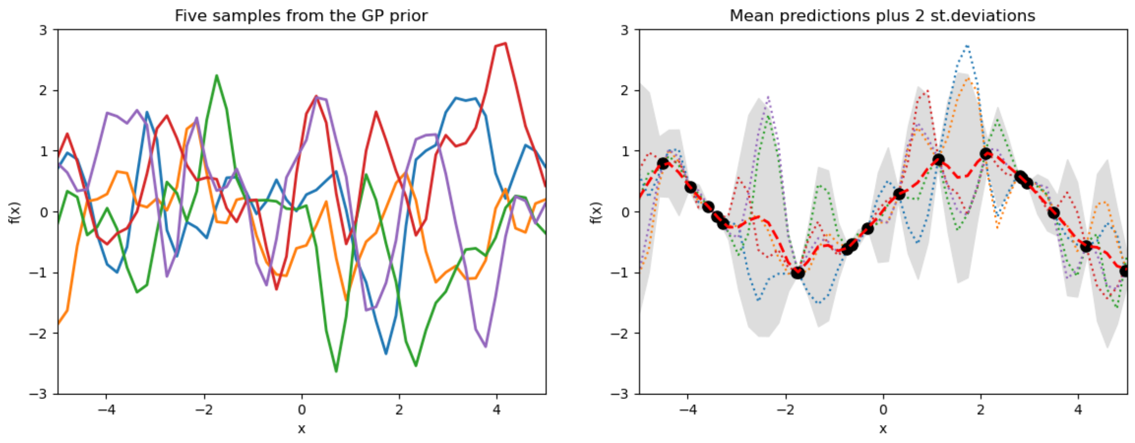

Figure 8: *Left:* Five samples drawn from the prior distribution. *Right:* The mean prediction (red dashed line) after introducing a few data points (black). The samples from the posterior are shown as dotted lines, followed by twice the standard deviation at each input value x (shaded region).

2 Gaussian process regression

Let us take the example of a training data set $\mathcal{D}$ of n observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., n\}$, where $\mathbf{x}$ is an input vector of dimension D and y is the output/target. We can also write $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is the $D \times n$ design matrix and $\mathbf{y}$ is the vector where all outputs are stored. We are interested in obtaining the conditional distibution of the targets/outputs given the inputs.

2.1 The weight-space view

The standard linear regression model with Gaussian noise can be expressed as,

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \quad y = f(\mathbf{x}) + \epsilon \tag{9}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the vector of parameters (represented by $\boldsymbol{\theta}$ in the previous sections) or weights of the linear model [1], f is the function value, y is the target/output value, and ϵ is the noise. It is assumed that the noise follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2.

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{10}$$

[1] A bias term or offset can be also included, which is not explicitly included in our notation

Based on this formulation the probability density of the observations given the parameters and the input values (or the likelihood) is given by,

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x_i}, w) = \mathcal{N}(\mathbf{X}^\top \mathbf{w}, \sigma_n^2 \mathbf{I}) \tag{11}$$

where $\mathbf{I}$ is the identity matrix. Referring back to the Bayes' rule in equation 6 we can assume a zero mean Gaussian prior with a covariance matrix of $\mathbf{\Sigma}_p$ on the weights.

$$\mathbf{w} \sim \mathcal{N}(0, \mathbf{\Sigma}_p) \tag{12}$$

The normalizing constant at the denominator of equation 6 is the same as equation 8 (instead of referring to the weights as $\boldsymbol{\theta}$ we are referring them as $\mathbf{w}$). The posterior distribution is a Gaussian with a covariance matrix $\mathbf{A}^{-1}$,

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \sim \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{y}, \mathbf{A}^{-1}) \tag{13}$$

where $\mathbf{A} = \sigma_n^{-2}\mathbf{X}\mathbf{X}^\top + \mathbf{\Sigma}_p^{-1}$. To obtain predictions we can average over all possible parameter values and weighted by their posterior probability. Thus the predictive distribution f_* at $\mathbf{x}_*$ can be given by

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y}) = \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^\top \mathbf{A}^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^\top \mathbf{A}^{-1}\mathbf{x}_*) \tag{14}$$

The predictivedistribution is Gaussian with the mean given by the posterior mean multiplied by the test input, and the variance being the quadratic form of the test input with the posterior covariance matrix.

We can make a simple enhancement in the example above by projecting the inputs into a high-dimensional *feature space* using a set of basis functions and applying the linear model there. By applying the "kernel trick" in the feature space we can gain significant computational advantages.

If we introduce a function $\boldsymbol{\phi}(\mathbf{x})$ which maps a D-dimensional input vector $\mathbf{x}$ into an N dimensional feature space, and the vector $\mathbf{\Phi}(\mathbf{X})$ is the aggregration of columns $\boldsymbol{\phi}(\mathbf{x})$ for all cases in the training set then the model is

$$f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w} \tag{15}$$

This equation is analogous to the standard linear model discussed above, except that the weight vector has a length N and $\mathbf{X}$ is substituted by $\mathbf{\Phi}(\mathbf{X})$. The predictive distribution can be re-written as,

$$f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y} \sim \mathcal{N}(\frac{1}{\sigma_n^2}\boldsymbol{\phi}(\mathbf{x}_*)^\top \mathbf{A}^{-1}\mathbf{\Phi}(\mathbf{X})\mathbf{y}, \boldsymbol{\phi}(\mathbf{x}_*)^\top \mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{x}_*)) \tag{16}$$

where the covariance matrix is of size $N \times N$ and inverting this might not be convenient if the feature space is large. So we can rewrite the equation in the following way,

$$f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y} \sim \mathcal{N}(\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_p \boldsymbol{\Phi}(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}\mathbf{y}, \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}_* - \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}_*) \quad (17)$$

where $\boldsymbol{\Phi} = \boldsymbol{\Phi}(\mathbf{X})$ and $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$ are used for readability and $\mathbf{K} = \boldsymbol{\Phi}^\top \boldsymbol{\Sigma}_p \boldsymbol{\Phi}$. By using the definitions of $\mathbf{A}$ and $\mathbf{K}$ we can show the equivalence of the mean term in equations 16 and 17. In equation 17 we invert matrices of size $n \times n$, which is much more convenient when $n < N$. Furthermore, the entries of the feature vector takes the form $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$, $\mathbf{x}$ and $\mathbf{x}'$ are either the training or the test datasets. Here $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}(\mathbf{x}')$ is an inner product with respect to $\boldsymbol{\Sigma}_p$ and $k(.,.)$ is a covariance function or a *kernel*. This is very useful as often times it is much easier to compute the kernel than the feature vectors.

2.2 The function-space view

In the function-space view a Gaussian process describes a distribution over functions. It is a collection of random variables, any finite number of which have a Gaussian distribution. The radom variables represent the value of the function $f(\mathbf{x})$ at $\mathbf{x}$. The Gaussian process is specified by its mean function $(m(\mathbf{x}))$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$. The mean function is often considered to be zero for simplicity.

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \quad (18)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \quad (19)$$

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \quad (20)$$

If we go back to our linear example in equation 15 then the mean and the covariance functions can be re-written as

$$\mathbb{E}[f(\mathbf{x})] = \boldsymbol{\phi}(\mathbf{x})^\top \mathbb{E}[\mathbf{w}] = 0 \quad (21)$$

$$\mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] = \boldsymbol{\phi}(\mathbf{x})^\top \mathbb{E}[\mathbf{w}\mathbf{w}^\top]\boldsymbol{\phi}(\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_p \boldsymbol{\phi}(\mathbf{x}') \quad (22)$$

Here $f(\mathbf{x})$ and $f(\mathbf{x}')$ are jointly Gaussian.

An example covariance function is the squared exponential function

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}|\mathbf{x} - \mathbf{x}'|^2) \quad (23)$$

The specification of the covariance function implies distribution over functions. As an example if we take a number of input points $\mathbf{X}_*$ and write out the covariance matrix in equation 23 then the we generate a random Gaussian vector and plot the generated values as a function of the inputs.

$$\mathbf{f}_* \sim \mathcal{N}(0, \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*)) \tag{24}$$

Figure 8 (*Left*) shows five such samples. Our goal here is to not draw random functions from the propr but to incorporate the training data knowledge about the function. As an example for the noise free case the joint distribution of the training $\mathbf{f}$ and test output $\mathbf{f}_*$ can be expressed as,

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) & \mathbf{K}(\mathbf{X}, \mathbf{X}_*) \\ \mathbf{K}(\mathbf{X}_*, \mathbf{X}) & \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right) \tag{25}$$

To get the posterior distribution over functions we need to restrict this joint prior distribution oto contain only those functions that agree with the observed data points. This can be achieved by conditioning the joint prior distribution.

$$\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*)) \tag{26}$$

Figure 8 (Right) shows the results. If we include the Gaussian noise then the predictive distribution has an exact correspondence to the weight-space equations (more details in [1]).

Assuming a zero mean GP prior we can demonstrate a simple application of Gaussian Processes in plain PYTHON. The following code was used to generate Figure 8.

The first step involves generating some input points with added noise.

```
#true unknown function
f = lambda x: np.sin(0.9*x).flatten()

# Sample input points and add noise
N = 10         # number of training points.
s = 0.00005    # noise variance.

X = np.random.uniform(-5, 5, size=(N,1))
y = f(X) + s*np.random.randn(N)
```

Once the data is defined we define the kernel. Here we are using a squared exopnential kernel. Instead of directly inverting the matrix to determine the mean and the variance we use a Cholesky decomposition.

```
def kernel(a, b):
    """ squared exponential kernel """
    kernelParameter = 0.1
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return np.exp(-.5 * (1/kernelParameter) * sqdist)

#Cholesky decomposition is used instead of directly inverting the the matrix
K = kernel(X, X)
L = np.linalg.cholesky(K + s*np.eye(N))
```

Now we are ready to make predictions for some test data.

```
# points we're going to make predictions at.
n=50
Xtest = np.linspace(-5, 5, n).reshape(-1,1)

# mean at our test points.
Lk = np.linalg.solve(L, kernel(X, Xtest))
mu = np.dot(Lk.T, np.linalg.solve(L, y))

# variance at our test points.
K_ = kernel(Xtest, Xtest)
s2 = np.diag(K_) - np.sum(Lk**2, axis=0)
s = np.sqrt(s2)

#draw samples from the prior distribution
L = np.linalg.cholesky(K_ + 1e-6*np.eye(n))
f_prior = np.dot(L, np.random.normal(size=(n,5)))

#draw samples from the posterior distribution
L = np.linalg.cholesky(K_ + 1e-6*np.eye(n) - np.dot(Lk.T, Lk))
f_post = mu.reshape(-1,1) + np.dot(L, np.random.normal(size=(n,5)))
```

References

[1] Carl Edward Rasmussen and Christopher K. I. Williams (2006) *Gaussian Processes for Machine Learning*, MIT Press.