Gaussian process

Flávia C. Motta

November 2021

1 Introduction

In both regression and classification problems in supervised learning, the main goal is to predict the value of a variable associated with a new observation using a function g that relates the attributes of the new entry to the generated prediction. Constructing this function g involves utilizing a training set where the variable of interest and its associated attributes are known. The challenge lies in obtaining the best possible function g that provides accurate predictions for all possible inputs.

Rasmussen (2003) describes two common approaches to tackle this issue. The first approach involves restricting the set of functions to be considered and selecting the function that performs the best based on a predetermined criterion. The second approach assigns a prior probability to each possible function, with higher probabilities associated with functions that are believed to enhance predictive power.

According to Rasmussen (2003), while the first approach faces difficulties in selecting the appropriate classes of functions for evaluation, ensuring good predictions, the second approach deals with an infinite set of possible functions within a finite timeframe. This is where the Gaussian process (GP) comes into play.

A Gaussian process extends the concept of the Gaussian distribution, which describes the distribution of vectors or random variables (in the univariate case), to describe the distribution over functions. It is a powerful non-parametric method that not only provides point estimates of the desired predictive function but also associates a confidence interval with the posterior predictive function.

This work aims to provide a concise overview of Gaussian processes, including their implementation feasibility for different types of problems, their limitations, and finally, present a simulated example to illustrate this method.

2 Definition

A Gaussian process is a stochastic process that is characterized by the property that any finite set of random variables belonging to the process, defined over a continuous domain, follows a multivariate Gaussian distribution. This means that any linear combination of these variables will also be normally distributed.

In essence, a Gaussian process is an extension of the Gaussian distribution. While the Gaussian distribution is determined by its mean and covariance, which are a vector and a matrix respectively, a Gaussian process is fully specified by its mean function $\mu(x)$ and covariance function $k(x, x')$.

It is important to note that the covariance function k is a positive definite kernel function, which measures the similarity between input arguments x. In other words, it provides a reference for how close or related two points are. While a multivariate Gaussian distribution is defined over vectors, a Gaussian process is defined over functions (Bousquet *et al.*, 2011), hence the notation is given by

$$f(x) \sim GP(\mu(x), k(x, x')). \tag{1}$$

The structure of a Gaussian process can be utilized to make inferences about the prediction functions of a test set. This is achieved by defining a prior distribution for the vector $\mathbf{f}^*$, which represents the collection of function values at the test set points. Although $\mathbf{f}^*$ is independent of the training set, it characterizes properties of the functions, such as smoothness (Bousquet *et al.*, 2011). Hence, we have

$$\mathbf{f}^* \sim N(0, K^{**}), \tag{2}$$

where K^{**} is the covariance matrix that describes the pairwise similarity between all points in the test set.

Once a Gaussian process is specified as a prior distribution, it can be updated based on the training data to compute a posterior distribution, which is then used for making predictions on new observations. In this context, let $\mathbf{f}$ represent the known function values for the training data, and $\mathbf{f}^*$ denote the set of function values corresponding to the inputs of the test set (Bousquet *et al.*, 2011). The joint distribution between $\mathbf{f}$ and $\mathbf{f}^*$, assuming no noise in the process, is given by

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K & K^* \\ K^{*'} & K^{**} \end{bmatrix} \right), \tag{3}$$

where K measures the similarity between all pairs of points in the training set, K^* represents the similarity between all points in the training set with all points in the test set, and K^{**} captures the similarity between all pairs of points in the test set (Rasmussen, 2003).

To obtain the posterior distribution over functions, it is necessary to condition the joint prior distribution on the observed data points, thereby restricting it to functions that are consistent with the data (Rasmussen, 2003). This can be achieved by conditioning the prior distribution on the observed data set, resulting in the following expression

$$\mathbf{f}^* | X_*, X, \mathbf{f} \sim N(K^{*'} K^{-1} \mathbf{f}, K^{**} - K^{*'} K^{-1} K^*). \tag{4}$$

This represents the posterior distribution for a specific set of test data. It is important to note that the posterior variance is obtained by subtracting a positive term, dependent on the inputs of the training set, from the prior variance function. Thus, the posterior variance is always smaller than the prior variance, as the training set provides additional information that reduces uncertainty about the distribution of interest (Bousquet *et al.*, 2011).

However, in real-world applications, it is common to have noisy versions of the true function values, represented as $y = f(x) + \varepsilon$. Assuming that the noise follows a Gaussian distribution, is independent, and identically distributed, i.e., $\varepsilon \sim N(0, \sigma_n^2)$, each $f(x)$ is associated with an additional covariance term (Bousquet *et al.*, 2011). Therefore, the joint distribution between y and $\mathbf{f}^*$, accounting for noise in the process, can be expressed as

$$\begin{bmatrix} y \\ \mathbf{f}^* \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & K^* \\ K^{*'} & K^{**} \end{bmatrix} \right). \tag{5}$$

In this context, the predictive distribution of the Gaussian process is expressed as follows

$$\mathbf{f}^* | X, y, X_* \sim N(K^{*'}[K + \sigma_n^2 I]^{-1} y, K^{**} - K^{*'}[K + \sigma_n^2 I]^{-1} K^*). \tag{6}$$

3 Advantages and limitations

The Gaussian process offers several advantages over other methods, including the ability to provide both point predictions and confidence intervals for the posterior predictive function. This feature is particularly valuable in domains where uncertainty in predictions is of interest, such as in medical applications for prognosis and diagnosis.

Another advantage of Gaussian processes is their flexibility in incorporating domain-specific knowledge through the choice or customization of the kernel function. However, this advantage can also be a limitation, as the performance of Gaussian processes heavily relies on selecting an appropriate kernel. Since there is no objective criterion for determining the optimal kernel, the choice becomes crucial for achieving good generalization properties.

A limitation of Gaussian processes is their computational cost, as they need to consider all training data during prediction, which becomes inefficient in high-dimensional spaces. The computational burden is further exacerbated by the matrix inversion involved in the method. Additionally, Gaussian processes inherently assume a Gaussian prior, which may not always be realistic for certain problem domains, limiting the applicability of this technique.

4 Simulation

To start the implementation, import the necessary libraries such as NUMPY for calculations and MATPLOTLIB for visualizations.

```
import numpy as np
import matplotlib.pylab as plt
np.random.seed(0)
```

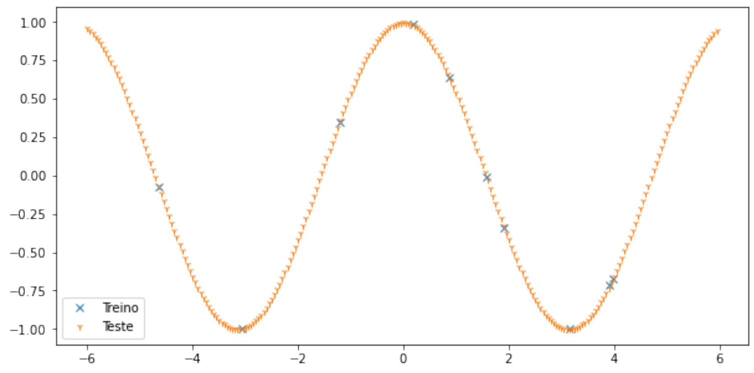

Figure 1: Simulated data.

Simulate cosine function data (using the *generate_data* function) to train the model.

```
def generate_data(n=10, noise_variance=1e-6):
    np.random.seed(20)
    X = np.random.uniform(-5., 5., (n, 1))
    X.sort(axis=0)
    y = np.cos(X) + np.random.randn(n, 1) * noise_variance**0.5
    return X, y

# simulating data sets
X_train, y_train = generate_data(10)
```

The training data was simulated within the range of -5 and 5, while the test data will be generated as a grid of values between -6 and 6. Figure 1 shows the simulated data.

```
X_test = np.arange(-6, 6, 0.05)
y_test = np.cos(X_test)
```

```
plt.plot(X_train, y_train, "x", label = "Treino")
plt.plot(X_test, y_test, "1", labe l = 'Teste')
plt.legend()
plt.show()
```

```
noise_var = 1e-6
N, n = len(X_train), len(X_test)
```

For the kernel function, we will use the Radial Basis Function (RBF) kernel with the formula

$$K(x, x') = \exp^{-\frac{||x-x'||^2}{2l^2}},$$

where l is a free parameter of the model. The function becomes smoother as l increases.

```
def kernel(x, y, l2 = 0.1):
    sqdist = np.sum(x**2,1).reshape(-1,1) + \
             np.sum(y**2,1) - 2*np.dot(x, y.T)
    return np.exp(-.5 * (1/l2) * sqdist)
```

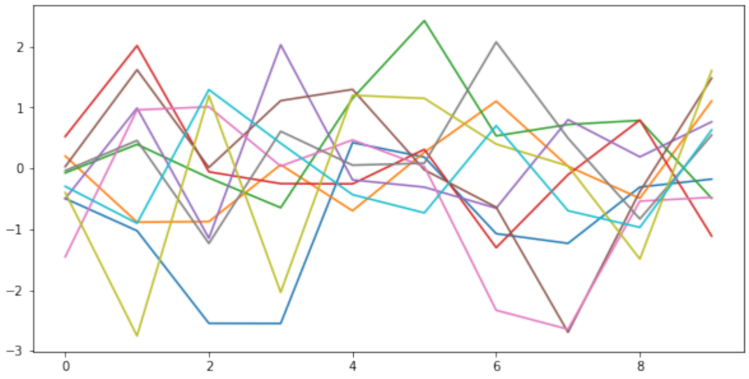

Figure 2: Simulated data of $\mu(x)$.

Figure 2 displays 10 average functions $\mu(x)$ generated from a Gaussian normal distribution $N(0, 1)$, using the following code:

```
n_samples = 10
mx = np.random.normal(loc=0.0, scale=1.0, size=(N, n_samples))
```

```
plt.plot(mx)
```

By calculating the similarity matrix K^{**} with $l^2 = 0.01$, we can obtain the prior f^* using Equation (2). Figure 3 presents the calculated priors along with the training points.

```
# calculating similarity matrix
K = kernel(X_train, X_train, l2 = 0.01)
L = np.linalg.cholesky(K + noise_var*np.eye(N)
# calculating prior
f_prior = np.dot(L, mx)
```

```
# Figure 3
plt.scatter(X_train[:,0], y_train[:,0], s = 100, color = 'red');
plt.plot(X_train[:,0], f_prior, alpha = .6);
```

To compute the posterior function for a specific set of test data, we can use Equation (6) and the following code snippet. Figure 4 displays the calculated posteriors along with the training points.

```
# similarity matrices
K = kernel(X_train, X_train, l2  = .1)
K_x = kernel(X_train, X_test.reshape(-1, 1), l2  = .1)
K_xx = kernel(X_test.reshape(-1, 1), X_test.reshape(-1, 1), l2  = .1)
# calculating posterior
L = np.linalg.cholesky(K + noise_var*np.eye(N))
Lk = np.linalg.solve(L, K_x)
mu = np.dot(Lk.T, np.linalg.solve(L, y_train))
L = np.linalg.cholesky(K_xx + noise_var*np.eye(n) - np.dot(Lk.T, Lk))
f_post = mu.reshape(-1,1) + np.dot(L, np.random.normal(size=(n,n_samples)
    ))
```

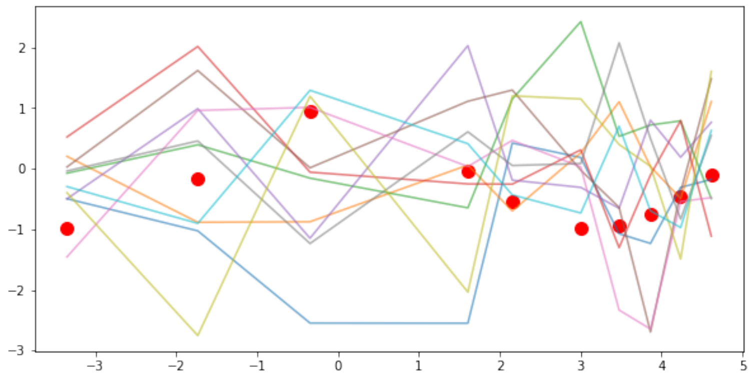
Figure 3: *Prior f^**

.

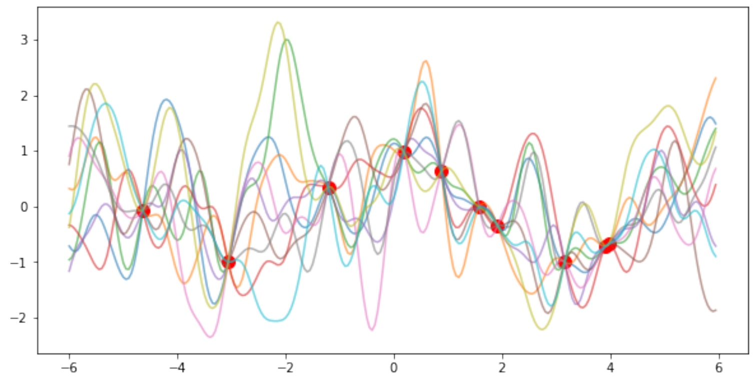
Figure 4: *Posterior* function and training points.

```
# Figure 4
plt.scatter(X_train[:,0], y_train[:,0], s = 100, color = 'red');
plt.plot(X_test[:,0], f_post, alpha = .6);
```

We can observe that the uncertainty is nearly zero for the points present in the training data set, and the variance increases as the values move away from the training points.

Finally, the Gaussian process model will be used to make predictions in a regression problem. Figure 5 shows the predictions for the test set. It can be observed that the predicted line closely aligns with the actual line, and the uncertainty is higher for points further away from the training set. Additionally, it is worth noting that the predicted line outside the training data limits may have a poor fit due to limitations of the method.

```
# Function to make predictions
def posterior(X, Xtest, y, l2=0.1, noise_var=1e-6):
    # compute the mean at our test points.
    N, n = len(X), len(Xtest)
    K = kernel(X, X, l2)
    L = np.linalg.cholesky(K + noise_var*np.eye(N))
    Lk = np.linalg.solve(L, kernel(X, Xtest, l2))
```

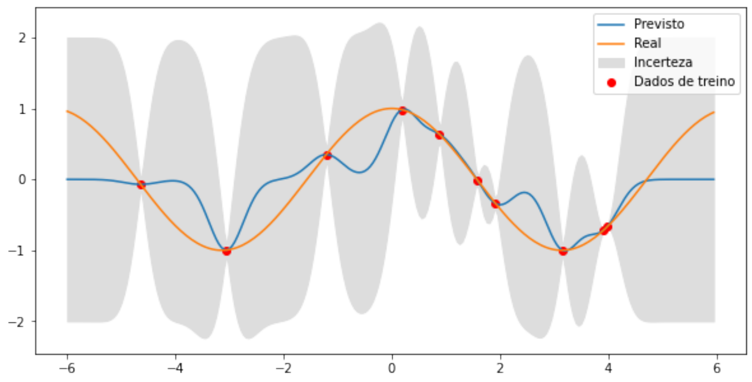

Figure 5: Prediction in a regression problem using Gaussian process.

```
    mu = np.dot(Lk.T, np.linalg.solve(L, y))
    # compute the variance at our test points.
    K_ = kernel(Xtest, Xtest, l2)
    sd = np.sqrt(np.diag(K_) - np.sum(Lk**2, axis=0))
    return (mu, sd)
```

```
# Predicting test points
pred_ = posterior(X_train, X_test.reshape(-1, 1), y = y_train, l2=0.1,
    noise_var=1e-6)
mu = pred_[0][:, 0]
stdv = pred_[1]
```

```
# Plotting the results
plt.plot(X_test.reshape(-1, 1)[:, 0], pred_[0][:, 0])
plt.plot(X_test, y_test)
plt.gca().fill_between(X_test, mu-2*stdv, mu+2*stdv, color="#dddddd")
plt.scatter(X_train, y_train, color = 'red')
plt.show()
```

Figure 6 demonstrates the effect of different values of l^2 on the fit using the same training and test sets. For larger values of l^2, the function becomes smoother. When $l^2 = 0.01$, points are considered similar only if they are very close, resulting in high sensitivity to the training points. However, for $l^2 = 1$, the best fit is achieved, with the predicted line closely matching the actual line and lower uncertainties. This highlights the importance of using hyperparameter tuning techniques to select the optimal value of l^2.

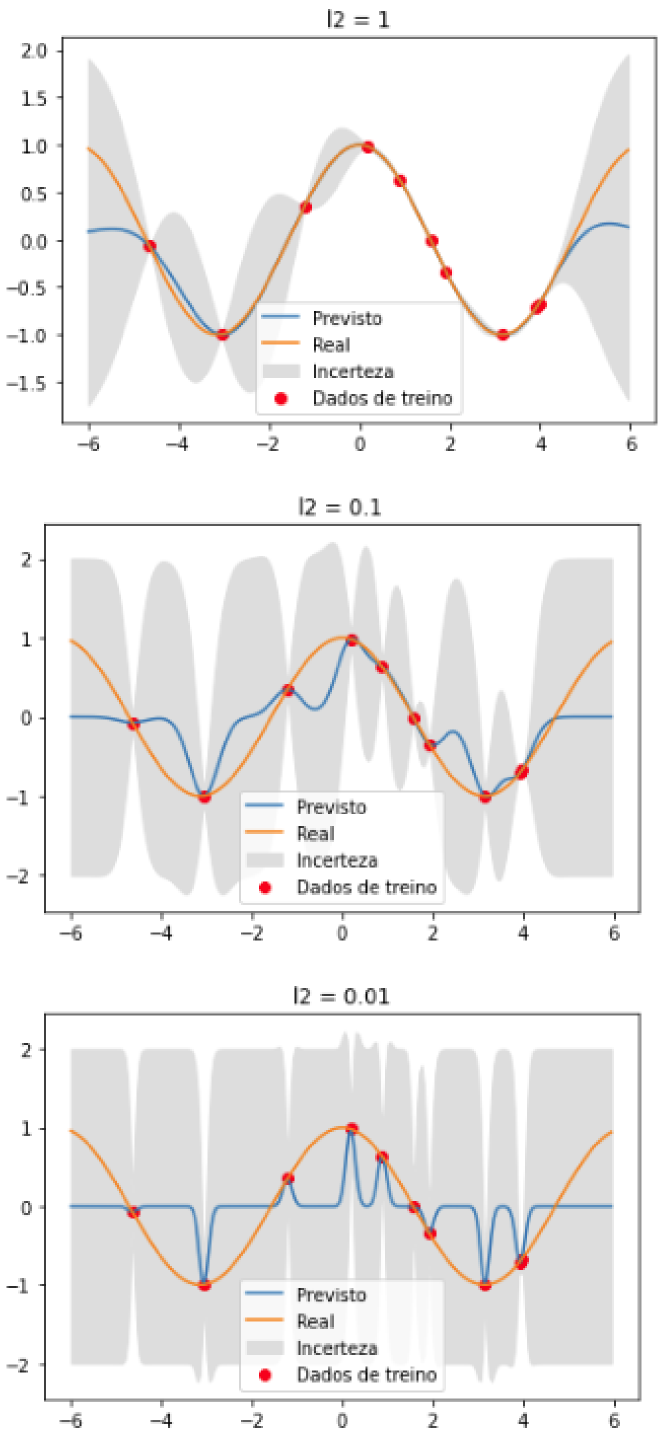

Figure 6: Fit effects due to different values of l^2.

5 Conclusions

The analysis has demonstrated that Gaussian process regression can be a highly effective method when appropriate choices of Kernel and hyperparameters are made. The method offers several advantages, performing well on small datasets and providing meaningful measures of uncertainty in predictions. Additionally, with minor adjustments, it can be extended to tackle classification problems. These qualities make Gaussian processes a promising tool for modeling and prediction in various fields. However, it is important to emphasize the importance of careful parameter selection to achieve optimal results.

References

Bousquet, O., von Luxburg, U. e Rätsch, G. (2011). *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176. Springer.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. Em *Summer school on machine learning*, páginas 63–71. Springer.