Notes on Gaussian Process Regression

(with Python examples)

Kamila Zdybał

Université libre de Bruxelles, kamila.zdybal@ulb.be
camillejr.github.io/science-docs, kamila.zdybal@gmail.com

Preface

This document is still in preparation. Please feel free to contact me with any suggestions, corrections or comments.

Keywords

Gausssian Process Regression (GPR), regression, covariance

Contents

1 Introduction

Gaussian Process Regression (GPR) is a regression technique which does not restrict what functions are used to model the data - unlike linear regression for instance. Instead, we allow many possible functions from a certain pre-specified "pool". To each of these function we attribute a certain probability of how likely we believe it will fit well our data and update this functions pool as new data points are observed.

1.1 Distribution over functions

In GPR we do not assume a specific class of function for regression. Instead, we allow many types of functions but to each of them we attribute a certain likelihood that it can well regress our data. The space of these functions is mapped on a space of likelihoods and such mapping is called a *distribution over functions*. Visually, one may treat this as a probability distribution drawn "above" the space of all possible functions - some functions are more likely, some less.

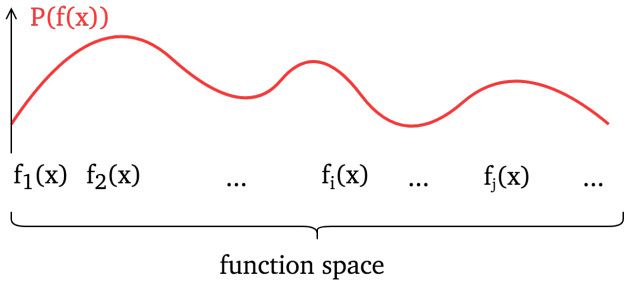

Figure 1: Conceptual visualisation of distribution over functions.

2 Probability

2.1 Prior and posterior

Two words that will be often used in GP are prior and posterior. The prior and posterior probabilities in GP are often linked to functions and it is important to understand their notion.
The ***prior probability*** is the probability we assign to a specific object (event) before we collect further evidence or make observations that can change it. It can be viewed as an "initial guess" or "initial belief" about the given system. The ***posterior probability*** is the updated prior probability that takes into account the new observations or evidence. Conceptually, we may write the following:

$$\text{prior distribution} + \text{data} = \text{posterior distribution} \tag{1}$$

As explained in section 1.1, in GP we encounter probabilities associated to functions. The prior will be the functions that we believe will fit our data well before we actually attempt fitting these functions to new observation points. After the observations have been made, the functions get the "update" to better fit the newly arrived points and these functions that are left will become our posterior.

2.2 Likelihood and marginal likelihood

3 Covariance matrix

Let's think about a dot product between two vectors:

$$\text{dot}(x_i, x_j) = |x_i||x_j|\cos(\phi) \tag{2}$$

It describes the amount of projection of vector x_i onto x_j (or vice versa) and can be useful when we need to know how much one vector points in the direction of the other.
Now imagine that you have a data set $\mathbf{X}$ with n vectors (features), and you would like to know what is the dot product of every possible pair drawn from these vectors. In other words, you would like to know how correlated are all vectors with each other. You can achieve this "global" dot product by multiplying:

$$\mathbf{S} = \mathbf{X}^T\mathbf{X} \tag{3}$$

the result $\mathbf{S}$ is called the *covariance matrix*. Notice that every entry (i, j) in this matrix is a dot product $\text{dot}(x_i, x_j)$ and it is:

$$\text{dot}(x_i, x_j) = \text{cov}(x_i, x_j) \text{ for } i \neq j \tag{4}$$

$$\text{dot}(x_i, x_j) = \text{var}(x_i, x_j) \text{ for } i = j \tag{5}$$

The covariance matrix is symmetric due to symmetry: $\text{dot}(x_i, x_j) = \text{dot}(x_j, x_i)$.

4 Covariance kernels

The *covariance kernel* is essentialy a function that populates the covariance matrix. This makes our life easier, since first, this matrix might be huge and second, we can easily implement the underlying structure to the covariance.
The covariance kernel has to be designed such that there is symmetry: $K(x_i, x_j) = K(x_j, x_i)$.[1]

4.1 Examples

Squared exponential kernel:

$$K(x_i, x_j) = h^2 \exp\left(\frac{-(x_i - x_j)^2}{\lambda^2}\right) \tag{6}$$

4.2 The meaning of hyperparameters

We will now take a look at the Squared Exponential kernel which was introduced before and explore how the hyperparameters affect the structure of the covariance matrix and the type of priors drawn from the distribution.
`y = np.random.multivariate_normal(mean, cov)`
Even the variables which are further away are correlated, which is also reflected in the type of prior that is drawn from this distrbution. For $\lambda = 5$ it looks much less random than for $\lambda = 0.1$ - the variable's value at any point is strongly affected by the values of the nearby variables. In other words, the derivatives at any point change much less drastically at $\lambda = 5$ than at $\lambda = 0.1$.

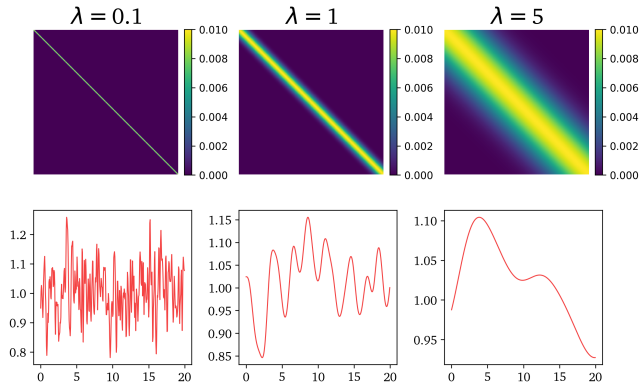

Figure 2: Adjusting the kernel width.

[1] In [3], it has been said that the covariance can vary in different directions which made me wonder...

Varying the scaling factor h, the priors are stationary and the covariance has the same width. What is changing however is the numerical value of the covariance matrix. For every h the covariance matrix gets scaled and the largest elements in the matrix are always h^2.

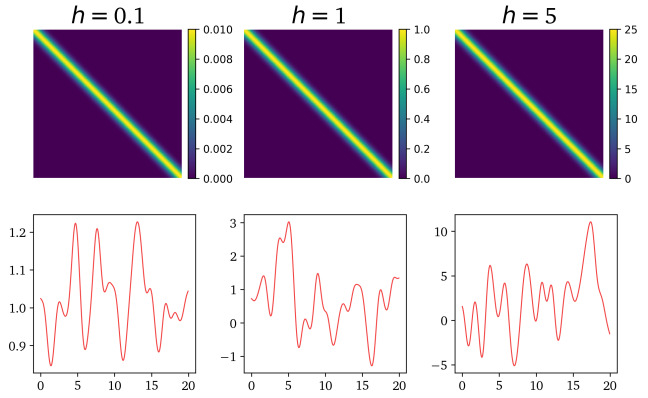

Figure 3: Adjusting the kernel scaling factor.

4.3 What is Gaussian in Gaussian Process?

The mean value of the distribution is reflected in the y-axis values in Fig. 1 and 2. It is yet more evident to look at the histograms from more than one prior realisation. In Fig. 3 20 realisations have been drawn and it already becomes visible that the y-value distribution starts to have the bell shape with the prescribed mean.

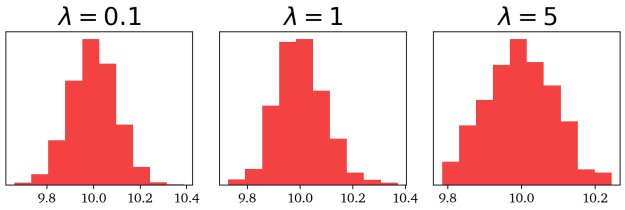

Figure 4: Histogram of 20 realisations with mean = 10.

In fact, one restriction is placed on the possible regression functions inside the "" pool": all functions have a Gaussian marginal distribution. (Though it does not need to be a case either - GPML uses other likelihoods - but we still call the technique **Gaussian** Process Regression). In other words, the histogram of realisations' values begin to look like a normal distribution as the number of realisations increase.

5 Building your GPR in Python

References

[1] C. E. Rasmussen, C. Williams, *Gaussian Process for Machine Learning*, 2006

[2] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, S. Aigrain *Gaussian Processes for Timeseries Modelling*, 2012

[3] A. Scaife, *Machine Learning: Gaussian Process Modelling in Python*, an online lecture