

Bayesian optimization in machine learning

José Jiménez Luna

January 9, 2017

Contents

1	Organization of this work	5
1.1	Introduction	5
1.2	How this manual is supposed to be read	6
2	Gaussian Process regression	7
2.1	A weight space view for Gaussian Processes	7
2.1.1	Standard Bayesian linear regression	7
2.1.2	Kernel functions in feature space	8
2.2	A function space view for Gaussian Processes	8
2.3	Prediction using a Gaussian Process prior	10
3	Bayesian optimization	11
4	Experiments	13
5	pyGPGO: A simple Python Package for Bayesian Optimization	15

Chapter 1

Organization of this work

1.1 Introduction

This master's thesis objective is to provide an easy to follow manual to users who want to use some form of Bayesian Optimization in practice. While the theory of Bayesian Optimization itself is pretty new, the foundations on which it is implemented in practice have been formally presented since the 90s. This work itself does not develop any new theory, but aims to provide users with both a theoretical and a practical introduction to Bayesian Optimization, as well as a Python implementation that users can use in their research.

Readers will find code during theoretical explanations, as I believe the easiest way to learn is to do. This will hopefully be beneficial for the reader. All the code implemented is available with a MIT license in a GitHub repository (<https://github.com/hawk31/pyGPGO>). This is a Python (>3.5) package that provides most of the functionality presented in this manual.

Bayesian Optimization focuses on the global optimization of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ over a compact set A . The problem can be formalized as:

$$\max_{\mathbf{x} \in A} f(\mathbf{x}) \quad (1.1)$$

Most optimization procedures (local based ones such as gradient ascent, for example) assume that the function f is closed-form, that is, can be written in a paper, that it is convex, with known first or second order derivatives or cheap to evaluate. Bayesian optimization focuses on all these problems proposing a very elegant solution. By the use of a surrogate model, a Gaussian Process, a Bayesian optimization procedure can help to find the global minimum of a non-necessarily convex, expensive functions. These methods shine also where there is no closed-form expression to evaluate and does not need any function derivatives.

Now is when the machine learning part of the title comes into play. In machine learning, we are usually interested in minimizing (or maximizing the opposite) a loss function L . These losses can take many forms, for example, when doing regression, a typical loss might be the mean squared error between predictions and observed values on a holdout test set.

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_i (y_i - \hat{y}_i)^2 \quad (1.2)$$

In binary classification, for example, a very popular loss function example is the logarithmic loss:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{n} \sum_i (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (1.3)$$

Focus in any case, that these losses are typically defined in a subset of \mathbb{R} . We focus on the supervised setting of machine learning, and more specifically regression. Depending on the problem at hand, even evaluating these losses can be very expensive from a computational point of view. This may have to do with the machine learning algorithm used or the size of the dataset at hand. These machine learning algorithms typically have *hyperparameters*, that is, parameters that have to be tuned in a sensible way to get the best performance possible out of these models. In the machine learning community it is common for practitioners to do an hyperparameter grid search or even randomize it. Since the training of a single model can already take substantial resources in terms of CPU cycles or memory, we would like to have a more efficient and cheap way to optimize these hyperparameters. Bayesian optimization will let us do that by proposing the next candidate point \mathbf{x} to evaluate according to several criteria.

1.2 How this manual is supposed to be read

Before we dive directly into the topic at hand, it is mandatory to explain how this manual is intended to be read. Different chapters will cover different material, so if the reader is familiar with a topic in a chapter, for example, the chapter on Gaussian Processes, he can skip directly to the chapter on Bayesian Optimization or to the implementation if he wishes. There is no complicated material in any of the chapters, as this manual does not aim to be research material on the the latest advances in the already quite crowded field.

Chapter 2 focuses on a swift but thorough introduction to regression problems using Gaussian Processes. These are the surrogate models we will use for Bayesian Optimization in Chapter 3. We will cover the theory behind them both from a weight space view and from a functional view. We will also explain different covariance functions and their role in these models, as well as methods for optimizing their hyperparameters. Finally we will provide the reader with usable code to fit a Gaussian Process in a regression problem. This chapter is heavily based on Carl E. Rasmussen excellent book *Gaussian Processes for Machine Learning*[ref]. In fact, if the reader is interested in a more advanced, wider, slowly-paced introduction to Gaussian Processes, this is the resource to go to.

Chapter 3 is about the main topic in this work, Bayesian Optimization. Once we have laid down all the foundations of Gaussian Processes, we can start explaining the theory of Bayesian optimization using these as surrogate models. The algorithms, while simple, are very powerful. The role of several acquisition functions, that is to say, the functions that will propose the next point to evaluate will be discussed, as well as their advantages or disadvantages. The references on this chapter will be more diverse than on the previous chapter, as I will try to summarize several publications.

Chapter 4 covers experiments using the software provided alongside this manual. These are mostly mid-sized regression or classification problems where we will compare the performance of Bayesian Optimization of hyperparameters with several regressors/classifiers with other strategies, such as random search or simulated annealing. Most of these datasets are related to experimental sciences, and some of them are typically used for other benchmarking purposes in other studies. Readers already familiar with Gaussian Processes should jump directly into this chapter.

Chapter 5 is the shortest one. It will cover technical explanations of pyGPGO, the software developed alongside this manual. I will also try to provide practical examples on how to use the software.

Chapter 2

Gaussian Process regression

We will focus on regression problems in this chapter. Assume we have some labelled data $D = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, n\}$, where x is a vector of covariates and y denotes a continuous objective variable. We wish to learn a predictive distribution over new values of x , so that we can make predictions and inference over these. In practice, for simplicity we write that $D = (X, \mathbf{y})$, where X is a covariate matrix.

One can interpret a Gaussian Process in several ways, the most used one is the function space view, which is the one we will cover second here and the one we will assume for the rest for the manual. In this view, we consider a Gaussian Process to be a distribution over functions, instead of over values. Inference takes place directly in this space. For completeness, we will also provide a weight-space view first, that might be more appealing to readers familiar with Bayesian linear regression.

2.1 A weight space view for Gaussian Processes

In this section we will try to draw connections between Bayesian linear regression and our introduction to Gaussian Processes, by the use of kernel functions.

2.1.1 Standard Bayesian linear regression

A Bayesian linear regression model with Gaussian error can be formulated as:

$$y = f(\mathbf{x}) + \epsilon \quad (2.1)$$

where we typically assume $\epsilon \sim N(0, \sigma_n^2)$. This noise assumption directly implies a Gaussian likelihood, thus it can be easily proven that:

$$p(\mathbf{y} \mid X, \mathbf{w}) \sim N(X^T \mathbf{w}, \sigma_n^2 I) \quad (2.2)$$

Assume now a Gaussian prior on the weights \mathbf{w} :

$$\mathbf{w} \sim N(\mathbf{0}, \Sigma_p) \quad (2.3)$$

We are interested now on the posterior distribution of w , given both X and y , and assuming the model in Equation 2.1, that is:

$$p(\mathbf{w} \mid \mathbf{y}, X) = \frac{p(\mathbf{y} \mid X, \mathbf{w}) p(\mathbf{w})}{p(\mathbf{y} \mid X)} \quad (2.4)$$

One can solve this problem by means of sampling procedures like Markov Chain Monte Carlo, but in this particular case, there is a closed-form solution. It can be proven quite easily that:

$$p(\mathbf{w} \mid X, \mathbf{y}) \sim N\left(\frac{1}{\sigma_n^2} A^{-1} X^T \mathbf{y}, A^{-1}\right) \quad (2.5)$$

where $A = \sigma^{-2} X X^T + \Sigma_p^{-1}$. Notice that a very easy MAP (maximum a posteriori) estimate of the weights can be obtained by just computing the mean of this distribution. Now, to make predictions for a particular test case \mathbf{x}_* , we average over all possible parameter values, hence we get a whole predictive distribution. Again, it can be proven that:

$$f_*|x_*, X, \mathbf{y} \sim N\left(\frac{1}{\sigma_n^2} \mathbf{x}_*^T A^{-1} X \mathbf{y}, \mathbf{x}_*^T A^{-1} \mathbf{x}_*\right) \quad (2.6)$$

2.1.2 Kernel functions in feature space

We have presented a very simple Bayesian approach to linear regression in the previous section. While useful, it lacks expressiveness. A very simple idea is to project this data into a higher dimension, where it may be more easily separated by a linear model of this sort. This is called using the kernel trick. We can do this by the use of a covariance (or kernel) function $\phi(\mathbf{x})$. Note by $\Phi(X)$ the aggregation of columns after computing this kernel function in the entire dataset at hand.

The model becomes now:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} \quad (2.7)$$

All the math presented in the previous section applies here, just placing $\phi(\mathbf{x})$ instead of \mathbf{x} . The predictive distribution over y becomes now, for example:

$$f_*|x_*, X, \mathbf{y} \sim N\left(\frac{1}{\sigma_n^2} \phi(\mathbf{x}_*)^T A^{-1} \Phi \mathbf{y}, \phi(\mathbf{x}_*)^T A^{-1} \phi(\mathbf{x}_*)\right) \quad (2.8)$$

where for simplicity we have written $\Phi = \Phi(X)$ and $A = \sigma_n^{-2} \Phi \Phi^T + \Sigma_p^{-1}$. The predictive distribution needs to invert $N \times N$ matrix. Equation 2.8 can be rewritten as:

$$f_*|x_*, X, \mathbf{y} \sim N\left(\phi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \mathbf{y}, \phi_*^T \Sigma_p \phi_* - \phi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \Phi^T \Sigma_p \phi_*\right) \quad (2.9)$$

where we have again simplified notation by $\phi_* = \phi(\mathbf{x}_*)$ and $K = \Phi^T \Sigma_p \Phi$. Now notice that the entries of K for both train and test set are of the form $\phi(\mathbf{x}_*^T) \Sigma_p \phi(\mathbf{x}_*)$. We have implicitly defined now a *covariance function* of the form $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}_*^T) \Sigma_p \phi(\mathbf{x}_*)$. This is in fact an inner product with respect to Σ_p . That is if we define $\psi(\mathbf{x}) = \Sigma_p^{1/2}(\mathbf{x})$, then a simple dot product representation of a covariance function is:

$$k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x})^T \psi(\mathbf{x}') \quad (2.10)$$

where $\Sigma_p^{1/2}$ can be defined by means of a singular value decomposition. Typically, we will replace the original feature vectors by these dot products, *lifting* to a higher space. This will make more sense in the following section.

2.2 A function space view for Gaussian Processes

We formally define a Gaussian Process as a collection of random variables, any finite number of which have a joint Gaussian distribution. This process is totally defined by two functions. The first one is its *mean function*:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \quad (2.11)$$

The second one is its *covariance function*:

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \quad (2.12)$$

In practice, we say that f is a Gaussian Process with mean $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ and write:

$$f(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \quad (2.13)$$

In practice, for simplicity we will take $m(\mathbf{x}) = 0$, but this can be specified otherwise. As stated before, a Gaussian Process fulfils the marginalization property, that is to say that if the GP specifies $(y_1, y_2) \sim N(\boldsymbol{\mu}, \Sigma)$ then this implies that $y_1 \sim N(\mu_1, \Sigma_{11})$. A Gaussian multivariate distribution is just a finite index set of a Gaussian process.

As seen in the previous section, a Gaussian Process can be viewed as a Bayesian regression model using a particular kernel, that is, for the model $f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$ and with the same prior as in equation 2.3 has mean and covariance functions:

$$\mathbb{E}[f(\mathbf{x})] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = 0 \quad (2.14)$$

$$\mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(\mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}') \quad (2.15)$$

It is now a good time to start specifying our first covariance function, the *squared exponential* kernel, defined as:

$$k(x, x') = \exp\left(-\frac{1}{2}|x - x'|^2\right) \quad (2.16)$$

Where $|\cdot|$ denotes the standard L_2 norm. Most of the covariance functions that we will see here are a function of this norm, therefore it is much more comfortable to write $r = |x - x'|$ and therefore the squared exponential kernel becomes:

$$k(x, x') = \exp\left(-\frac{1}{2}r^2\right) \quad (2.17)$$

The squared exponential covariance kernel is equivalent to a Bayesian linear model with an infinite number of basis functions. This also implies a distribution over functions. To see this, choose an arbitrary number of points X_* and compute the squared exponential kernel for those. Then just sample from the following multivariate Gaussian:

$$\mathbf{f}_* \sim N(\mathbf{0}, K(X_*, X_*)) \quad (2.18)$$

To illustrate this point, we will write a very simple Python script to draw (finite) samples from this function. For the moment, consider evenly spaced samples with a step of $\frac{\pi}{4}$. Some of the code here already uses the pyGPGO implementation of the covariance function, to simplify computation. This code produces Figure 2.1.

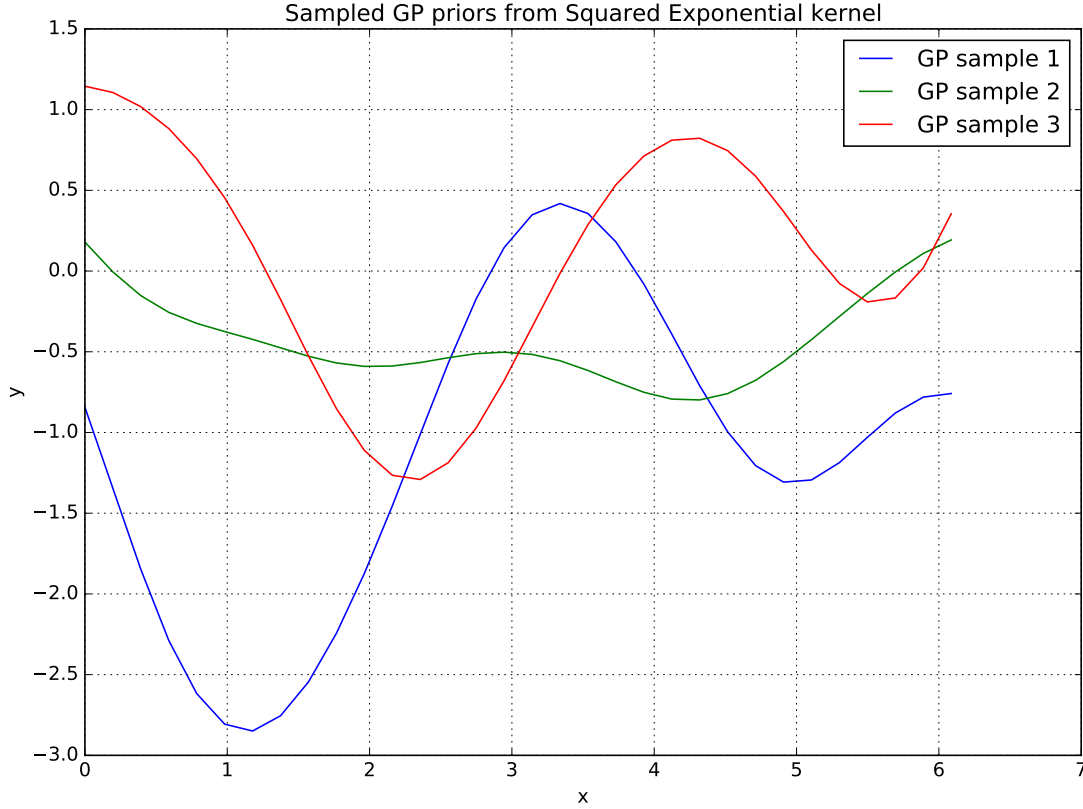
```

1 import numpy as np
2 from numpy.random import multivariate_normal
3 from covfunc import squaredExponential
4 import matplotlib.pyplot as plt
5
6 if __name__ == '__main__':
7     np.random.seed(93)
8     # Equally spaced values of Xstar
9     Xstar = np.arange(0, 2 * np.pi, step = np.pi/16)
10    Xstar = np.array([np.atleast_2d(x) for x in Xstar])[:, 0]
11    sexp = squaredExponential()
12    # By default assume mean 0
13    m = np.zeros(Xstar.shape[0])
14    # Compute squared-exponential matrix
15    K = sexp.K(Xstar, Xstar)
16
17    n_samples = 3
18    # Draw samples from multivariate normal
19    samples = multivariate_normal(m, K, size = n_samples)
20
21    # Plot values
22    x = Xstar.flatten()
23    plt.figure()
24    for i in range(n_samples):
25        plt.plot(x, samples[i], label = 'GP sample {}'.format(i + 1))
26    plt.xlabel('x')
27    plt.ylabel('y')
28    plt.title('Sampled GP priors from Squared Exponential kernel')
29    plt.grid()
30    plt.legend(loc = 0)
31    plt.show()

```

There's one important concept to explain before we move on. Notice in Figure 2.1 that the drawn functions seem to have a characteristic length-scale. This can be interpreted as the distance you have to move in input space before the function value changes significantly. By default, the squared exponential kernel uses a characteristic length-scale of 1 ($l = 1$). To change this behaviour to another, it is sufficient to consider r/l instead of r in Equation 2.17. This can be thought as an hyperparameter to optimize, but we will return to this in another section.

Figure 2.1: Three sampled Gaussian Process priors using the Squared Exponential kernel.



2.3 Prediction using a Gaussian Process prior

This is probably the most important section in this chapter. We will learn how to incorporate the knowledge of training data $D = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, n\}$ into our Gaussian Process to obtain a posterior predictive distribution. We will start considering the case that we have a noiseless function, that is to say, when $\sigma_n^2 = 0$. Let us define $K(X, X_*)$, the covariance function evaluated on train and test points, $K(X, X)$ the covariance function evaluated at only the training points, $K(X_*, X_*)$ equivalently defined for the test values. Notice the last two have to be square matrices by definition.

Let us also use the following theorem:

Theorem 1 *Let \mathbf{x} and \mathbf{y} be jointly Gaussian:*

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim N \left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \right) \quad (2.19)$$

Then $\mathbf{x}|\mathbf{y} \sim (\boldsymbol{\mu}_x + CB^{-1}(\mathbf{y} - \boldsymbol{\mu}_y), A - CB^{-1}C^T)$

The reader might already be guessing what we are about to do now in terms of the training output points \mathbf{f} and the corresponding testing points \mathbf{f}_* . According to the prior chosen in 2.18, assume that \mathbf{f} and \mathbf{f}_* are jointly Gaussian:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim N \left(\mathbf{0}, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right) \quad (2.20)$$

We are interested now in the distribution of $\mathbf{f}_*|\mathbf{f}$. Simply applying Theorem 1, we can obtain:

$$\mathbf{f}_*|\mathbf{f} \sim N \left(K(X_*, X)K(X, X)^{-1}\mathbf{f}, K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*) \right) \quad (2.21)$$

This is pretty much everything basic there is to know about Gaussian Process estimation for regression. Now we have a complete predictive distribution over test values \mathbf{f}_* , and we can do with it what we please. For

example, one could obtain an estimate of this function by drawing samples from a multivariate normal with the computed posterior parameters, or obtain a MAP estimate using the posterior mean.

Let us now consider the scenario where observations are not noise-free, that is, each time you query the function there is a Gaussian error with variance $\sigma_n^2 \neq 0$

Chapter 3

Bayesian optimization

Chapter 4

Experiments

Chapter 5

pyGPGO: A simple Python Package for Bayesian Optimization