Tuning Machine Learning Models with Bayesian Optimization

Vinícius O. Silva, Renato M. Assunção

[1]Computer Science Department – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

viniciusoliveira@dcc.ufmg.br, assuncao@dcc.ufmg.br

Abstract. *Bayesian optimization is a mathematical method that allows one to find the optimal arguments of an unknown function while observing its true outcomes as few times as possible. In an era where Machine Learning (ML) algorithms are growing in complexity while becoming more popular than ever, such a method seems an appropriate approach to fine tune the hyper-parameters that control these algorithms' behavior. This paper aims to discuss the mathematical foundations of the Bayesian optimization technique while focusing on its practical aspects, more specifically, how to use it to obtain a better performance when solving ML problems.*

1. Introduction

Nowadays, "Artificial Intelligence" (AI) is a term that is becoming more and more common in our daily lives. AI is being used across different industries and it's been demonstrating its capacity to augment our human capabilities in a wide variety of tasks [Systems 2018]. Applications of this technology include creating e-commerce recommendation systems [Smith and Linden 2017] and helping doctors to diagnose dangerous diseases, such as cancer, much earlier than by using previous techniques [Kourou et al. 2015].

Among the many sub-fields of Artificial Intelligence, *Machine Learning* (*ML*) draws special attention in the contemporary era. According to the book *Introduction to Machine Learning* [Alpaydin 2009], machine learning can be defined as "programming computers to optimize a performance criterion using example data or past experience". By using this definition and noticing that astonishingly high amounts of data are being generated every day [Laney 2001], it doesn't take much effort for one to understand why ML is gaining such a notorious status in our day and age [Shoham et al. 2018].

It turns out, however, that machine learning techniques are rarely hyper-parameter[1]-free. In their paper *Practical Bayesian Optimization of Machine Learning Algorithms* [Snoek et al. 2012], Hugo Larochelle and Jasper Snoek argue that often it is required that humans manually set these hyper-parameters, which makes developing ML models with many of them quite inconvenient. The authors also suggest that another approach that can be taken on this problem is to use some sort of black-box optimization

[1]We'll refer to *ML* parameters that need to be provided to the algorithms as *hyper-parameters* in order to distinguish them from those parameters that are obtained during the training process.

method in order to automatically find the appropriate values for these hyper-parameters. By doing so, it is possible to mitigate the impact of uninformed parameter choices made by machine learning practitioners who are not completely aware of all the nuances of their model or of the specific problem at hand.

A parameter optimization method that is of particular interest is called *Bayesian Optimization*. It is based on the probabilistic Bayesian framework and is adequate for optimizing hyper-parameters of machine learning models when the performance function is believed to be relatively smooth with respect to these hyper-parameters and whose performance is hard or expensive to evaluate [Brochu et al. 2010].

This was the case when developing the ML model that was the basis for the *Projeto Orientado em Computação I* (POC I) report. Due to fact that it was a Reinforcement Learning (*RL*) model that took a long time to run, experimenting different values for its hyper-parameters and then choosing the ones that yielded the best results was not a viable strategy. This particular problem of finding optimal values for hard to evaluate functions captured our attention and it was decided that it would be the focus of the coursework to be developed for the *Projeto Orientado em Computação II* (POC II) module.

Considering this whole context, we present a detailed research on how Bayesian optimization works when used to tune the hyper-parameters of machine learning models whose performance is computationally expensive to evaluate. We describe the mathematical theory that powers this method and build a prototype software to tune the hyper-parameters of a simple machine learning model.

2. Black-Box Optimization

Math and computer models have been for years powerful tools in the engineering field. From physical systems simulations to financial analysis and more recently, machine learning, math-based computer models have been used in a wide variety of applications. This approach, however, has proved to be a challenge to scientists and engineers in the sense that the implementations of such models may take a very long time to finish executing and often turn out to be financially expensive as well.

It is then, desirable that the executions of these computational models are done as efficiently as possible, avoiding any repeated work and maintaining its accuracy levels to an acceptable standard. This becomes a rather challenging task when we consider that the computer models we are interested in executing often require that a variety of hyper-parameters to be chosen by hand, heavily depending on subjective prior knowledge.

Machine Learning algorithms are clear examples of computational models that usually rely on human expertise in order to perform well. A simple feed-forward neural network, for instance, requires that the implementer choose how many neuron layers to be used, the number of neurons that will exist in each layer, the initial values for each synapse weight, what activation function will be used in each neuron, what optimization algorithm will be used while training, etc. Making the appropriate choice for each of those hyper-

parameters is not a simple job, especially considering that although the chosen values heavily impact on the neural network's performance, the correlation between each hyper-parameter and the network's observable behavior is not clear.

In order to overcome such difficulties, a whole domain of science has been working towards finding efficient methods to choose hyper-parameter values for computational models whose behavior is hard to describe. The name *"Black-box optimization"* comes from the fact that the scientists must figure out the best parameters for a function or system only by observing their outputs when an input is given. This means that in this case, it is not possible for one to have access to the derivatives of the true mathematical formulations that describe the system, and thus it is necessary that a certain degree of creativity to be used in order to solve the problem.

A number of algorithms are typically tried when facing black-box optimization problems. Among the most common ones, we can quote grid-search, random-search, evolutionary algorithms and Bayesian Optimization. The latter is the focus of this paper because it offers us the benefit of minimizing the number of function evaluations while it searches for the best hyper-parameters. This is a significant advantage when dealing with machine learning problems, since blindingly re-training the whole model for many combinations of hyper-parameters is a cost-prohibitive task.

Prior, however, to describing the Bayesian optimization method in detail, it is first necessary to understand the concept of Gaussian Processes (*GPs*). Although one can implement the Bayesian optimization algorithm without making use of GPs, the classical implementation relies on such mechanisms and the examples that will later be presented in this paper also depend on them. It makes sense then, for us to offer an introduction to this topic up to a level where it is sufficient for one to understand the code pieces that will be shown. For this reason, we will dive into the topic of GPs in the next section. We aim to provide a clear and concise explanation on it, enabling the reader to follow on the text with no major problems.

3. Gaussian Processes

A Gaussian Process can be thought as generalization of the Gaussian Probability Distribution. Whereas probability density distributions describes how random variables (vectors or scalars) behave, a Gaussian Process is a stochastic process that specify a distribution over functions [Pappa 2019]. Just as one can draw random values from a probability density distribution, it is also possible to draw random functions from a Gaussian Process.

In the same way a multivariate Gaussian distribution is completely specified by its mean and covariance, a GP is completely specified by its mean function, m and covariance function, k:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{1}$$

It is intuitive to think of a GP as analogous to a function, but instead of outputting a

scalar $f(\mathbf{x})$ for an arbitrary $\mathbf{x}$, it returns the mean and the variance of a normal distribution over the possible values of f at $\mathbf{x}$ [Brochu et al. 2010]. By observing Figure 1 it is possible for one to have a better understanding of what we mean when we refer to a Gaussian Process.

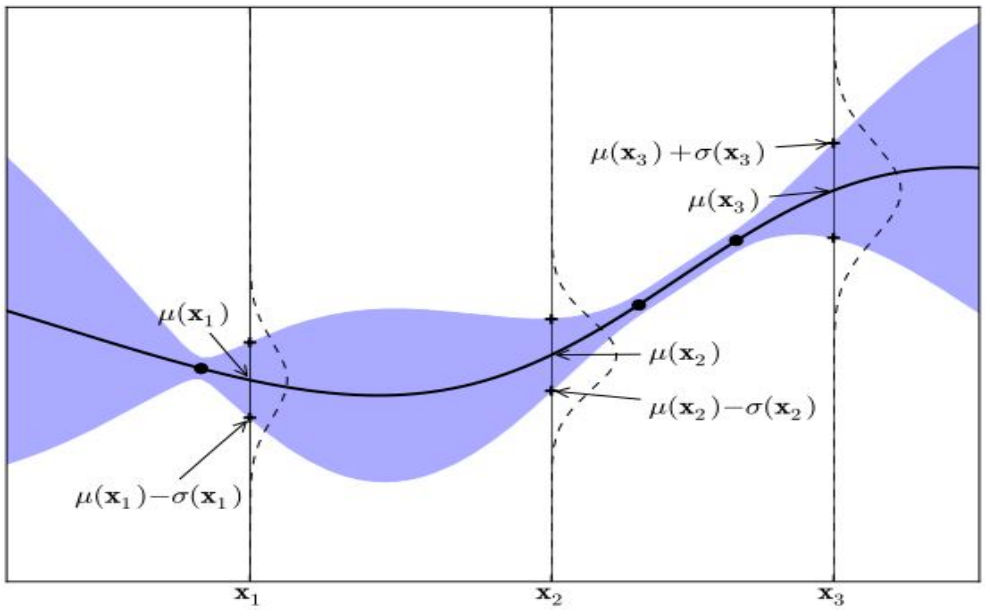

Figure 1. **Representation of a 1-D Gaussian Process with 3 observations. The solid black line is the GP's mean function and the blue shaded area shows the mean plus and minus the variance. The superimposed Gaussians correspond to the GP mean and standard deviation ($\mu(.)$ and $\sigma(.)$ of the points $\mathbf{x}_{1:3}$) [Brochu et al. 2010]**

One of the main applications (and the one we are interested in) of Gaussian Processes is to perform *non-parametric* regressions. By *non-parametric* we mean that one does not need to priorly specify if the unknown function will be approximated by a straight line (as in linear regression), a polynomial curve (as in polynomial regression), a logistic curve (as in logistic regression) or any other predefined formats. In GP regression it is assumed that the unknown function was sampled from a Gaussian Process and the training procedure consists in determining the mean function and covariance matrix of such process given the observed data points.

Training a Gaussian Process regression model comes down to solving two simple equations that come from the Multivariate Gaussian Theorem [Freitas 2013]:

Suppose $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is jointly Gaussian with parameters

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

The marginals are given by

$$p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1 \mid \mu_1, \Sigma_{11})$$
$$p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2 \mid \mu_2, \Sigma_{22})$$

And the posterior conditional is given by:

$$p(\mathbf{x}_1 \mid \mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1 \mid \mu_{1|2}, \Sigma_{1|2})$$

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \mu_2) \quad (2)$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \quad (3)$$

What these equations allows us to do is determining the mean function and the covariance matrix of a Gaussian Process that passes through the observed data points. In practice, the first thing that needs to be done is to consider a prior Gaussian Process with an arbitrary mean (typically $f(\mathbf{x}) = 0$) and a covariance matrix that is built by applying a *Kernel function* to all the pairs $(\mathbf{x}_i, \mathbf{x}_j)$ that belong to the domain of the unknown function (Equation 4).

$$\Sigma = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \quad (4)$$

The *Kernel function* is used to express the correlation between any given pair of points and heavily influences the smoothness of the functions that can be sampled from the GP. The Kernel function is a hyper-parameter of the GP regression algorithm, but we do not consider this as an disadvantage in the context of Bayesian Optimization. As one can incorporate their prior knowledge about the unknown function's smoothness into the algorithm by choosing an appropriate Kernel function, we consider this to be beneficial, since it can speed up the optimization process.

Among the many possible choices of Kernel functions, we will focus on a simple, but quite useful one: The parameterized squared exponential kernel, expressed in Equation 5:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\theta^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \quad (5)$$

By using such a Kernel function, we can quite easily vary the smoothness of the GP by simply choosing a value for θ that reflects our prior belief about the unknown function. In order to illustrate how the θ parameter affects the GP, we have sampled 5 functions from 3 Gaussian Processes whose only difference was the θ value used in their Kernel functions. The obtained results are shown in Figure 2.

Once the initial (*prior*) Gaussian Process is defined, we can start including the observed (*training*) data to our GP. In a noiseless scenario (which is the scope of this paper), this will restrict the Gaussian Process so that it will only be able to generate functions that pass through these observed data points. The addition of said data points to the GP consists of solving the posterior conditional equations (2 and 3), considering that

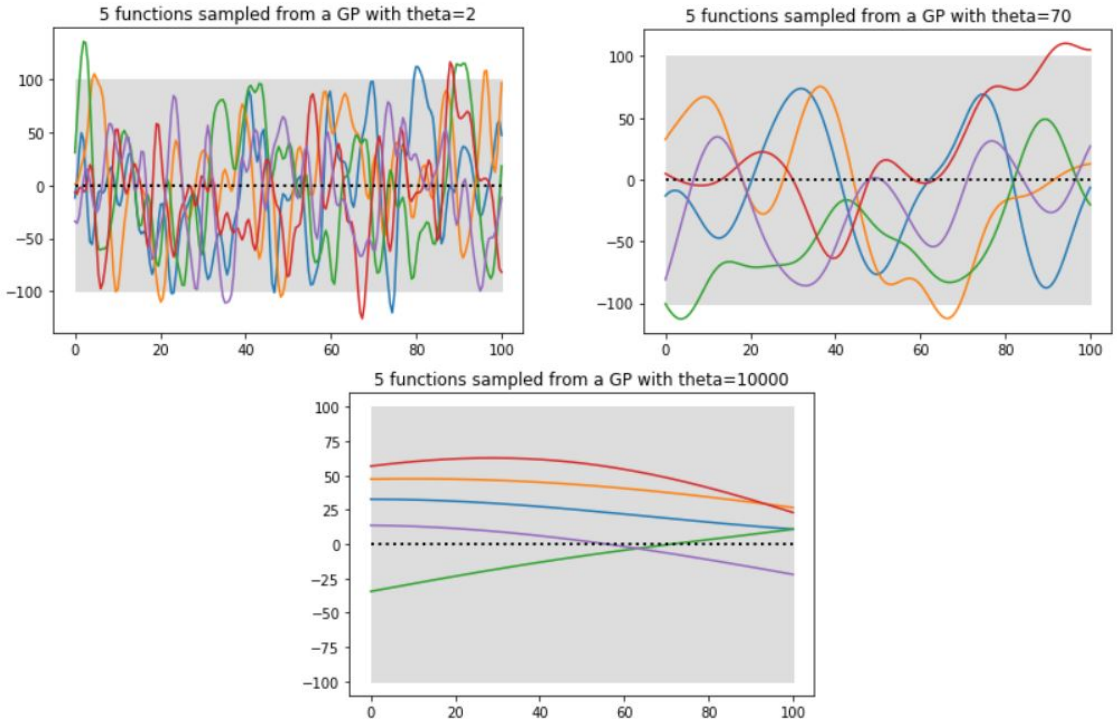

Figure 2. **Dotted black line represents the GP's mean function and the shaded gray area represents the GP's variance.**

the given data is the set of the observed evaluations of the unknown function. Figure 3 shows an example of a Gaussian Process approximating an unknown function by using 5 observed data points.

One last clarification that needs to be done before we approach the Bayesian Optimization algorithm itself refers to the problem of sampling functions from a Gaussian Process. The equations that enables us to do that are based on the process of sampling scalar values from a 1-Dimensional Gaussian Distribution. In the 1-D case, we know that a value $x \sim \mathcal{N}(\mu, \sigma^2)$ can be obtained simply by sampling a standard Gaussian Distribution ($\mathcal{N}(0, 1)$), multiplying it by the desired standard deviation and adding it to the desired mean as we can see from Equation 6:

$$x \sim \mu + \sigma\mathcal{N}(0, 1) \tag{6}$$

The multivariate case is quite similar to the univariate case we just presented. The sampling method is equivalent and the only change we need to make is to translate the scalar operations to their matrix counterparts. Instead of sampling a univariate Gaussian distribution ($\mathcal{N}(0, 1)$), we sample a standard multivariate Gaussian distribution ($\mathcal{N}(0, I)$) where I is the $n \times n$ Identity matrix. Analogously to what we did previously, we need to multiply the resulting sample to the square root of the desired variance. In this case, however, the distribution's variance is represented by a matrix and formally, taking the square root of a matrix is an undefined operation. In order to overcome this problem we must notice that a covariance matrix has the convenient property of being symmetric positive definite. This allows us to rewrite it as product of a matrix L and its transpose

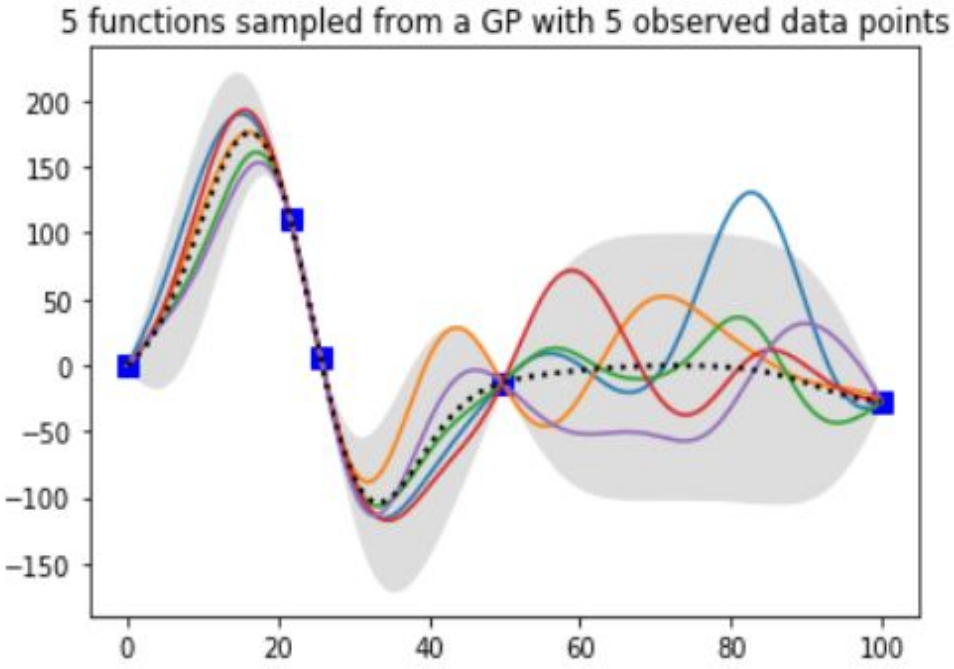

Figure 3. **5 functions sampled from a GP trained to include 5 data points. The blue squares represent the observed evaluations of the unknown function and the dotted black line represents the GP's mean. The kernel function used was the parameterized squared exponential with θ=70**

L^T:

$$\Sigma = L \times L^T \tag{7}$$

L can be obtained by running the *Cholesky Decomposition Algorithm* [Parker 2017] over the desired Σ matrix. This is a well documented procedure that runs in cubic time and outputs the matrix that we will informally consider as the "square root" of the GP's covariance. Once L is obtained, we can proceed to sampling functions of the desired Gaussian Process; equation 8 shows us how to do that.

$$f(\mathbf{x}) \sim \mu + L\mathcal{N}(0, I) \tag{8}$$

Equation 8: **Sampling a function from a Gaussian Process. Note that μ represents the desired mean function and hence, is a vector.**

4. Bayesian Optimization

Bayesian Optimization is a robust tool for optimizing unknown objective functions whose evaluation is costly. We are particularly interested in solving problems where the goal is to find the global maximum of an expensive function $f : \mathcal{X} \to \mathbb{R}$

$$\mathbf{x}_{\text{opt}} = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}} f(\mathbf{x})$$

within a domain $\mathcal{X} \subset \mathbb{R}^d$. When tuning machine learning models, this method of optimization is particularly adequate because typically, d is small and the models are often quite expensive to evaluate.

[Dewancker et al. 2015]

The basic idea behind this algorithm is to use evidence and prior knowledge to maximize the function at each step, so that each new evaluation decreases the distance between the true global maximum and the expected maximum given the model [Brochu et al. 2010]. In practice, each iteration of the Bayesian optimization algorithm consists of the maximization of a known function that operates over the same domain as the original unknown function. This known function is called **acquisition** or **utility** function and it quantifies how useful it is to sample each region of the hyper-parameter space of the objective function. By optimizing the acquisition function we find out what combination of hyper-parameters we should evaluate using the actual expensive function.

The Algorithm 1 shows the general idea of how Bayesian Optimization works:

for *t = 1, 2, ...* **do**
  Find $\mathbf{x}_t$ by optimizing the acquisition function over the GP:
  $\mathbf{x}_t = \text{argmax}_{\mathbf{x}} u(\mathbf{x} \mid \mathcal{D}_{1:t-1})$;
  Sample the objective function $y_t = f(\mathbf{x})$;
  Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ and refit the GP using the augmented dataset.
end

[Brochu et al. 2010]

In order to explain what the algorithm is doing in a visual manner, we will also show an illustration published by [Brochu et al. 2010] where it is possible to see 3 sequential iterations of the algorithm. By carefully analyzing Figure 4, we believe it is possible for one to grasp a clear idea of how Bayesian optimization chooses the points where to sample the original function.

4.1. Acquisition Functions

As we just covered, the role of the acquisition function is to guide the search for the optimal hyper-parameters. Since the optimal values are likely to be in regions where the GP's mean is high or where the variance (uncertainty) is great, we have to consider the exploration-exploitation dilemma [Sutton and Barto 1998] when choosing the appropriate acquisition function to use in our optimization. Three typical acquisition functions that are typically used in the context of Bayesian optimization for machine learning consider

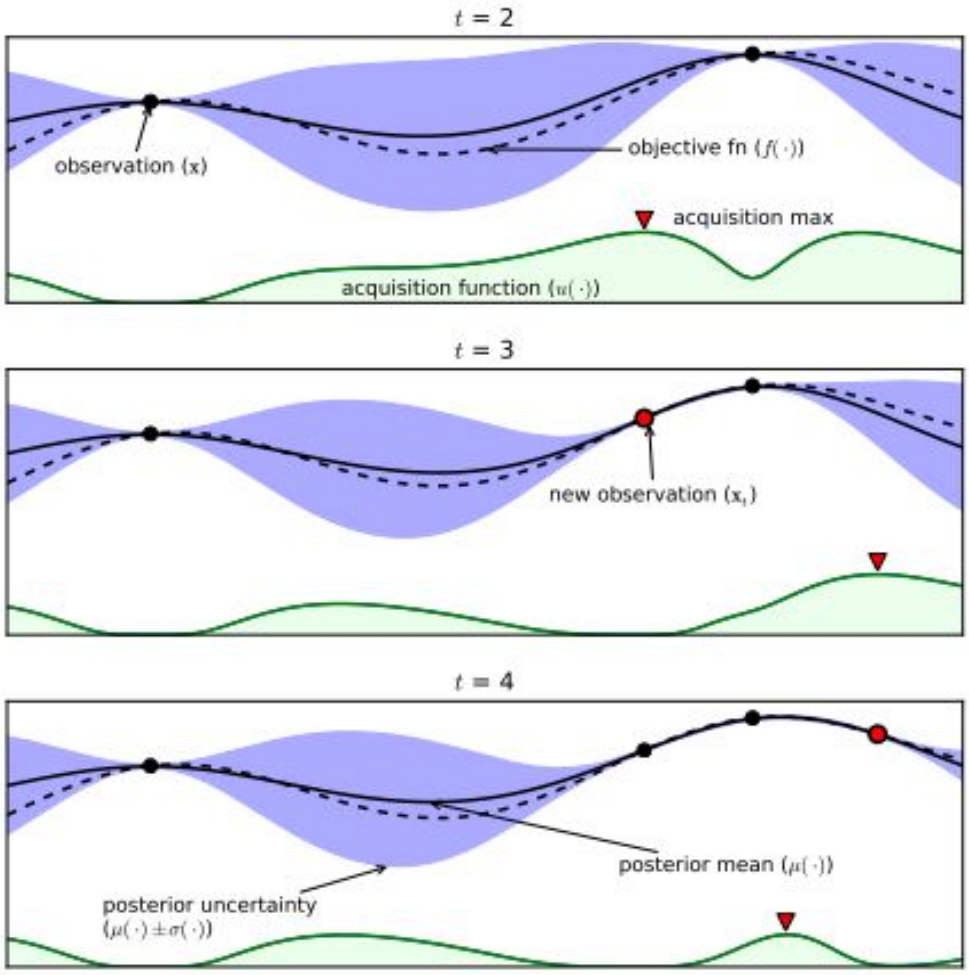

Figure 4. Three sequential iterations of the Bayesian optimization algorithm. [Brochu et al. 2010]

the trade-offs involved in exploiting regions in which we expect the evaluation of the objective function to be high and exploring regions in which we do not have a clear idea of what to expect. The ensuing subsections offer a brief introduction to the technical traits of each one of them.

4.1.1. Probability of Improvement Function

The Probability of Improvement method tells us to sample the unknown function at the point where it is the most likely that we will observe a value that is greater than the best value we have observed so far. This approach was proposed by H.J. Kushner [Kushner 1964] in 1964, but as it favors exploitation only, a slight modification was added to it: Adding a trade-off parameter $\xi \geq 0$ that controls how large an improvement needs to be in order to be considered for the optimization process. The equation that formalize the Probability of Improvement (PI) acquisition function is shown below:

$$PI(\mathbf{x}) = P(f(\mathbf{x}) \geq f(\mathbf{x}^+) + \xi)$$

$$= \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})}\right)$$

The Φ function is the Cumulative Density Function (CDF) of a Gaussian Distribution and $\mathbf{x}^+$ is the point where the highest observed evaluation of the unknown function f occurs.

Figure 5 shown below provides a visual clarification of the criteria that is used by the Probability of Improvement Function when deciding which point to sample next:

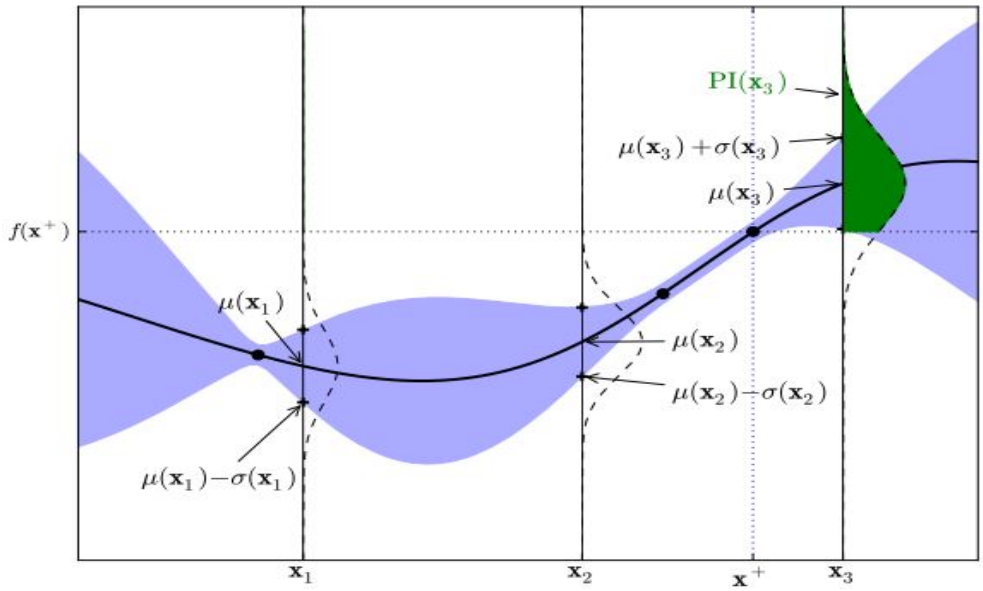

Figure 5. **The maximum observation is at $\mathbf{x}^+$. The green shaded area in the superimposed Gaussian above the dashed line can be used as a measure of improvement. The model predicts nearly no probability of improvement at $\mathbf{x}_1$ or $\mathbf{x}_2$, while sampling at $\mathbf{x}_3$ is more likely to improve on $f(\mathbf{x}^+)$.**
[Brochu et al. 2010]

4.1.2. Expected Improvement Function

This is one of the most adopted strategies when choosing an acquisition function for the Bayesian optimization algorithm and our choice for the examples that will be shown in the next section. Its main advantage over the Probability of Improvement function is that it does not consider only how likely it is to find a better point than what we already have, but also the magnitude of the improvement this new point can yield. The goal of the Expected Improvement strategy is to sample the unknown function at the point that minimizes the

expected deviation from the true maximum ($f(\mathbf{x}^*)$). Formally, at each iteration we acquire the next observation $f(\mathbf{x}_{t+1})$ such that:

$$\mathbf{x}_{t+1} = \underset{\mathbf{x}}{\operatorname{argmin}}\ \mathbb{E}(\|f_{t+1}(\mathbf{x}) - f(\mathbf{x}^*)\||\mathcal{D}_{1:t}) \tag{9}$$

In practice, the function we optimize is a derivation from Equation 9, shown in Equation 10. The math work that yields this result is intentionally left out of this paper due to its complexity, but the derivations can be found in [Jones et al. 1998].

$$EI(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+))\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0. \\ 0 & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{10}$$

$$Z(\mathbf{x}) = \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}$$

$\Phi(.)$ and $\phi(.)$ indicate the Gaussian Cumulative Distribution Function (CDF) and the Gaussian Probability Distribution Function (PDF), respectively.

4.1.3. Thompson Sampling

This is the simplest of the three alternatives we describe. The Thompson Sampling strategy consists of drawing a function of the Gaussian Process (see Figure 3) and using the location where its peak occurs as the next point to sample from the unknown function. Although this is a simple approach, it has shown good results in practice [Scott 2015].

5. Bayesian Optimization Example

In order to provide an easy-to-follow illustration of how the Bayesian Optimization algorithm works in practice, we have written a toy example that implements all the concepts discussed in this paper. The code was developed using the Python language and is publicly available at [Silva 2019a].

In the code we try to optimize the function $f(x) = x \sin(\frac{x}{6})$ over the range between 0 and 100. We chose this function because it is smooth and contains multiple local maxima, just as we expect from the behavior of a real Machine Learning model. We tried to make the code simple, but actively avoided libraries that would encapsulate the Bayesian Optimization algorithm or the operations involved when dealing with Gaussian Processes.

During the execution, we generated graphs that provide a visual idea of what the algorithm is doing at each iteration. We present those graphs in Figure 6.

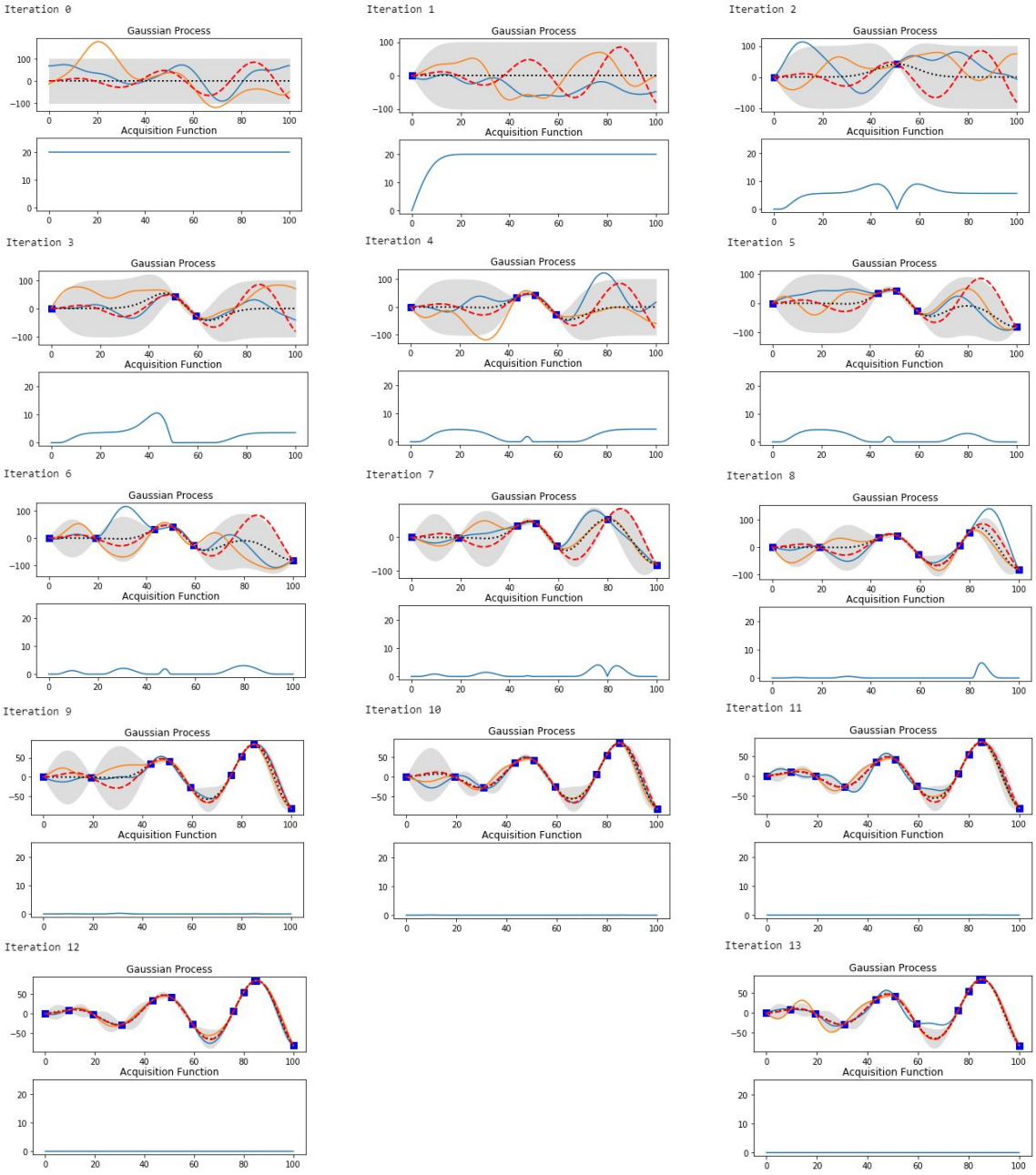

Figure 6. **Execution steps of the Bayesian Optimization Algorithm until convergence.**

6. Machine Learning Tuning

In order to show how one can use Bayesian Optimization to tune an actual Machine Learning model, we are also making available a Jupyter notebook that contains a simple guide on how to optimize an XGBoost model in a regression task. The code is available as gist in Github and can be accessed at [Silva 2019b].

We encourage the reader to run the code an try a variety of experiments on it. Our goal in publishing this notebook is to make Bayesian Optimization accessible for a wide range of Machine Learning practitioners. Differently to the first notebook where we avoid encapsulating key concepts by using libraries, in the second one we make use of every tool available. The reason for that is that at this point we assume that the reader is familiar with the concept of Bayesian Optimization and thus it is not necessary to write everything from scratch.

7. Conclusion and Future Work

The development of this coursework allowed us to prepare a broad literature review in the Bayesian Optimization topic and to get a deeper understanding of why it is important to fine tune the hyper-parameters of a machine learning model while keeping the number of trials to a minimum.

We also had the opportunity to show in practice how one can use such a powerful method in order to choose the hyper-parameters of a model, and we believe that by doing so, we will make it possible to Data Scientists and Machine Learning Engineers to deliver better products and obtain better results in their day-to-day operations.

For the future, we aim to improve our understanding of the algorithm and to investigate how one can use it to tune ML models whose hyper-parameter space is not static. An example of such model is a neural network, in which the hyper-parameters are dependent of each other. In this specific model, if the number of neuron layers is increased, the dimensionality of the hyper-parameter space increase as well (i.e.: if the model has 3 layers, we need to add 3 hyper-parameters that control the activation function of each layer, for instance). This kind of problem is a serious limitation of the approach described in this paper and is certainly drawing our attention to future research in the field.

Finally, we would like to conclude by stating that the development of this coursework has been challenging but rewarding. Although we did not contribute with innovative tools and techniques, we believe that our goal has been achieved. We intended to get a solid understanding of the Bayesian Optimization field and the preparation of this report drove us towards this goal.

References

Alpaydin, E. (2009). *Introduction to machine learning*. MIT press.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Dewancker, I., McCourt, M., and Clark, S. (2015). Bayesian optimization primer.

Freitas, N. (2013). Gaussian processes. Lecture slides.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492.

Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V., and Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106.

Laney, D. (2001). 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1.

Pappa, G. L. (2019). Automated machine learning bayesian optimization. Lecture slides.

Parker, M. (2017). Chapter 13 - matrix inversion. In Parker, M., editor, *Digital Signal Processing 101 (Second Edition)*, pages 149 – 162. Newnes, second edition edition.

Scott, S. L. (2015). Multi-armed bandit experiments in the online service economy. *Applied Stochastic Models in Business and Industry*, 31(1):37–45.

Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., Manyika, J., Niebles, J. C., Lyons, T., Etchemendy, J., and Bauer, Z. (2018). The ai index 2018 annual report.

Silva, V. (2019a). Bayesian Optimization From Scratch. `https://gist.github.com/silva-vinicius/e94a3f55289e46bcf4c29e16404e767e`. [Software].

Silva, V. (2019b). Practical Bayes Optimization. `https://gist.github.com/silva-vinicius/323656e98a3c24a49f51ffd054ca3f34`. [Software].

Smith, B. and Linden, G. (2017). Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Systems, I. P. (2018). The real-world business of ai.