Bachelor's Thesis

Study on Automated Light Gradient Boosting Machine: Efficient Algorithms for Hyper-parameter Optimization

Lam Gia Thuan

Matriculation Number: 1148372

supervised by

Prof. Dr. Doina Logofatu

co-supervised by

Prof. Dr. Christian Andersson

Frankfurt University of Applied Sciences
Vietnamese-German University

January 2020

Declaration of Authorship

I hereby declare that I have authored this bachelor thesis independently, that I have not used any sources other than those listed in the bibliography and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

LAM GIA THUAN

Acknowledgement

Thank you first and foremost to all my thesis supervisors, Prof. Dr. Doina Logofatu and Prof. Dr. Christian Andersson for giving me the opportunity to implement this thesis project which is itself the biggest project ever in my life.

I would like to express a special thank to Prof. Dr. Doina Logofatu for leading me into the world of machine learning and data science through the exciting GECCO Challenge 2017 - Monitoring of drinking-water quality - which serves as the main inspiration for this thesis, and also for all her kind support and encouragement for me during all my years at the Frankfurt University of Applied Sciences. Meeting and working with Prof. Logofatu is undoubtedly the best thing happening since the start of my higher education.

Another thank to my second supervisor, Prof. Dr. Christian Andersson for being the same teacher who taught me about Linear Algebra. The knowledge of matrix operations has been a big help for understanding the concepts needed for my thesis, which I never expected :).

Last, but not least, thank you to the Frankfurt University of Applied Sciences and the Vietnamese-German University for creating this incredible opportunity. I would be remiss without mentioning some professors who have had a particular influence on my work, and on my appreciation of Computer Science: Prof. Dagmar Rühl for always helping me, Prof. Michel Toulouse and also Ms. Trang Nguyen Thi Thuy for always tolerating my unreasonable requests.

Abstract

Rarely any machine learning algorithms exist without hyperparameters whose values have an unparalleled impact in the performance of the resulting model. That is why finding their optimal values is a must for any machine learning tasks. The process is, however, extremely challenging in that machine learning problems have certain properties that intensify the complexity of hyperparameter optimization compared to the normal black-box optimization problem - an open and on-going research topic itself. Can this process be automated?

This work studies the most popular algorithms for automating hyperparameter optimization, namely Bayesian Optimization and Random Search, and applies them on the famous Light Gradient Boosted Machines (LightGBM) algorithm for benchmarking. To supplement our research, an implementation called KotlinML (Kotlin for Machine Learning) is created which is also the first Kotlin library of its kind with support for LightGBM, Bayesian Optimization and related algorithms. In addition, a variety of experiments were conducted using KotlinML to verify multiple phenomena in the world of machine learning and hyperparameter optimization with detailed instructions and analysis. Albeit many open questions have been left unanswered due to various restrictions on project scope and computing resources to ensure a timely completion, this thesis has succeeded in completing its fundamental goal of understanding the most popular algorithms for hyperparameter optimization and contributing a highly practical framework to the world. The framework is publicly available at: https://github.com/duckladydinh/KotlinML.

Contents

List of Tables

List of Figures

List of Listings

List of Algorithms

Chapter I

Introduction

1. Motivation

The workflow of data science is simple at first glance (Fig. 1.1), but being well-regarded as the sexiest job of the 21^{st} century implies an equivalent level of expertise to practice this profession. In this area, a state-of-the-art algorithm is insufficient as an acceptable solution, but it is how well the algorithm is utilized that will decide the resulting performance, leading to the weird situation where age-old but well-tuned models of the last decades are often superior to their modern but poorly-optimized successors. The job of data scientists is, therefore, challenging in that hyperparameter optimization is an art, not for the unskilled and impatient.

The rationale behind its complexity lies in the number of configurable parameters accompanied with every algorithm in machine learning and data science, together creating infinitely many possible combinations. A slight deviation from one to another might result in a huge loss in accu-

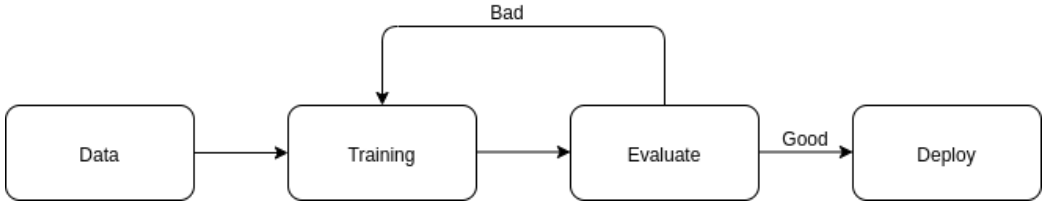


Figure 1.1: A simplified workflow for Data Science

Table 1.1: Standard Requirements

Category	**Requirements**
Optimization algorithms	Bayesian Optimization, Random Search
Learning Objective	Binary Classification
Supported Model	LightGBM
Supported Language	Kotlin

racy, which makes it practically impossible for a given model to unleash its full potential. Consequently, data scientists often neglect this important but difficult step and resort to naive methods such as grid search or randomization for an appropriate result, which may have a negative impact on the research and development of new machine learning algorithms in that old models are still relatively good and the latest ones, with more parameters, are unmanageable. *If machine learning models can 'just' work out-of-the-box, how different things would be?*

This puzzle has always been on my mind since my first machine learning project at the Frankfurt University of Applied Sciences, in which I did personally experience the pain and hardship from the pointless adventure to search for a good combination of parameters among innumerable possibilities. It was like wandering in the mysterious void universe where neither sign nor direction exists and all could be done was to try this and that manually for all eternity. That is why this is perfect as the final quiz I would love to solve before my graduation, and what is more, I strongly believe that this topic has its own merit in that its impact is potentially equivalent to that of steam power to the First Industrial Revolution since machine learning has already dominated the current era in its current semi-automated state.

Within the scope of this thesis project, I would like to make my first step into the area of Automated Machine Learning (AutoML) by implementing a framework for hyperparameter optimization, with the first supported citizen being Light Gradient Boosting Machine (LightGBM).

2. Project Scope

Within the scope of this thesis, the resulting product will follow, but not restrict to, the requirements specified in Table 1.1.

Additional features will be added afterwards, but those basic requirements are compulsory for limiting the unpredictability of our research and ensuring the successful completion of this project in a timely manner. The choice of those constraints will be explained in subsequent chapters. Nonetheless, they will not alter the challenging nature of this topic.

3. Thesis Contribution

This thesis researches the currently most important algorithm for hyperparameter optimization - Bayesian Optimization with Gaussian Process - and provide an implementation in Kotlin for reference, namely **KotlinML** (Kotlin for Machine Learning) for technical reason. The project is publicly available at: `https://github.com/duckladydinh/KotlinML`. Albeit the soul of our research is an already well-studied algorithm, KotlinML is proudly an original contribution to the world of Machine Learning and Data Science for introducing:

1. The 1^{st} ever Kotlin API for LightGBM.
2. The 1^{st} ever Kotlin API for L-BFGS-B algorithm.
3. The 1^{st} ever Kotlin implementation for Bayesian Optimzation and Gaussian Process.

In addition, we also explore some some fundamental concepts of Machine Learning such as verifying the No Free Lunch Theorem and analyzing the current metric for Hyperparameter Optimization (see chapter **Research & Experiments**), as well as deriving intuitive explanations for hard concepts such as Bayesian Optimization and Gaussian Process (see chapter **Preliminaries**) without abusing the complex mathematics as in classic publications.

As a side note, even if our claim to be the *first ever* is currently without public acknowledgement, we are absolutely confident of our contribution as the first of its kind given the very new status of Kotlin in Data Science and our consecutive verifying attempts using the omnipresent Google Search Engine during the implementation of this project.

The public introduction of KotlinML in the future will partially contribute to the acceleration of Machine Learning on the Java-Virtual Machine (JVM) [1] and the growth of Kotlin as the new language for Data Science, rivaling the de factor Python [2] and replacing Scala [3] - the current king of data science on JVM.

Table 1.2: Abbreviations and Acronyms

Short Form	**Full Form**
HPO	Hyperparameter Optimization
BO	Bayesian Optimization
API	Application Programming Interface
JVM	Java Virtual Machine
NFLT	No Free Lunch Theorem
UCB	Upper Confidence Bound

4. Thesis Structure

The remaining of this thesis contains 3 primary chapters, a brief summary of each is presented as follows:

Chapter II Preliminaries will introduce the basic concepts required for the understanding of subsequent chapters. In most cases, instead of detailed analysis of each and every concept, our self-derived intuitions will be given with relevant references for avid readers.

Chapter III Software Product will outline our final project in as much details as possible, including an overview of main components and examples on how to use them.

Chapter IV Research & Experiments will explore other areas of machine learning and data science with LightGBM by means experiments and provide some experimental results for ensuring the quality of our product.

After that comes **Summary and Conclusion** which will summarize and re-emphasize the contributions of our thesis. Last, but not least, also in this chapter, we will outline some possible directions for future research on this topic.

5. Abbreviations and Acronyms

Throughout this documentation, some abbreviations and acronyms are used for convenience. Table 1.2 covers those whose full and short forms may be used interchangeably.

Chapter II

Preliminaries

---

1. What is Machine Learning?

At the core of almost every popular applications of this era - from the search engine (Google) no netizen could live without, to the social network (Facebook) that has become part of our everyday life - lies a single common thing: Machine Learning. However, what exactly is Machine Learning? In his popular course [4], Prof. Andrew Ng defined it as *the science of getting computers to act without being explicitly programmed*. That explanation is however unable to provide any hints from the technical point of view. Alternatively, a more insightful intuition for fans of mathematics can be summarized in (2.1).

$$y = f(x) \tag{2.1}$$

In mathematics, given a function f and its input $x = (x_1, x_2...x_n)$ as a vector of parameters, mathematicians can calculate the output y, but the job of a machine learner is on another dimension of complexity, in which we are *given x and y and are required to figure out the underlying f*. One or more pairs of (x, y) is known as *data*, f is a machine learning *model* and the art of finding a model from data is known as *training* or *learning*. This form of learning is referred to as supervised learning, but within the scope of this thesis, understanding other types of machine learning is unneeded.

What could f be? It could be anything, ranging from an identity function ($f(x) = x$), to an exponential one ($f(x) = e^x$), or even something inexpressible by the current form of mathematics such as a way to map from a picture of oneself to his or her future appearance. How could something inexpressible be found? The answer is unsurprisingly simple: No way.

Finding a perfect function f from a set of data is practically impossible, but it is not infeasible to find its approximation g which is as close as possible to f. Their closeness must be measurable by some standard metric function $\mathcal{L}$, called *loss* or *cost function*. The lower the loss function, the closer g is to f. The end goal of a learning algorithm is to search for a function g which minimizes the cost incurred from a given cost function.

$$g = \underset{f \in \mathcal{F}}{\operatorname{argmin}}(\mathcal{L}(y, f(x))) \tag{2.2}$$

where $\mathcal{F}$ denotes the space of all possible functions.

The technique of finding such a function could be explained briefly in 100 pages [5], but within the scope of this thesis, it is sufficient to assume that algorithms for making it happen have been known and well-researched. Denoting one such algorithm as a black-box function $\mathcal{T}$, (2.2) can be simplified into (2.3).

$$g = \mathcal{T}(x, y) \tag{2.3}$$

Every function has, in addition to their input parameters x, other internal parameters that control its evaluation, as illustrated in Example 1. Among them are those from g which could be learned in the training such as weights, biases or split points and those from $\mathcal{T}$ that must be known before the training begins - *hyperparameters*. Unlike the former, hyperparameters also include numerous parameters of g depending on the model of choice and the search for their optimal values is an art known as *hyperparameter optimization*.

Example 1.

Function $f(x) = x_1$ has parameters $\theta = (1)$.

Function $f(x) = x_1 + 2x_2$ has parameters $\theta = (1, 2)$.

Function $f(x) = x_1 + 2x_2 + 3x_3$ has parameters $\theta = (1, 2, 3)$.

2. Hyperparameter Optimization

2.1. Definition

By definition, *hyperparameters (HPO) are the internal parameters of the model or its training algorithm that govern the training process and must be known before the training begins*. For example, most training algorithms would contain a regularization parameter which is responsible for balancing the performance difference of a given model on training and real-world datasets. Specific algorithms have their own specific hyperparameters, with different meaning and responsibility, and understanding even the simplest one is non-trivial and off-topic for this report since they will be optimized as those of a black-box function. Nonetheless, those hyperparameters are of unparalleled importance to any training algorithms. A good selection of them could turn the simplest models into gold, while a bad choice could destroy the most state-of-the-art models. That is why optimizing hyperparameters is a must for any machine learning tasks.

2.2. Current State of HPO

This important step, unfortunately, does not get the attention it deserves due to its overwhelming complexity. Researchers usually resort to naive methods such as trying some sets of chosen (grid search [6]) or random (random search [7]) values and opt for one that maximizes the performance on some validation set (a portion of the data that has not been used in training). These methods are, however, too simple and hence could leave an impression that something else could be done and there is room for further improvement. Is there a better algorithm?

To the mass of machine learning practitioners, searching for optimal hyperparameters is exactly like optimizing a black-box function [8] which is an old research topic with a rich legacy of efficient algorithms. However, machine learning problems have certain characteristics that distinguishes HPO from the rest and renders various optimization techniques unusable. Followings are a brief overview of those properties:

- **Lots of data**. Following the growth of big data, machine learning in this century rarely involves small amount of training data, which

makes their training extremely costly. Population-based optimization techniques such as evolutionary algorithms [9], ant colony optimization [10] or particle swarm optimization [11] usually require a huge number of computations and hence are unfriendly for this task.

- **Derivative-free**. Formulating the analytic form of a training algorithm's derivative at a given set of hyperparameters is a demanding task even if it exists. That immediately disqualifies various gradient-based techniques such as Newton-Quasi method [12], L-BFGS-B [13] or Gradient Descent [14]. Although numerical methods [15] provide a painless alternative to compute the derivative (if exists), it is too costly to be practical.
- **No efficient transition technique**. Various search algorithms require a transition method to find the next state from the current state, but for HPO, from a current state there is no known standard for transitioning to another state. The reason is that hyperparameters are very different from one to another, with different scales and meaning, even variable across datasets. Hence, finding a next state from current is not much different from searching for a random state. That limits the use of numerous popular methods such as simulated annealing [16].

For the above-mentioned reasons, the currently available options for HPO are so few that almost every known algorithm would revolve around a common technique, called Bayesian Optimization - an efficient technique that is known to work well with expensive, derivative-free black-box function and requires no method for state transition. This thesis studies this algorithm and benchmarks it against the naive random search method. The majority of other alternatives such as the recent Bayesian Optimization and Hyperband (BOHB) [17] are simply variants of Bayesian Optimization and are beyond the scope of this thesis.

2.3. Hyperparameter Metric

As do all optimization problems, HPO requires an objective function. In this case, the objective function is a mapping between a set of hyperparameters to the expected performance of the model it could generate on real-world datasets. The formulation is exactly like (2.1), only with different meanings for x, and y, in which x is a vector of hyperparameters and y is its generated model's expected performance. The most popular

choice so far has been the *k-fold cross validation* [18].

In k-fold cross validation sometimes called rotation estimation, the dataset D is randomly split into k mutually disjoint subsets (folds) $D_1, D_2 \dots D_k$ of approximately equal size. The given model is trained and tested k times. Each time $i \in \{1, 2 \dots k\}$, it is trained on D/D_i, tested on D_i and produces a certain score by some model performance metric such as F1 score [19] or Area Under the Curve (AUC) [20]. The mean and variance of all k scores are summarized for analysis, but usually *only the mean* would serve as the performance metric for HPO in automated processes. It is worth emphasizing that performance metrics for a model (after training) and a set of hyperparameters (used for training) are different and let's denote them as $\mathcal{M}$ (for model) and $\mathcal{H}$ (for hyperparameters), respectively, for future reference.

Nonetheless, an efficient metric for estimating the real-world performance of a model has yet to exist as demonstrated in chapter IV. In simpler terms, even if a set of hyperparameters that could produce a model which maximizes our cross validation on the training data is found, it does not guarantee the same performance in real-world datasets.

3. Bayesian Optimization

3.1. A Brief Intuition

Bayesian Optimization (BO) [21] is the most important technique known for hyperparameter optimization, which is essentially a black-box optimization method for *costly and derivative-free functions*. Almost all literature in existence about this topic would start off by explaining about conditional probability and the Bayes' theorem - from which its name is inspired. The overuse of mathematics is both complex and confusing in that it could be freed from the general idea of Bayesian Optimization. The concepts of conditional probability could have been contained solely in the surrogate model and Bayesian Optimization would be more comprehensible for those with a basic understanding of just distributions [22] or less. This section will discuss this algorithm from the perspective of a computer scientist, which may or may not align to the formal definition set by some math geeks.

In Bayesian Optimzation, the original expensive objective f is substituted by an inexpensive function g, which is commensurate to f and can be used for searching for the best location x - vector of hyperparameters - to

evaluate by f. The rationale behind this is because f is expensive and we would prefer to avoid calling it as much as possible.

This cheaper function is a composition of a probabilistic model - *surrogate* $\mathcal{S}$ - and an *acquisition function* $\mathcal{A}$. In other words,

$$g(x) = \mathcal{A} \circ \mathcal{S}(x) \tag{2.4}$$

Intuitively, a probabilistic model is just a mathematical function as formulated by (2.1), but the only different is that y is now a distribution of possible outcomes instead of a single number. In detail, it would return 2 values - mean μ and variance σ^2 - which defines some distribution as seen in (2.5). The acquisition function will then analyze this distribution and summarize the information in a single scalar that indicates our confidence of the current x. If $g(x)$ is high, it is more likely that $f(x)$ will also be good. This transforms an optimization problem to another, but on a much cheaper function and hence could be efficiently optimized by numerous algorithms for normal black-box optimization.

$$(\mu, \sigma^2) = S(x) \tag{2.5}$$

In addition, Bayesian Optimization is interactive, which means that the surrogate $\mathcal{S}$ is not constructed only once at the beginning but will be reconstructed interactively, each time more accurate than the former. In each iteration, it will search for a good x' based on the current $g = \mathcal{A} \circ \mathcal{S}$, compute $y = f(x')$ and update $\mathcal{S}$ with the new data (x', y). The general picture of Bayesian Optimization is summarized in Algorithm 1.

The real complexity of Bayesian Optimization lies in the surrogate model $\mathcal{S}$ and acquisition function $\mathcal{A}$ in which the hardship of probability resides. Nonetheless, treating them as black-box functions simplifies Bayesian Optimization to just a few lines of pseudocode as depicted in Algorithm 1. Even simpler, ignoring both $\mathcal{S}$ and $\mathcal{A}$, the whole algorithm will only depend on g - just some function cheaper than f - and can be free from all concepts of probability. This is the missing explanation for computer scientists that could not be found through countless math-abusing articles.

How can $\mathcal{S}$ and $\mathcal{A}$ be implemented? Why does $\mathcal{S}$ return a distribution? How can $\mathcal{A}$ summarize information from $\mathcal{S}$? All these lingering questions will be answered intuitively in subsequent sections.

Algorithm 1 Bayesian Optimization

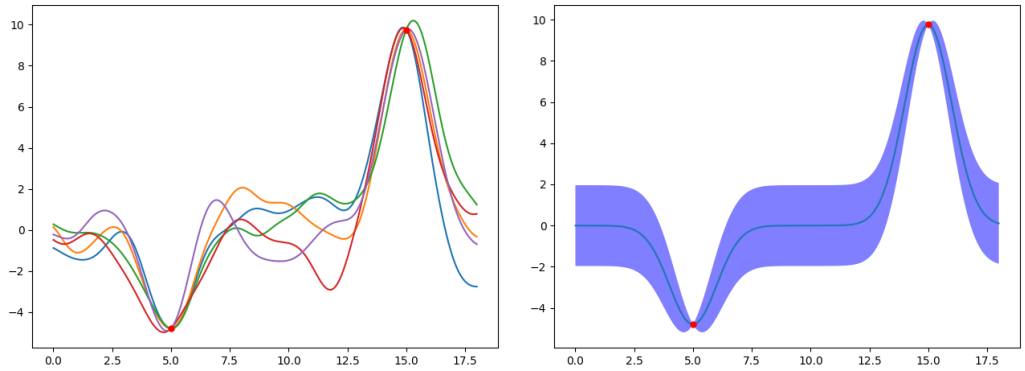
Make $\mathcal{S}$ from initial (x, y)
$x^* \leftarrow null, y^* \leftarrow -\infty$
for $i \in \{1, 2 \dots N\}$ **do**
  $x' \leftarrow \underset{x \in \mathcal{X}}{\text{argmax}}(\mathcal{A} \circ \mathcal{S}(x))$
  $y \leftarrow f(x')$
  if $y^* < y$ **then**
    $x^* \leftarrow x'$
    $y^* \leftarrow y$
  end if
  Update $\mathcal{S}$ with (x', y)
end for

3.2. Related Work

The simple design of Bayesian Optimization gives it the flexibility to evolve. By mutating $\mathcal{A}$ and $\mathcal{S}$ with appropriate acquisition functions and surrogate models, respectively, a variety of new algorithms could be born with their own interesting properties. Hence Bayesian Optimization acts more like a framework rather than an individual algorithm. Studies on this topic has existed since the last century with various results on what could integrate well with the BO framework.

For surrogate models, the traditional and probably the first candidate is Gaussian Process [23] which implies that its resulting distributions will be Gaussian (normal distributions) [24]. This method has demonstrated state-of-the-art performance in various machine learning applications where the input vector is consisted of only real numeric features (which is most of cases) and hence still enjoys high popularity even after decades of presence. Recently, research on this topic has involved other types of distributions including the inverse Wishart [25] and Student-t distribution [26], and gave birth to a promising alternative called Student-t Process [27], but they are not yet as popular as the traditional model. Last, but not least, another lesser known model worth mentioning is Random Forests [7] - a well-known machine learning model which has also proven to work well as a surrogate for data with categorical features.

In the area of acquisition functions, the traditional methods still remain relevant. They include, but not restrict to, Upper Confidence Bound (UCB) [28], Expected Improvement (EI) [29], Knowledge Gradient (KG)



(a) Sample Draws of 5 (b) Mean and Variance

Figure 2.1: A sample Gaussian Process of 2 points

[30], Entropy Search (ES) [31] and Probability of Improvement (PI) [32]. Nevertheless, most applications would employ EI and UCB by default for their superior performance.

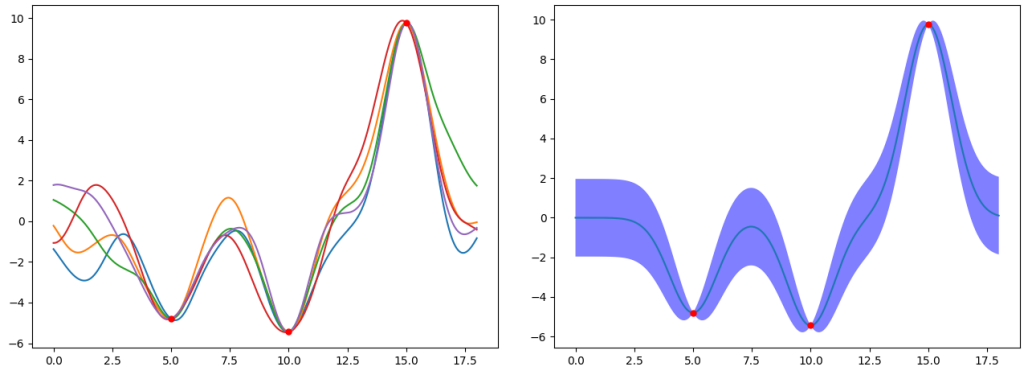
In this project, only Gaussian Process and UCB will be implemented. The former has been known as the de facto standard for Bayesian Optimization, whilst for the former, it is the simplest of all, which works well and is free from the burden of probability (at least for using it).

3.3. Gaussian Process

An intuitive explanation

Probably the best-known companion to Bayesian Optimization, a Gaussian Process (GP) is mathematically defined as a *probability distribution over all possible functions*. This popular definition of Gaussian Process is, however, far beyond the comprehensible realm of mortals with just a novice understanding of probability. Alternatively, a Gaussian Process could be regarded as *a collection of points* (x, y) and *all the functions that go through all these points*. An intuitive example of a Gaussian Process in the 2-dimensional space could be visualized in Figure 2.1. As can be seen in 2.1a, all functions - just lines in space - controlled by a Gaussian Process intersect at a certain number of points. There may be infinitely many such functions, but most of them are contained in the shaded area in 2.1b.

From the previous subsection, it has been mentioned that a probabilistic model such as Gaussian Process would evaluate to a distribution instead



(a) Sample Draws of 5 (b) Mean and Variance

Figure 2.2: A sample Gaussian Process of 3 points

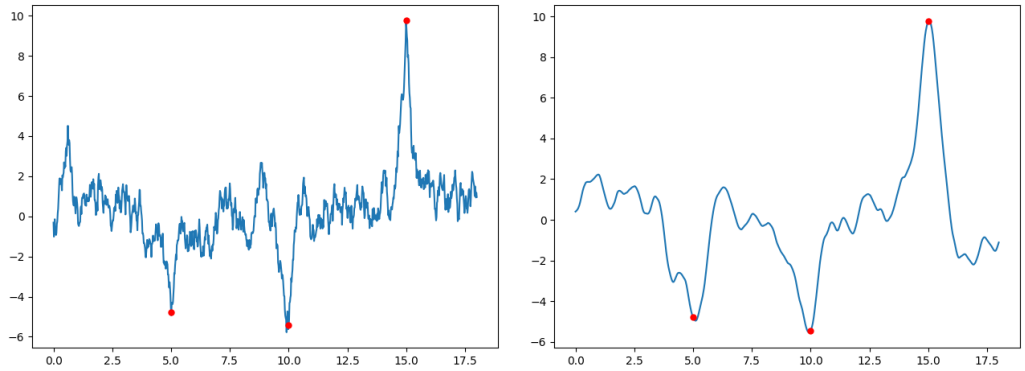
of a scalar. The rationale behind it could be explained by drawing a vertical line intersecting with the Gaussian Process. As an infinite collection of lines, they will intersect at an infinite number of points, the y-values of which will form a normal distribution with known mean - intersection at the middle line of 2.1b - and variance - deducible from the projected length of the shaded area which denotes a 95% confidence interval. It is worth noting that other surrogate models may have different distributions not restricted to normal.

Updating a Gaussian Process means adding more points to it. An updated version of the Gaussian Process in Figure 2.1 could be visualized in Figure 2.2, in which another point is added into the middle of its current 2 points. As can be seen, the shaded area has become thinner, implying that the number of possible target functions has been reduced significantly.

In a nutshell, a Gaussian Process is just a mathematical function which returns a distribution of possible outcomes instead of an exact value. The more data points it has, the more likely it contains the perfect function we seek for in its shaded area.

What type of functions is covered?

As can be seen, albeit the shaded area in Figure 2.1b may contain countless possible functions, it is nowhere wide enough to cover all possible lines between the given pair of points, implying that only a special class of functions is involved by the given Gaussian Process. The component



(a) A rough function (b) A smoother function

Figure 2.3: Examples of function smoothness

responsible for filtering functions is called a *kernel* function [33].

Exactly how the filtering is conducted is a mystery to us even after countless hours of research, but at the very least, we could provide an intuitive example that can give an insight into how it can be done. As can be seen in 2.2a, all the line functions are very *smooth*. In case the term 'smooth' is confusing, let's take a look at a few *rough* (not so smooth) functions in Figure 2.3 for comparison.

Now you have acquired a feeling of smoothness for a function, but mathematically, smoothness is measured by to what extent a function can be differentiated. The reason why functions 2.2a are so smooth is because its kernel function is the radial basis function (RBF) kernel whose analytic form is represented in (2.6) where x and x' are some multidimensional vectors of the same number of dimensions.

$$k(x, x') = -\exp \frac{|x - x'|^2}{2\sigma^2} \tag{2.6}$$

As you may know, a function of type e^x is infinitely differentiable and hence its representative lines are extremely smooth. By mutating the kernel function with others, such as the Matern kernel function, the lines can be very rough as in Figure 2.3a. The best reference for Gaussian Process could be found in [23] with detailed mathematical analysis.

An unintuitive focus

At this point, we have already acquired the intuition over how a Gaussian Process can be used to manage over a set of infinite number of possible target functions. This intuition is the main outcome of our research acquired through countless readings, but nonetheless, it does not give an insight into how a Gaussian Process can be implemented. The role of this subsection is to provide more information in that regard. Rewriting the kernel equation (2.6) with a change in notation for σ as follows

$$k(x, x') = -\exp \frac{|x - x'|^2}{2\theta^2} \tag{2.7}$$

As can be seen, θ is an internal parameter of the kernel. Since the kernel controls which functions to be included, θ must be chosen accordingly to guarantee that the Gaussian Process is consisted of only good target functions. How can θ be chosen? That is where the Bayes' theorem comes into play. Bayes' theorem provides a method to compute the probability that θ is best given a set of data (X, y). ($X = x_1, x_2 \dots x_n$ is a $n \times m$ matrix, x_i is a horizontal input vector and y is a vertical vector of outputs, one for each input vector)

$$P(\theta|y, X) = \frac{P(y|X, \theta)P(y)}{P(y|X)} \tag{2.8}$$

For a constant set of data (X, y), $P(y)$ and $P(y|X)$ should remain constant. Hence it implies that

$$P(\theta|y, X) \propto P(y|X, \theta) \tag{2.9}$$

In other words, when the *marginal likelihood* $P(y|X, \theta)$ is maximum, θ would be the best. The task of searching for a good θ is equivalent to a maximization problem on the likelihood function. Instead of computing $P(y|X, \theta)$ directly, an alternative version called Log Likelihood function $\log P(y|X, \theta)$ is preferred since its analytic form has been known as in (2.10).

$$\log P(y|X, \theta) = -\frac{1}{2} y^T K^{-1} y - \frac{n}{2} \log |K| - \frac{n}{2} \log 2\pi \tag{2.10}$$

where K is a covariance matrix computed by the kernel function [$K_{ij} = k(x_i, x_j)$] and $|K|$ is the matrix determinant. After searching for the best θ, the Gaussian Process is ready to make evaluation at some location x' as follows

$$\mu = K'K^{-1}y + E(y) \tag{2.11}$$

and

$$\sigma^2 = k(x', x') - K'K^{-1}(K')^T \tag{2.12}$$

where K' is a horizontal vector computed by the kernel function [$K'_i = k(x', x_i)$].

3.4. Upper Confidence Bound

To sum up from the previous subsections, a probabilistic surrogate does not directly estimate our confidence for a given set of hyperparameters x, but instead it gives us a distribution of possible outcomes. Within the scope of this thesis, Gaussian Process is our only model of consideration, and hence its resulting distribution is a Gaussian (normal) distribution $\mathcal{N}(\mu, \sigma^2)$. How can such a distribution be summarized into a single number for proper evaluation?

How about $\mu + \sigma$ for starter? It is a single value and can be used for comparison, so there should not be any problem with it. In fact, it is not only valid, but it has been used extensively as a famous acquisition function, namely Upper Confidence Bound. The full form of UCB for Bayesian Optimization is described in (2.13)

$$\mathcal{A}_i(\mu, \sigma) = \mu_{i-1} + \beta_i^{1/2}\sigma_{i-1} \tag{2.13}$$

where i denotes the current iteration (as Bayesian Optimization is an interactive method as mentioned in Algorithm 1), μ_{i-1} and σ_{i-1} represents the distribution from the surrogate updated from the last model, and β_i is a confidence band of the current iteration. In practice, it is possible to always set $\beta_i = 1$ or just some constant.

Why does it even work? The answer is mysterious to us even after working with this magical formula for years before this thesis. The only insight we got is that it is also the same policy that enables the famous

Monte Carlo Tree Search (MCTS) algorithm [34] that is extremely successful in Artificial Intelligence and Games. The fundamental idea is that we need to balance between exploitation (exploiting the current best state) and exploration (exploring other not so good states) in our search for an optimal value. In this case, the variance stands for exploration (high variance means wider shaded area and more possible target functions), the mean represents the exploitation (the general goodness of all functions contained in the Gaussian Process at some point), and the confidence band β_i is a balancing factor. When all terms are filled accordingly, the magic happens and everything just works magically. Avid readers who wish to explore this magical formula in the context of Bayesian Optimization can find more information in [35, 36].

What is special about it? It is fast and differentiable. Unlike the black-box function representing the training algorithm whose derivative is widely perceived as non-existent, an acquisition function (or at least UCB) is differentiable. The analytic form is unfortunately unknown, but because it is so lightweight, numerical methods can be applied to compute the empirical derivative using (2.14). In this case, $f(x) = g(x) = \mathcal{A} \circ \mathcal{S}(x)$.

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{2.14}$$

4. Random Search

Random Search is another popular algorithm for hyperparameter optimization which is known for its simplicity and intuitiveness. Unlike the other, Random Search is supposed to be too intuitive to receive a detailed explanation: just searching randomly and filtering for the best. It is, nonetheless, the first algorithm to be implemented and we would like to outline the structure of Random Search for maximization in Algorithm 2.

5. LightGBM

5.1. Gradient Boosting

What is booting? Boosting is a supervised learning technique inspired from the Wisdom of the Crowd phenomenon, in which the collective opinion of a group of people is said to be equivalent to that of an expert within

Algorithm 2 Random Search

$x^* \leftarrow null, y^* \leftarrow -\infty$
for $i \in \{1, 2 \dots N\}$ **do**
  Generate a random x
  $y \leftarrow f(x)$
  if $y^* < y$ **then**
    $x^* \leftarrow x$
    $y^* \leftarrow y$
  end if
end for
Output (x, y)

that group. Boosting aims to develop a strong predictive model from multiple weak learners, typically Decision Trees. Equation (2.15) gives an example of such a model, where the final answer is accumulated over all weak models $\{f_1, f_2 \dots f_n\}$, each with its own weight c_i ($\sum_i^n c_i = 1$).

$$f(x) = \sum_{i=1}^{n} c_i \times f_i(x) \tag{2.15}$$

What is a weak learner? A weak learner f_i is a normal machine learning model, which does not possess sufficient training resources and hence are not much more capable than random guessing. An example could be a decision tree with a very limited number of leaves or a linear regression polynomial of low degree. Individually it is of no practical value, but gradient booting can help a large number of them evolve into models with state-of-the-art performance such as XGBoost or LightGBM.

What is gradient booting? Gradient booting is a booting algorithm that combines the idea of Gradient Descent [14] and booting. The idea is to build an additive model to which weak learners are added interactively to minimize a differentiable loss function. When a new learner is added, previous learners are frozen and remain unchanged (stage-wise). Unlike the traditional step-wise model AdaBoost [37], gradient booting can be used with any differentiable loss function and has applications expanding far beyond binary classification problems, including regression, multi-class classification and much more. A detailed guide to gradient boosting can be found in [38].

5.2. LightGBM - What?

LightGBM is a gradient boosting framework crafted by Microsoft, with multiple novel enhancements compared to other competitors such as XGBoost [39] or CatBoost [40]. In simple terms, LightGBM's algorithm can provide comparatively accuracy within a much tighter budget on time and memory in comparison to other frameworks given the same hardware condition. The supreme performance is not without some compromises on accuracy, but the different is insignificant over its advantages, and it has been known as the winning solutions in a variety of machine learning competitions.

5.3. LightGBM - Why?

The focus of this thesis leans more towards hyperparameter optimization and it could have been any other algorithms such as linear regression [41] or even deep learning [42], since all of them would be treated as a black-box function without much consideration of their implementation. Nevertheless, LightGBM stands out as a promising candidate for the following advantages:

- **Fast & Light**. As a student's individual project, our computational resources are incapable of supporting heavy-weight models such as neural networks [43] or support vector machines [44]. That restricts our available options to a very limited pool of candidates, one of which is LightGBM - a framework known for being extremely fast and light on memory.
- **Accuracy**. Gradient boosting has been known to provides state-of-the-art performance, which is demonstrable through numerous competitions in machine learning and data science. In addition, it is known for its tolerance on various types of data [45] compared to others. Since our focus is only hyperparameter optimization, other types of optimization are ignored despite being no less important. Choosing a gradient boosting framework like LightGBM saves us the trouble of data engineering even just a bit.
- **No Kotlin API**. Since we aim to make something original, something that has not existed in Kotlin - the language of our choice - would be best. Compared to LightGBM, XGBoost provides slightly better accuracy, but unfortunately, it has its own functional binding in Java already - that is ready to use in Kotlin - and hence is of less preference to us.

Those reasons leave us with a single choice for LightGBM - the fastest gradient boosting framework in existence and most suitable for our purpose.

6. L-BFGS-B

L-BFGS-B is a limited-memory and bound-constrained version of the popular Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) for solving unconstrained nonlinear optimization problem. The bounded optimization problem can be formally defined as searching for the optimal local x^* that minimizes (or maximizes but let us focus on minimization for now without loss of generality) some equation $f(x)$ as mathematically illustrated by (2.16).

$$x^* = \underset{x \in \mathcal{X}}{\operatorname{argmin}} f(x) \tag{2.16}$$

where x belongs to some bounded domain $l \leq x \leq r$.

In this project, L-BFGS-B is used as a black-box algorithm and hence its internal details will not be reported in this documentation. Instead curious readers are welcome to explore the full technical structure of the algorithm in [13]. However, it is still critical to understand at least its basic functionality and how it could be used in this project.

What does it do? For a minimization problem, L-BFGS-B algorithm will find the value x that minimizes a black box function $f(x)$, but different from your general expectation, the algorithm is only capable of searching for a *local optimum*. An illustration could be visualized in Figure 2.4. As can be seen, B is one such optimum, but it is only the best in its neighborhood. The true (or global) optimum for the region bounded by line l and r is D where the function is minimal. L-BFGS-B is a gradient-based algorithm which means it does so by analyzing the gradient of the target function as shown in (2.14).

(2.14) shows that if $f'(x) < 0$, that means $f(x + \epsilon) < f(x)$ and so by going towards that direction to $x + \epsilon$, we reach a point lower than it was at x. Repeating the same process until $f'(x) = 0$, we will meet a local optimum. Verifying if it is a local minimum or maximum can be easily completed by analyzing its second-order gradient $f''(x)$ (the Hessian [46]).

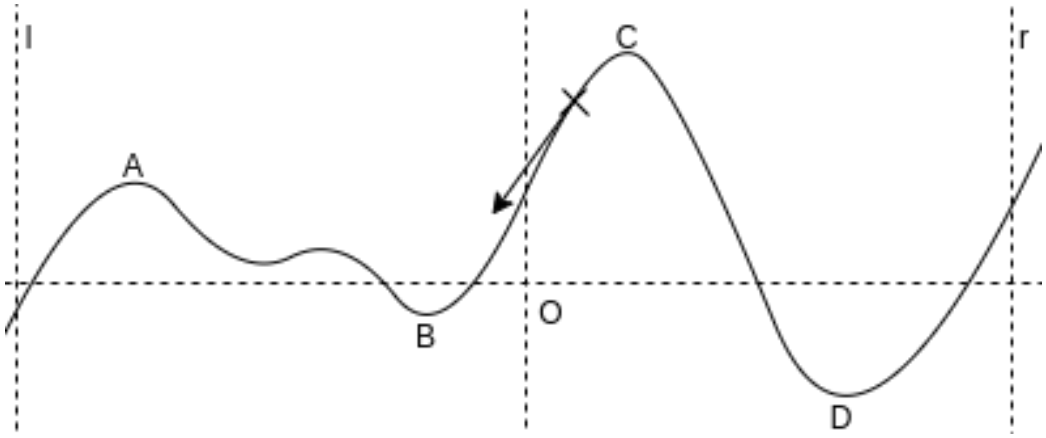


Figure 2.4: A sample view of local optimima

How can it benefit our project? As can be seen in Algorithm 1, finding an optimum for a maximization problem is one of the necessary requirements to implement Bayesian Optimization and this method allows us to do it by translating a maximization problem to a minimization problem (default by the L-BFGS-B framework created by the authors). Even if it can only find local optima, it can be tweaked by, for example, executing multiple times at random initial locations, to find the (nearly) global optimum. That is the algorithm employed in our code for its simplicity and intuitiveness.

7. Kotlin for Data Science

7.1. Why not Python?

Python is very flexible. Programming in Python permits data scientists to quickly prototype their ideas without worrying about strict software standards like those established in many other programming languages. Most imperatively, Python has an active community with a vast selection of high-quality libraries providing out-of-the-box support for data science. Python is simply the best, the de facto and the only practical choice for data scientists of this era. So why not Python?

Python was indeed the first language of our choice for the initial period of the project. Work was done, progress was made and the development was initially up to expectation. But the deeper we dived, the harder it was to move further due to the lack of static typing and hard software

requirements in Python. As the project increased in size, our beloved flexibility revealed its evil side and we realized that Python is not the best candidate for the job. Why?

Python code is hard to follow. The lack of static typing complicates the meaning of Python API. For example, Listing 2.1 is a sample Python API which provides a method for fitting a machine learning model called *fit* with parameter *X* and *y* and the learning rate *alpha*. What kind of input should we provide for them? Should we provide a nested list or a Numpy matrix for *X*? Should we provide a list or a Numpy array for *y*? Is *alpha* a number or some convention such as a string called *'auto'*? Or are there some self-defined classes of objects specifically designed for them? Working with Python is initially smooth and simple, but as the project gets larger, developers will slowly lose track of their own classes and methods, unless they are documented with utmost consideration, which is rare among Python developers even in some of the most famous projects.

```
class Model:
    def fit(X, y, alpha = None):
        pass
```

Listing 2.1: Python sample API

Lack of a good IDE. Python has a great degree of support from its community and numerous commercial companies, but a good IDE (Integrated Development Environment) for Python has yet been found, partly due to its flexibility and lack of static typing. If human developers find Python API hard to comprehend, no machine could, and hence a variety of coding features such as code intellisense or code generation would suffer greatly. Even PyCharm - the Python IDE for Professional Developers [47] - is far from perfect in comparison to its counterpart for Java - IntelliJ IDEA [48].

Python is really slow. As Python is an interpreted language, it has been known to be terrible on performance, especially on basic controls such as for- and while-loop. Various projects such as Cython, Numba and multiple C/C++ binding libraries have been born out of the need to compensate for Python's lack of speed, which essentially destroys its elegant syntax and replaces it with some half-C half-Python substitute. Porting a C/C++ implementation to Python is basically harder than just doing everything in C/C++. Hence why should we even use Python?

Python has everything. Yes, this is its strongest upside, but at the same time, its most negative aspect from the perspective of this thesis. Our aim is to study the most popular algorithms for hyperparameter optimization and the best way to understand them is obviously to implement them, but there have been dozens of libraries in Python offering almost the same algorithms with decent quality. Why should another be implemented? It could, of course, be done, but the life of such a project would hardly last beyond the scope a school project. Computer science is an always changing field. An ambitious computer scientist would always look for a possibility to make an original contribution to the world and Python offers little hope in that regard.

Regardless, Python is undoubtedly the king of data science and was the only choice available for our project until the announcement of Kotlin support for Data Science in December 2019 when we made the most difficult decision ever: restarting everything from zero for Kotlin.

7.2. Kotlin - What and Why?

What is Kotlin?

Kotlin [49] is a newcomer in the world of programming languages, created by JetBrains. Aimed to become a better Java, it is equipped with fantastic language features that live up to its expectation.

Kotlin Features

First is its 100% interoperability with Java that allows all available tools and packages of Java - the most popular programming language in existence - accessible from Kotlin. Learning Kotlin is hence much easier for all developers since most would know Java or be familiar with at least a C-style programming language.

Second is its sleek, elegant and modern syntax which could pose a serious challenge to Python in all aspects. Albeit being a compiled language implies certain restrictions, the trade-off is worthwhile given its support for static typing and good software standards that are missing from Python.

In addition, Kotlin is fast, comparably as fast as Java since they are both compiled into byte code and executed by the same JVM. Coding in Kotlin is hence easier compared to Python since it is unnecessary to worry about avoiding basic controls such as loops for performance. Porting a C/C++

library to Kotlin is the same as from C/C++ to Java, which is not harder than to Python and could even be ignored thanks to continuous improvement on JVM's performance.

Last, but not least, is its creator - JetBrains. Crafted by the creator of the undisputed number one Java IDE - IntelliJ IDEA - implies that Kotlin will enjoy excellent first-class tooling support in the future. While Python's growth is dependent on open-source community, backed by a commercial company gives Kotlin more potential and job security to evolve. That has already been the case! For example, even Google accepted Kotlin as the first-class citizen for Android in 2017, just shortly after its first stable release. Gradle - one of the most popular software build tool for Java - announced its new build language in Kotlin. Kotlin has evolved beyond its original platforms - JVM and mobile - and now can initially target native, web and most importantly, data science.

Support for Data Science

Support for data science in Kotlin has been initiated since 2016, but only until recently - December 2019, has data science been listed as an official target on Kotlin's homepage [50]. Various projects have been started with the most important ones being Kotlin-Numpy and Kotlin kernel for Jupyter. The former is expected to be the center of all future machine learning and data science libraries in Kotlin, as was Numpy in Python, whilst for the latter, it allows the interactive use of Kotlin on the world's most popular tool for scientific computing - Jupyter [51].

Unfortunately, even if official support has been announced, it is still a plan for the near future as most important libraries are far from being production-ready. Regardless, Kotlin shreds a new hope for data science on JVM. We strongly believe that if our project is implemented in Kotlin, it will last beyond the life of a school project, and has the potential to hold an important position in the world of machine learning and data science on JVM as the first library of its kind. That is why Kotlin was chosen for our project despite the upcoming challenges it might bring.

Chapter III

Software Product

1. Architecture

1.1. Overview

This section presents an overview of the available components implemented in our framework. The project could be analyzed from 3 different aspects: a) software engineering, b) research and c) experiment. Inside each is a set of components implemented for the same purpose of their group. Last, but not least, our software is accompanied with a set of various datasets for user experiment. A complete overview of our product is illustrated in Figure 3.1 and each component will be explicitly discussed in subsequent subsections.

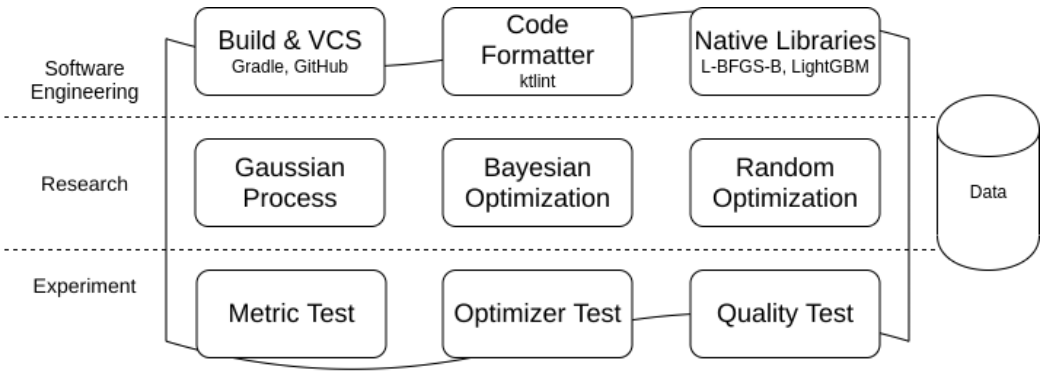


Figure 3.1: Software Architecture Overview

1.2. Software Engineering

This area covers the technical work needed to facilitate the implementation of all other components in our project. Unlike Python notebooks, a scalable software library needs to be well-structured from the beginning as they cannot be simply executed on the fly. Components in this area are responsible for:

- Defining the standard code quality.
- Backing up important code milestones.
- Managing all external dependencies required by every component in further sections.
- Enabling continuous integration and deploying the final product in a way that other external projects can use.

To achieve these goals, the following components are set up from the beginning of the project: a build and version control system (VCS), an automated code formatter, and native libraries.

Build & VCS

As you may know, a program written in a high-level programming language, from Python to Kotlin, or even the fast C/C++, must be compiled into machine code and only then are they executable by a computer. A build system is responsible for automating this task for a large number of files in a large-scale project where manually compiling every one of them seems daunting and impossible. In our project, the build tool of our choice is Gradle [52] - a popular Java build tool and hence can be usable from Kotlin. Gradle offers the following functionalities:

- Building all Kotlin files & automating tests.
- Downloading and linking external libraries.
- Packing and publishing the resulting artifact for external use.

However, building can be an expensive task which requires a lot of computational resources on our local machine, hence cannot be done regularly. To solve this issue, an idea is to migrate it to a remote machine and that is where the role of GitHub comes into play. To the mass of developers, GitHub [53] is nothing more than a free version control system, but from our point of view it could be used as a precious free virtual machine for testing our code. The idea is to load the code to the remote GitHub

Figure 3.2: GitHub Actions

repository and delegate most of our testing and building on the machine hosted by GitHub with GitHub Actions. Figure 3.2 shows an overview of how GitHub Actions has saved us a huge amount of computational time with every build costing from 5 to 12 minutes. In detail, GitHub helps us with the following functionalities:

- Free back-up for our code history in case we would like to start over at a certain point in time.
- Free virtual computer for conducting our continuous integration and testing.

Code Formatter

In real-world projects where hundreds of people work on the same product, a common agreement in code styles is important, and that is where a code formatter comes in handy as it will perform code refactoring automatically. In this thesis project where I am the sole developer, a code formatter is still of paramount importance in that it could prepare my code for the future public availability and encourage more contributors. That is why we employed one such formatter in our project: *ktlint - an anti-bikeshedding Kotlin linter with built-in formatter* [54] that is used for guaranteeing a publicly acceptable code quality.

Native Libraries

What are native libraries? Numerous algorithms have been written not in Java, Kotlin or even Python, but in C/C++ and even Fortran to be embedded in higher-level programming languages for performance. They

include LightGBM and L-BFGS-B - the algorithms of our choice for this project. Since Kotlin cannot just use these native libraries directly, there are steps that must be followed to enable them for use within Kotlin:

- **Step 1**. Build the given library into shared objects (files ending in .so) in Linux or its corresponding type in other operating systems.
- **Step 2**. Use SWIG or some similar software to build its low-level (with control over pointers and memory management) API for use within Java (Java API is usable within Kotlin).
- **Step 3**. Pack the shared objects and low-level Java API into a single Java Archive (JAR) for easy distribution.
- **Step 4**. Build a high-level Kotlin API (getting rid of pointers and memory management) from the low-level Java API.

LightGBM is lucky to be born by Microsoft and steps 1-3 have been completed by its father. Step 4 was the only step left for us to create the first ever Kotlin API for LightGBM. In our library, LightGBM provides the following 2 static methods training and cross-validation.

```
fun fit(                        // from class Booster
    params: Map<String, Any>, // hyperparameters input
    data: Matrix<Double>,     // training data
    label: DoubleArray,       // training output
    rounds: Int               // number of training iterations
): Booster                    // returning fitted model

fun cv(                         // from class Booster
    metric: Metric,           // performance metric
    params: Map<String, Any>, // hyperparameters input
    data: Matrix<Double>,     // training data
    label: DoubleArray,       // training output
    maxiter: Int,             // number of training iterations
    nFolds: Int               // k in k-fold cross validation
): DoubleArray                // returning k scores


/**
 * Making predictions for multi- or single input or save
 * These are Booster internal methods
 */
fun predict(
```

```
        data: Matrix<Double>       // multi-input as matrix
    ): DoubleArray                 // multi-output
    fun predict(
        x: DoubleArray             // single vector input
    ): Double                      // single output
    fun save(filePath: String)

    // Please always call this function after all predictions
    fun close()
```

Listing 3.1: LightGBM API

However, the progress was not so smooth with L-BFGS-B, as the original authors have written it in Fortran and that is where the trouble starts. After countless hours of random experiments and searching for alternatives, we finally found a usable solution. Mateusz Kobos provided a C wrapper for the original Fortran code and has completed most of steps 1-2 [55]. Before using his wrapper, we have modernized the Fortran code, disabled noisy logs, optimized basic operations in the C wrapper, restructured the low-level Java API, and enabled another layer of compiler optimization (-Ofast instead of -O3) for performance. After that we applied steps 3-4 and the resulting product is a fast, modern and ready-to-use Kotlin API for L-BFGS-B. L-BFGS-B can be used from KotlinML by invoking the following 2 static methods

```
    fun minimize(                   // from class LBFGSBWrapper
        func: DifferentialFunction, // function to optimize
        xZero: DoubleArray,         // initial guess
        bounds: Array<Bound>,       // constraints
        maxIterations: Int = 15000  // number of iterations
    ): Summary                      // just output summary

    fun maximize(                   // from class NumericOptimizer
        func: DifferentialFunction, // function to optimize
        xZero: DoubleArray,         // initial guess
        bounds: Array<Bound>,       // constraints
        maxiter: Int = 15000,       // number of iterations
        type: OptimizerType = OptimizerType.L_BFGS_B // keep this
    )
```

Listing 3.2: L-BFGS-B API

1.3. Research

In this area, our focus is about understanding the algorithms and translating them into the language of a computer. For all algorithms introduced in this section, we had no previous knowledge of them and hence had to research from scratch and have already presented every concept we learned in chapter II - Preliminaries. Therefore, the rest of this subsection is concentrated on introducing the API for each algorithm. This section includes 3 algorithms: Random Search, Gaussian Process and Bayesian Optimization.

Random Search and Bayesian Optimization

Both Random Search and Bayesian Optimization inherit a common interface, namely Optimizer, implying that they will offer the same way of invocation. To use either of them, just invoke the following method specified by the Optimizer interface

```
interface Optimizer {
    fun argmax(
        func: (Map<String, Any>) -> Double,// function evaluating
                                           // hyperparameters
        xSpace: XSpace,                    // parameter Domain
        maxiter: Int                       // number of iterations
    ): Pair<Map<String, Any>, Double>
}
```

Listing 3.3: Optimizer API

Gaussian Process

A Gaussian process is not only a companion to Bayesian Optimization, but can also be used as an independent machine learning model. The use of Gaussian Process in KotlinML is enabled by the following 2 methods

```
    fun fit(                           // from class GPRegressor
        data: Matrix<Double>,          // training data
        y: Matrix<Double>,             // training output
        maxiter: Int = 1,              // number of iterations
        kernel: Kernel = RBF(),        // kernel function
```

```
        noise: Double = 1e-10,        // y noise
        normalizeY: Boolean = false   // false means mean is 0
    ): GPRegressor                    // trained Gaussian Process

    fun predict(                      // internal of GPRegressor
        x: DoubleArray                // vector input
    ): GPPrediction                   // (mean, variance)
```

Listing 3.4: Gaussian Process Regression API

1.4. Experiment

In addition to researching about Bayesian Optimization, we also conduct various experiments to explore other aspects of Machine Learning and Data Science. The structure and results of these experiments will be discussed in chapter **Research & Experiments**. Here is a summary of what experiments we have conducted as a companion to our software:

1. **Metric Correlation** This experiment evaluates the efficiency of the currently most popular hyperprameter metric.
2. **No Free Lunch Theorem**. This experiment verifies the truth about No Free Lunch Theorem in the context of Machine Learning and Data Science.
3. **Are optimizers always good**? This experiment confirms the effectiveness of our optimizers.
4. **UCB and Hyperparameter Metric**. This experiment attempts to apply the power of Upper-Confidence Bound algorithm for the current hyperparameter metric.

1.5. Data

This project also includes multiple datasets for user testing. They cannot, however, be packed into the library artifact and must be copied accordingly. A summary of provided datasets is shown in Table 3.1. They were collected from various sources, including [56, 57]. All of them are for binary classification with the target being the first column.

Table 3.1: Datasets by Size

Dataset	Train Set	Test Set
nba_logreg	308	460
pima_indians_diabetes	537	803
imblearn_yeast_me2	594	890
wine_quality	1960	2938
imblearn_abalone	1671	2506

2. User Guide

2.1. System Requirements

To use the library, please upgrade your system accordingly to ensure that following requirements are satisfied:

- Fortran Dependencies (libgfortran-9-dev) (absolute)
- Java 11 - GraalVM Enterprise 19.3.0.2 (recommended)
- IntelliJ IDEA Ultimate 2019.3 (highly recommended)
- Ubuntu 19.10 (highly recommended)
- Gradle (optional)

Albeit there exist alternatives such as OpenJDK 11 for GraalVM 11, Eclipse for IDEA, Windows or MacOS instead of Ubuntu 19.10 and so on, we strongly recommend the users to upgrade their system to the specified requirements. All those dependencies are free of charge (with your education mail account) and have been tested thoroughly by the creator of this project. Links for installing the recommended dependencies are provided in [58, 59, 60]. In addition, it will always be useful if your computer is around 2 year old or less - manufactured in 2018 - and has a minimum of 16 gigabytes of random access memory (RAM) and operates on a fast solid-state drive (SSD).

Relaxing the hardware and software requirements is our aim for the future when the project has gotten more attention and contribution from the public. However, please bear with us for the time being since our budget as a student's project is very limited.

Last, but not least, for installing Fortran dependencies on Ubuntu 19.10 - a very important dependency, please enter the following command in

your favourite terminal as a super user

```
>>  sudo apt install libgfortran-9-dev
```

Listing 3.5: Installing Fortran libraries on Ubuntu

2.2. Installation

In the future, this project will be publicly available via popular repositories such as JCenter or Maven Central and the this step can be conveniently skipped. However, until the general availability, it is necessary to build and deploy the software for local use. Please go to the project folder (**kotlinml** or **KotlinML**) and open it in your favorite terminal.

```
>>  ./gradlew clean build -x test
>>  ./gradlew publishToMavenLocal
```

Listing 3.6: Building KotlinML for local use

After that, open your favourite IDE and create an empty Kotlin project with Maven or Gradle. Using Maven, we need to add the corresponding dependency for KotlinML to the pom.xml file as follows

```
<dependencies>
    <dependency>
        <groupId>thuan.handsome</groupId>
        <artifactId>kotlinml</artifactId>
        <version>0.1</version>
    </dependency>
</dependencies>
```

Listing 3.7: Maven dependency for KotlinML

For Gradle lovers, it is even simpler by adding the following few lines to your build.gradle configuration file.

```
dependencies {
    implementation 'thuan.handsome:kotlinml:0.1'
}
```

Listing 3.8: Gradle dependency for KotlinML

After that, the software is right at your service. Welcome to KotlinML!

2.3. Features

- Provides a Kotlin API for basic classification and regression with LightGBM.
- Provides a Kotlin API for local optimization with the gradient-based L-BGFS-B algorithm.
- Provides a uniform Random Optimizer in Kotlin for optimizing any black-box function.
- Provides a Bayesian Optimizer in Kotlin for optimizing any black-box function.
- Provides a basic implementation of Gaussian Process Regression with 2 popular kernels - Matern and RBF - for supporting Bayesian Implementation.
- Provides basic metrics such as F1 Score and Accuracy for verifying model performance.
- Provides a cross-validation method for general use.
- Provides various tests and examples for ensuring quality and assisting users to get familiar.
- Provides datasets of small and medium sizes for testing and experimenting.

2.4. Examples

Optimizing a mathematical function

In this example, our goal is to minimize a simple mathematical function $f(x, y, z, w) = (x-1)^2 + (y-2)^2 + (z-3)^2 + (w-4)^2$ assuming that $-100 \leq x, y, z, w \leq 100$. It would be obvious that the optimal solution is $f(1, 2, 3, 4) = 0$. How can we perform this task using KotlinML?

Firstly, the parameters' domain has to be explicitly defined as shown in Listing 3.9. Intuitive enough, for each parameter, we need to provide its name, lower bound and upper bound.

```
val xSpace = UniformXSpace()
xSpace.addParam("x", -100.0, 100.0)
xSpace.addParam("y", -100.0, 100.0)
xSpace.addParam("z", -100.0, 100.0)
xSpace.addParam("w", -100.0, 100.0)
```

Listing 3.9: Defining parameters' domain in KotlinML

After that the function needs to be translated from its mathematical form to Kotlin. Listing 3.10 gives an example how it can be done. Firstly, the parameters are extracted from the *params* map, then we convert them to their proper type, and compute the result. Since all optimizers in KotlinML only support maximization problem for the time being, the result is negated to convert it to a minimization problem.

```
val func = fun(params: Map<String, Any>): Double {
    val x = params["x"] as Double
    val y = params["y"] as Double
    val z = params["z"] as Double
    val w = params["w"] as Double
    val res = ((x - 1).pow(2)
        + (y - 2).pow(2)
        + (z - 3).pow(2)
        + (w - 4).pow(2))
    return -res
```

Listing 3.10: A sample math function in Kotlin

After that, picking an optimizer of your choice, we can find the (nearly) optimal solution as follows

```
val optimizer = BayesianOptimizer(kernel = RBF()) {
val (x, y) = optimizer.argmax(func,
    xSpace = xSpace,
    maxiter = 50
)
```

Listing 3.11: Running Bayesian Optimizer

After executing Listing 3.11, we acquired the following results: $x = 1.3962461549129601$, $y = 1.7591653979718713$, $z = 2.2328101524681827$,

$w = 3.5988294061709527$ and $f(x, y, z, w) = -0.9645304283263962$, which is pretty close. Since this is a black-box optimization problem, perfect accuracy would be ideal but should not be expected. For users with modern hardware, it is possible to improve the accuracy by increasing the *max_iter* parameter. It should be, nonetheless, used with caution since the time complexity of Bayesian Optimization is no less than $O(n^4)$ with n being the number of iterations.

Binary classification with LightGBM

Since LightGBM is another pillar of this thesis, an example with LightGBM must be provided. In this part, we are going to introduce how to perform a simple binary classification task with LightGBM using KotlinML.

Machine learning is learning from data, so first of all, we need to load the datasets for training and testing, which could be done using the convenient method called *getXY* provided by KotlinML as follows

```
val (trainData, trainLabel) = getXY(
    csvMatrixPath = "data/imblearn_abalone_train.csv",
    labelColumnIndex = 0
)
val (testData, testLabel) = getXY(
    csvMatrixPath = "data/imblearn_abalone_test.csv",
    labelColumnIndex = 0
)
```

Listing 3.12: Loading datasets for training and testing

Method *getXY* inputs a path to the data in CSV format and the index of the target column (0-based). The data must be a matrix with neither header nor index where the delimiter is a comma. After the datasets have been loaded, the training can begin as follows

```
val booster = Booster.fit(
    params = mapOf(
        "objective" to "binary",
        "verbose" to -1
    ),
    data = trainData,
    label = trainLabel,
```

```
        rounds = 100
    )
```

Listing 3.13: Training a LightGBM model

booster is our trained model. As the model has been ready, predictions or scoring can be made as follows

```
    val metric = F1Score()

    val trainPredictions = booster.predict(trainData)
    val trainScore = metric.evaluate(trainLabel, trainPredictions)

    val testPredictions = booster.predict(testData)
    val testScore = metric.evaluate(testLabel, testPredictions)
```

Listing 3.14: Evaluating a given model

Last, but not least, *booster* must always be closed after use, unless your computer is super capable of handling memory crash.

```
    booster.close()
```

Listing 3.15: Closing a model

Hyperparameter Optimization

As you can guess, all we need to perform hyperparameter optimization is to combine the above 2 examples. By boxing the training and testing processes into a black-box function, hyperparameter optimization is exactly the same as optimizing a mathematical function. Listing 3.16 gives an overview of how such a black-box function would look like.

```
    val func = fun(params: Map<String, Any>): Double {
        val scores = Booster.cv(metric, params,
            data = trainData,
            label = trainLabel,
            maxiter = 30,
            nFolds = 5
        )
```

```
        return scores.mean()
    }
```

Listing 3.16: A sample black-box function for HPO

After that, using an optimizer of your choice, the best hyperparameter set *params* that maximizes our black-box function can be found as follows

```
    val (params, _) = optimizer.argmax(
        func, xSpace,
        maxiter = 32
    )
```

Listing 3.17: Optimizing the HPO black-box function

Now the best set of hyperparameters has been found and can be used for training a LightGBM model

```
    val booster = Booster.fit(params, trainData, trainLabel, 30)
```

Listing 3.18: Training LightGBM with given hyperparameters

It is worth noting that the **xSpace** domain has to be redefined accordingly for LightGBM. An example is given as follows

```
    val xSpace = UniformXSpace().apply {
        addConstantParams(
            mapOf(
                "objective" to "binary",
                "is_unbalance" to true,
                "verbose" to -1
            )
        )

        addParam("min_child_weight", 10.0, 25.0, XType.INT)
        addParam("num_leaves", 63.0, 127.0, XType.INT)
        addParam("max_depth", 10.0, 25.0, XType.INT)
        addParam("min_split_gain", 1e-3, 1e-1)
        addParam("learning_rate", 1e-2, 2e-1)
        addParam("subsample", 0.6, 0.999)
```

```
        addParam("lambda_l1", 1e-9, 1.0)
        addParam("lambda_l2", 1e-9, 1.0)
    }
```

Listing 3.19: A sample hyperparameter space for LightGBM

Chapter IV

Research & Experiments

1. Preparation

1.1. Selecting Hyperparameters

Before diving into exciting experiments, it is essential to re-emphasize the basic requirements specified by Table 1.1: This framework is currently concentrated only on binary classification. Further objectives will definitely be considered in the near future, but this condition was chosen for its simplicity and was vital for the timely completion of this project. In addition, since this is a hyperparameter optimization framework for LightGBM, a specified set of LightGBM's hyperparameters was also selected and presented in Table 4.1. Together they form a standard domain for all optimizers within the scope of this project. In addition, another hyperparameter called *is_unbalance* will always be set to true for performance reasons unless otherwise noted.

The choice of these hyperparameters and their respective ranges are rather random since they will be treated as parameters of a black-box function without much concerns regarding their meaning and responsibilities. However, during the selection of them, the official guide on parameters tuning for LightGBM [61] has been consulted extensively and may have a certain impact on the final decision.

Table 4.1: LightGBM Selected Hyperparameters

Hyperparameters	**Type**	**Range**
min_child_weight	Integer	$[10, 25]$
num_leaves	Integer	$[2^6 - 1, 2^7 - 1]$
max_depth	Integer	$[10, 25]$
min_split_gain	Double	$[10^{-3}, 10^{-2}]$
learning_rate	Double	$[10^{-3}, 0.2]$
subsample	Double	$[0.6, 0.999]$
lambda_l1	Double	$[10^{-9}, 1.0]$
lambda_l2	Double	$[10^{-9}, 1.0]$

1.2. Searching for a Good Model Metric

What is a model metric? A model metric $\mathcal{M}$ estimates the quality of an already trained model f and summarizes it into a number $\mathcal{M}(f)$ that indicates how well or badly a model would perform on a dataset $\mathcal{D} = \{(x_i, y_i)\}$.

The simplest metric: accuracy. An intuitive metric for classification is the *accuracy metric* which can be formularized as follows

$$\mathcal{M}(f) = \frac{|\{x_i | y_i = f(x_i)\}|}{|\mathcal{D}|} \tag{4.1}$$

where the nominator represents the cardinality of all observations where the trained model makes a correct prediction, while the denominator stands for the total number of observations in the dataset. Their ratio is referred to as the accuracy of a model.

What is wrong with accuracy metric? Albeit accuracy is an intuitive and simple metric, the biggest problem with it is that false classification is not treated with high priority. Let's imagine a new disease will occur tomorrow with infection rate of 95%. A dummy model $f(x) = 0$ (0 stands for being infected, and 1 for being healthy) will have a ground-breaking accuracy of 95% and all humanity will be wiped out if such a model is used (since we do not know who to save anymore). In medical applications or weapon control, misclassification is deadly expensive with the most severe example probably being the 1983 Soviet Nuclear False Alarm Incident which almost triggered the 3^{rd} World War. That is why another

metric which heavily penalizes false classification is required for most applications.

A promising metric: F1 Score. Assuming the class of higher priority is class 1 where false classification is intolerable, the formula for F1 Score is presented in (4.2).

$$\mathcal{M}(f) = \frac{2|\{x_i|1 = y_i = f(x_i)\}|}{2|\{x_i|1 = y_i = f(x_i)\}| + |\{x_i|y_i \neq f(x_i)\}|} \tag{4.2}$$

The idea behind it is rather intuitive. The term $|\{x_i|y_i \neq f(x_i)\}|$ represents the number of misclassified observations, which is taken into account since we care about misclassification as mentioned before. The term $|\{x_i|1 = y_i = f(x_i)\}|$ stands for the number correctly-classified observations of the prioritized class 1. Since class 1 is of paramount important to us, we multiply it by 2 to increase its overall impact in the formula. For the correct classifications of the lower prioritized class 0, they are not of our interests and hence are ignored, giving more weight to the class of our interests.

Other literature would start off by explaining about the confusion matrix and introducing a dozens of concepts, including True Positive, False Positive, True Negative, False Negative, Recall, Precision, Specificity and so on, which is extremely unnecessary and unintuitive from our point of view. From our perspective as computer scientists, it would be more efficient to introduce a raw equation whose terms are explanable in the context of our priority. Nonetheless, math lovers are welcome to explore more about the F1 Score metric in [19] in a more mathematical way. Most of our datasets will be slightly imbalanced and hence F1 Score will serve as the best metric for evaluating our trained model performance for all experiments in this thesis project.

2. Experiments

2.1. Discovery 1: Metric Correlation

Challenge

As mentioned before, the most popular metric $\mathcal{H}$ for scoring hyperparameters is k-fold cross validation mean. The goal of this subsection is to explore how well this metric correlates to the real-world performance of

a model. In other words, is $\mathcal{H}(w) \propto \mathcal{M}(f)$ where $f = \mathcal{T}(w)$ is a model trained using hyperparameter set w ($\mathcal{T}$ is a training algorithm) and $\mathcal{M}$ is a model metric?

Experiment Setup

This task can be accomplished by randomly generating different sets of hyperparameters, evaluating them using the k-fold cross validation mean ($k = 5$ for all our tests) and their respective trained model using F1 Score. The experiment can be outlined in Algorithm 3 $N = 100$ for all test scenarios.

Algorithm 3 Metric Correlation

$X \leftarrow \{\}, Y \leftarrow \{\}$
for $i \in \{1, 2 \dots N\}$ **do**
  Generate a random hyperparameter set w
  $f \leftarrow \mathcal{T}(w)$
  $x \leftarrow \mathcal{H}(w)$, $y \leftarrow \mathcal{M}(f)$
  $X \leftarrow X \cup \{x\}$, $Y \leftarrow Y \cup \{y\}$
end for

After that the correlation ρ of the random variable X and Y are calculated using (4.3).

$$\rho(X, Y) = E(XY) - E(X)E(Y) \tag{4.3}$$

Experimental Results

Table 4.2 summarizes the results of our experiments. At first glance, the mean cross validation metric seems to perform excellently on most of our datasets where the correlation is almost perfect as in dataset imblearn_abalone. The correlation, however, varies greatly across different sets of data, to as low as 0.42 in the nba_logreg dataset, which is not too terrible. However, is that really the case?

After careful consideration, we decided to dive deeper as these values seem too good to be true. The best way to analyze it is via visualization. Figure 4.1 visualizes the correlation of the most perfect dataset: imblearn_abalone.

As can be seen, something is definitel wrong, as the points are not uniformly distributed. There are some noisy points near $(0, 0)$ and $(0.3, 0.32)$

Table 4.2: Metric Correlation

Dataset	**Correlation** $\rho(X, Y)$
nba_logreg	0.4231851663570632
pima_indians_diabetes	0.8987012274961834
imblearn_yeast_me2	0.593337898540218
wine_quality	0.8967283064811504
imblearn_abalone	0.9922448629152825

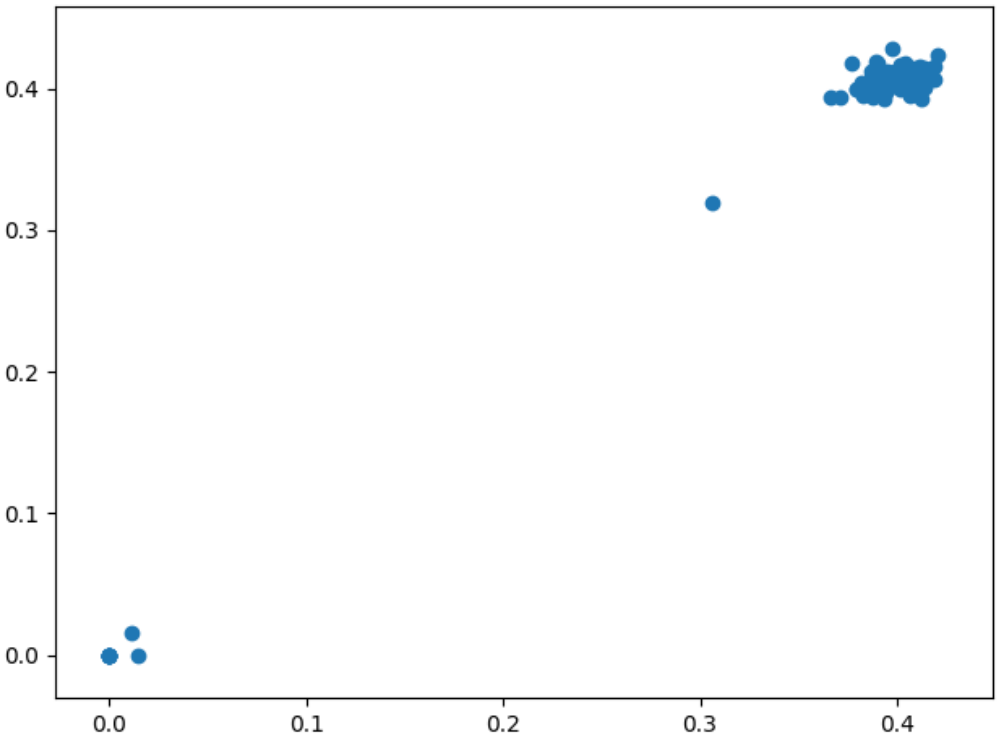


Figure 4.1: Visualization of imblearn_abalone dataset correlation

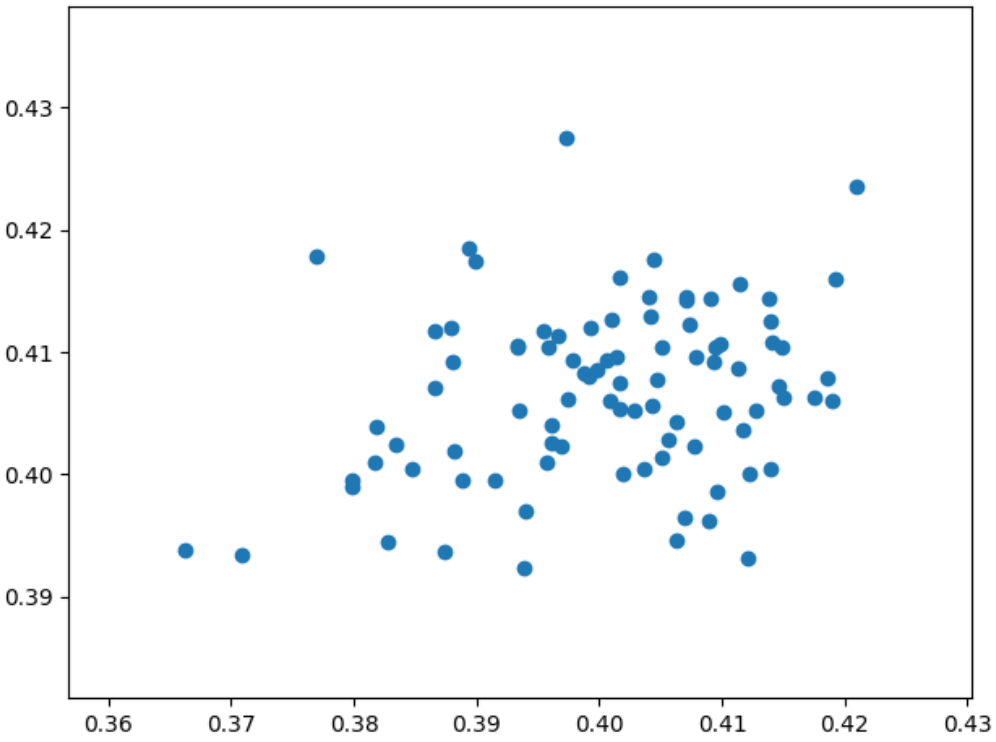


Figure 4.2: Re-visualization of imblearn_abalone dataset correlation

which have resulted in the misconception. After manually filtering the dataset by removing around 10 such points, Figure 4.2 shows a more realistic visualization of the correlation between the hyperparameter metric and real-world performance for the best dataset: imblearn_abalone. At this point, the updated correlation is down to 0.27485912 - an embarrassing result. Is that the same situation for all?

We, unfortunately, cannot afford to manually filter all datasets and hence can only leave that question for future research. Nonetheless, even if mean cross validation has proven far from being a good indicator for real performance, we managed to confirm one of its good characteristics: The graph's tendency is in the right direction. Let us analyze Figure 4.3 for clearer details.

Figure 4.3 visualizes the correlation for the previously weakest dataset: nba_logreg. As can be seen, even if they are not closely correlated, the figure shows that if hyperparameter metric $\mathcal{H}$ improves, it is likely that the model metric $\mathcal{M}$ will also improves. That property should make it

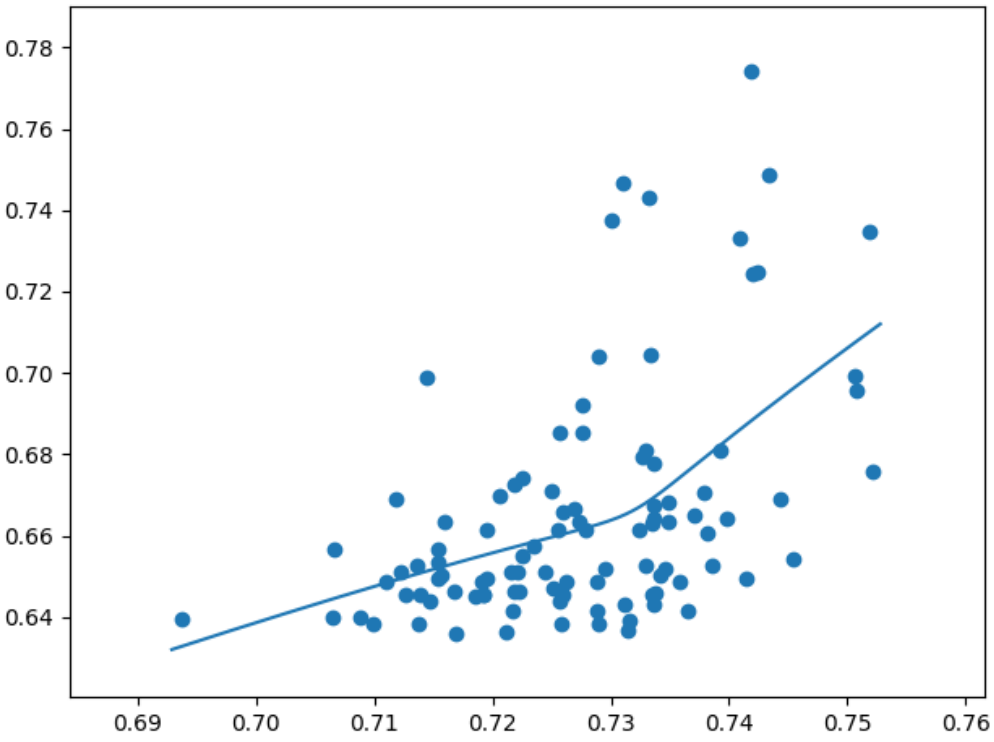


Figure 4.3: Visualization of nba_logreg dataset correlation

sufficient to use this metric as an objective function for hyperparameter optimization.

Experiment Conclusion

Even if mean cross validation metric is far from being a good indicator, it is still a usable metric of optimization for hyperparameter optimization. Nonetheless, a better metric should be preferred if exists considering the very low correlation.

2.2. Discovery 2: No Free Lunch Theorem

Challenge

No Free Lunch Theorem (NFLT) [62] is probably among the best-known impossibility theorems among optimization problem solvers, which states that *any two optimization algorithms are equivalent when their performance is averaged across all possible problems*. In one way, it could be interpreted in the context of hyperparameter optimization that the naive random search is not necessary inferior to the advanced Bayesian Optimization. Refusing to acknowledge this theorem led us to the implementation of this project and it is time to verify its correctness by means of experiments. Will the NFLT hold true for hyperparameter optimization?

Experiment Setup

This experiment will execute the 2 implemented algorithms in this project - Random Search and Bayesian Optimzation - on the same set of hyperparameters specified by Table 4.1 on all available datasets. After that their results will be compared and analyzed. The implementation of this experiment will follow Algorithm 4.

Experimental Results

The results of our experiment have been summarized in Table 4.3 which showcases the most unanticipated outcome: They are comparatively equivalent. As can be seen, except for the first dataset - nba_logreg - where Bayesian Optimization is clearly the victor, the remaining datasets barely show a difference. In other words, the NFLT is perfectly accurate for hyperparameter optimization, which could somehow explain the general trend to use naive methods such as Random Search. The reason is simple, they are perfectly equivalent.

Algorithm 4 No Free Lunch Theorem Experiment

for every dataset D **do**
  $Y \leftarrow \{\}$
  for $\mathcal{O} \in \{$ RandomOptimizer, BayesianOptimizer $\}$ **do**
    for $i \in \{1, 2 \dots N\}$ **do**
      $w^* \leftarrow \mathcal{O}(\mathcal{H}, D) = \underset{w}{\text{argmax}}(\mathcal{H}(w))$
      $f \leftarrow \mathcal{T}(w)$
      $y \leftarrow \mathcal{M}(f)$
      $Y \leftarrow Y \cup \{y\}$
    end for
    Display $E(Y)$ for dataset D and optimizer $\mathcal{O}$
  end for
end for

Table 4.3: Optimization Results

Dataset	Random Optimizer	Bayesian Optimizer
nba_logreg	0.691317070019488	0.7611633873392302
pima_indians_diabetes	0.6861976571409985	0.6763430545354692
imblearn_yeast_me2	0.2914189117327615	0.3035042336100022
wine_quality	0.3458412324861378	0.3347696886324151
imblearn_abalone	0.41379522291370424	0.41303731968474056

One could, however, thinks of the possibility that something is wrong with our implementation, which is unlikely the case thanks to the quality tests we have conducted. The quality of our implementation is perfectly up to par. The previous results have just proved the NFLT for machine learning problems, and has not yet been able to generalize for all optimization tasks.

Another possibility may be because of the way we conduct our experiment, in which we limit the optimizers to only 16 iterations. Will increasing the limit improve the situation? That will be left as another task for the future due to our insufficient computing resources.

Experiment Conclusion

This experiment confirmed the correctness of the No Free Lunch Theorem and partially explained the popularity of the naive random search technique despite more advanced methods such as Bayesian Optimization have been available for decades. Nevertheless, the outcome that both methods had the equivalent performance despite their difference in complexity is disappointing and may need more tests for reconfirmation.

2.3. Discovery 3: Are optimizers always good?

Challenge

The heart of our library - a hyperparameter optimization framework with support for LightGBM - is whether our optimizers can really improve the default performance of LightGBM. In this section, we would like to verify the efficacy of our software in boosting LightGBM performance.

Experiment Setup

In this experiment, we need to iterate through all datasets, and execute the default model f_0 of LightGBM with unmodified default hyperparameters. Detailed instructions are presented in Algorithm 5.

Experimental Results

Comparing Table 4.4 against Table 4.3 demonstrates that our framework can truly provide a boost to LightGBM's default performance on *most* datasets.

Algorithm 5 LightGBM Default Performance Experiment

for every dataset D **do**
  $Y \leftarrow \{\}$
  for $i \in \{1, 2 \dots N\}$ **do**
    $y \leftarrow \mathcal{M}(f_0)$
    $Y \leftarrow Y \cup \{y\}$
  end for
  Display $E(Y)$ for dataset D
end for

Table 4.4: LightGBM Default Performance

Dataset	**Default Performance**
nba_logreg	0.6643990929705216
pima_indians_diabetes	0.6052631578947368
imblearn_yeast_me2	0.32786885245901637
wine_quality	0.2328767123287671
imblearn_abalone	0.3525305410122164

The overall improvement is significant as can be shown in Table 4.5, especially for dataset wine_quality where Random Optimizer increase the default score by 48.51%, followed by dataset nba_logreg where the credit goes towards Bayesian Optimizer for improving the performance by 14.56%. It can be seen that the improvement varies greatly across different datasets, from highly significant to even negative as for dataset imblearn_yeast_me2, where the performance was downgraded by 11.12% by Random Optimizer and, slightly less, 7.43% by Bayesian Optimizer. In other words, it is not always a great idea to use optimizers for LightGBM. Nevertheless, it is the only dataset with negative improvement. One good result is that Bayesian Optimizer seems to perform slightly better than Random Optimizer on average, though the difference is insignificant (just around 1.5%).

Experiment Conclusion

This experiment showcased that modifying LightGBM's default hyperparameters would not always provide improved results and it was highly dependent on the datasets. Nevertheless, this experiment also confirmed that our optimizers were useful in most scenarios and hence should always be used for unknown datasets. Another worth mentioning insight

Table 4.5: Improvement by Optimizer (in %)

Dataset	**Random Optimizer**	**Bayesian Optimizer**
nba_logreg	4.051477	14.564182
pima_indians_diabetes	13.371787	11.743635
imblearn_yeast_me2	-11.117232	-7.431209
wine_quality	48.508294	43.754043
imblearn_abalone	17.378546	17.163557
Mean	14.438574	15.958842

is that Bayesian Optimizer seemed slightly superior to Random Optimizer in terms of overall improvement despite the difference being too small to be generalized.

2.4. Discovery 4: UCB and Hyperparameter Metric

Challenge

The first experiment has showed that the current metric for hyperparameter optimization is not so good, which is why searching for an alternative metric is the future of research in this area. Given the fact that we can not only compute the mean from cross-validation, but also the variance, is it not similar to a probabilistic model? If UCB acquisition function works well for a probabilistic model, will it apply for cross-validation?

Experiment Setup

Previously, the hyperparameter metric $\mathcal{H}(w)$ only returns a mean from cross-validation (CV). Applying the UCB algorithm, $\mathcal{H}$ will become

$$\mathcal{H}(w) = \mu + \beta\sigma \tag{4.4}$$

where

$$(\mu, \sigma^2) = CV(w) \tag{4.5}$$

The only thing to do is to search for a good β, which can be accomplished by repeating Algorithm 3 for a variety of β values in this experiment.

Table 4.6: UCB Correlation

β	**nba_logreg**	**pima_indians_diabetes**
-2.5	0.16903839945952664	0.5793467304066431
-2.0	0.2876221519692538	0.670921140183881
-1.5	0.10912976487479553	0.6989479639738452
-1.0	0.13328083607420466	0.766984973808668
-0.75	0.27746761445979445	0.8258799428507659
-0.5	0.3543745138129664	0.8557056170952839
-0.25	0.4872348585958887	0.8855090062842459
0.0	0.43723545983753864	0.8455329805409841
+0.25	0.3832589308441612	0.7563819261811323
+0.5	0.2556876864530273	0.8714067855924856
+0.75	0.18968321955145528	0.6995772333451941
+1.0	0.23779294914983842	0.8894566269960664
+1.5	0.10825333208539774	0.5135942919514589
+2.0	0.17543104529529283	0.3559708943927017
+2.5	0.07682568416668871	0.32668290132425665

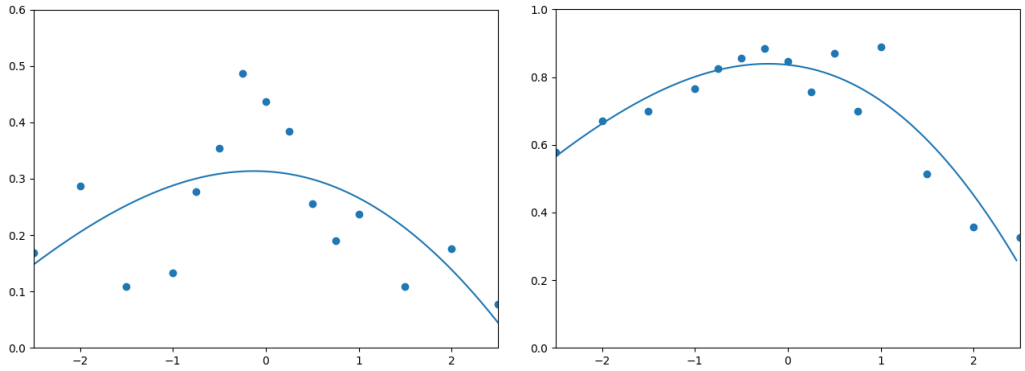
Experimental Results

The results are summarized in Table 4.6 for only 2 datasets: nba_logreg and pima_indians_diabetes. The reason is because this experiment is quite expensive, so we did not generalize to all datasets. The selected 2 are our smallest datasets by number of observations.

As can be seen, altering β could increase or decrease the correlation between the given metric and real-world performance. For clearer analysis, let's take a look at their graphical form in Figure 4.4. The results are really interesting.

The first conclusion can be drawn is that the mean itself is a very good indicator. In both tests, however, the best location is actually slightly below the mean $(\mu - \frac{\sigma}{4})$, but the current amount of data is insufficient to draw such a conclusion. Regardless, the mean is shown to provide an excellent correlation.

In addition, this experiment also rejects other indicators such as min and max since they would exceed the range $(\mu - \sigma, \mu + \sigma)$ which seems bad according to the curves.



(a) nba_logreg (b) pima_indians_diabetes

Figure 4.4: UCB Correlation Visualization

Experiment Conclusion

This experiment is too expensive and hence could not be performed on all datasets, which makes it insufficient to draw any conclusion such as $\mu - \frac{\sigma}{4}$ is better than μ. However, it has shown some interesting properties of β - its ability to decrease or increase the correlation - and rejected other indicators such as min and max for hyperparameter optimization.

3. Quality Assurance

This section provides some benchmarks for assuring the quality of our software. The 2 core benchmarks are for Bayesian Optimizer - the heart of our library - and LightGBM - the first-class citizen of our framework.

3.1. LightGBM Benchmark

In this subsection, we benchmarked our LightGBM API against its equivalent Python package on all datasets. In order to guarantee consistency, we provided the same set of hyperparameters for both (See Table 4.8). Table 4.7 summarizes our results and demonstrates that our product is fully functional with the only difference of 10^{-16} occurring for dataset pima_indians_diabetes.

Table 4.7: LightGBM Benchmark

Dataset	KotlinML API	Python API
nba_logreg	0.6480186480186481	0.6480186480186481
pima_indians_diabetes	0.6811145510835913	0.6811145510835912
imblearn_yeast_me2	0.3013698630136986	0.3013698630136986
wine_quality	0.3157894736842105	0.3157894736842105
imblearn_abalone	0.40052356020942403	0.40052356020942403

Table 4.8: LightGBM Sample Hyperparameters

Hyperparameters	Value
min_child_weight	16
num_leaves	96
max_depth	10
min_split_gain	0.01
subsample	0.8
lambda_l1	0.9
lambda_l2	0.9

3.2. Bayesian Optimizer Benchmark

The experiment for NFLT has led to suspicion that something is wrong with our Bayesian Optimizer, since it was providing equivalent or even worse results compared to Random Search in multiple test cases. The aim of this subsection is to prove that it is only applicable to machine learning and cannot be generalized, by performing a test on a different target: a mathematical function. Our target will be a familiar function we have met in chapter III:

$$f(x, y, z, w) = (x-1)^2 + (y-2)^2 + (z-3)^2 + (w-4)^2 \tag{4.6}$$

In this test, we will apply optimization using both Random Search and Bayesian Optimization. Both methods will be executed 5 times with 50 iterations each. The results are summarized in Table 4.9. As can be seen, they are vastly different. Albeit Bayesian Optimization cannot guarantee to always provide (nearly) optimal answers as shown in the 1^{st} and 5^{th} run, it was obviously much better than Random Search whose results were mostly in the thousands.

Table 4.9: Optimizer Benchmark

Run	Random Optimizer	Bayesian Optimizer
1	3260.1505901050186	88.05385775935363
2	2494.573608296885	-0.1363935072606174
3	156.09823687476452	0.25149950301425567
4	2276.8016588840196	0.2919311180186169
5	2338.6164402435124	442.8331295378901

In other words, our Bayesian Optimizer is perfectly functional. Nonetheless, it is worth noting that even with 50 iterations allowed, Bayesian Optimization is still incapable of producing sub-optimal results, which might have been the reason behind its poor performance in the experiment for NFLT where we only allowed 16 iterations per run. Regardless, the verification will be left for the future since our hardware is still not so capable of large number of iterations. In addition, given the currently not so good metric for hyperparameter optimization, it is pointless to optimize it further, since higher metric evaluation does not guarantee real-world performance.

Summary and Conclusion

Conclusion

This project truly started in the second week of December 2019 when official support for Data Science was announced for Kotlin. Within the length of a month or more, we are delighted to announce the birth of KotlinML (https://github.com/duckladydinh/KotlinML) - a functional machine learning library and probably the first of its kind with the first ever support for LightGBM, L-BFGS-B, Bayesian Optimization and Gaussian Process in Kotlin.

In addition, we also explored various aspects of Machine Learning and Data Science, such as verifying the No Free Lunch theorem, confirming that it will be a good idea to always optimizer hyperparameters in most scenarios, analyzing cross-validation mean as the current hyperparameter metric and even proposing a new type of metric using the Upper-Confidence Bound algorithm which, unfortunately, has too little data to support its validity. The progress was intense but exciting. For the first time ever, we have found a true application of probability in reality via implementing the Gaussian Process based on just a few abstract mathematical formulas. The technical work was also significant. It was our first time ever to try using Gradle as our build tool, the first time ever to perform DevOps [63] practices using GitHub Actions as well as the first time ever to build a project of this size. This is basically the biggest project ever in all my days at the Frankfurt University of Applied Sciences.

Most importantly, our goal to study the Bayesian Optimization algorithm - the most important technique for hyperparameter optimization - has been successfully completed, even if much of its related concepts such as

Gaussian Process and acquisition functions have been still a mystery to us due to our lack of knowledge in multivariate probability and matrix algebra. Regardless, we still managed to gain a very good intuition of those complex concepts, which will be essential for future research.

Last, but not least, we are proud that our project is one with significant practical value and confident that it will have the potential to be a pillar of Machine Learning and Data Science on JVM one day given its status as being the very first of its kind for Kotlin. Albeit this hope is a little far-reached, it is definitely not impossible and only time can answer.

Future Work

This project has succeeded in completing its original goals but at the same time it left numerous tasks for future research.

Firstly, is Bayesian Optimization really not much better than Random Search for hyperparameter optimization? Even if our experiments demonstrated this fact, we still find it hard to accept given the overwhelming complexity difference in implementing them. Is it possibly because of our insufficient computational resource that failed to reveal Bayesian Optimzation true potential? More tests need to be performed to answer this question.

Secondly, we have proved that the current metric for hyperparameter optimization is not of excellent quality due to its poor correlation with real-world expected performance. Another metric should be found or hyperparameter optimization will not be able to advance. That is undoubtedly no trivial task.

Last, but not least, even if the implementation is already functional, we must prepare for the public availability of KotlinML by publishing it to popular repositories such as Maven Central and JCenter. In addition, the software needs to be cross-platform, meaning that it has to provide support for operating systems other than Ubuntu. This will involve a lot of technical work but will be the simplest to complete compared to others.

Bibliography

[1] Bill Venners. *The java virtual machine*. McGraw-Hill, New York, 1998.

[2] Jake VanderPlas. *Python data science handbook: essential tools for working with data*. " O'Reilly Media, Inc.", 2016.

[3] Pascal Bugnion. *Scala for data science*. Packt Publishing Ltd, 2016.

[4] Andrew Ng. What is machine learning? URL: https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v.

[5] A. Burkov. *The Hundred-page Machine Learning Book*. Andriy Burkov, 2019. URL: https://books.google.de/books?id=0jbxwQEACAAJ.

[6] JC Becsey, Laszlo Berke, and James R Callan. Nonlinear least squares methods: A direct grid search approach. *Journal of Chemical Education*, 45(11):728, 1968.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, pages 281–305, 2012.

[8] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.

[9] Georges R Harik, Fernando G Lobo, and David E Goldberg. The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4):287–297, 1999.

[10] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

[11] Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950. IEEE, 1999.

[12] José Mario Martınez. Practical quasi-newton methods for solving nonlinear systems. *Journal of Computational and Applied Mathematics*, 124(1-2):97–121, 2000.

[13] Yun Fei, Guodong Rong, Bin Wang, and Wenping Wang. Parallel l-bfgs-b algorithm on gpu. *Computers & Graphics*, 40:1–9, 2014.

[14] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

[15] Steven C Chapra, Raymond P Canale, et al. *Numerical methods for engineers*. Boston: McGraw-Hill Higher Education,, 2010.

[16] Harold Szu and Ralph Hartley. Fast simulated annealing. *Physics letters A*, 122(3-4):157–162, 1987.

[17] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[18] Jun Shao. Linear model selection by cross-validation. *Journal of the American statistical Association*, 88(422):486–494, 1993.

[19] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[20] Joel Myerson, Leonard Green, and Missaka Warusawitharana. Area under the curve as a measure of discounting. *Journal of the experimental analysis of behavior*, 76(2):235–243, 2001.

[21] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 525–532. Morgan Kaufmann Publishers Inc., 1999.

[22] Vijay K Rohatgi and AK Md Ehsanes Saleh. *An introduction to probability and statistics*. John Wiley & Sons, 2015.

[23] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[24] Yung Liang Tong. *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[25] Steven W Nydick. The wishart and inverse wishart distributions. *J. Stat*, 2012.

[26] Mourad EH Ismail et al. Bessel functions and the infinite divisibility of the student *t*-distribution. *The Annals of Probability*, 5(4):582–585, 1977.

[27] Amar Shah, Andrew Wilson, and Zoubin Ghahramani. Student-t processes as alternatives to gaussian processes. In *Artificial intelligence and statistics*, pages 877–885, 2014.

[28] Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

[29] Romain Benassi, Julien Bect, and Emmanuel Vazquez. Robust gaussian process-based global optimization using a fully bayesian expected improvement criterion. In *International Conference on Learning and Intelligent Optimization*, pages 176–190. Springer, 2011.

[30] Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5267–5278, 2017.

[31] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501, 2016.

[32] Felipe Viana and Raphael Haftka. Surrogate-based optimization with parallel simulations using the probability of improvement. In *13th AIAA/ISSMO multidisciplinary analysis optimization conference*, page 9392, 2010.

[33] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

[34] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[35] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, 2012.

[36] Kirthevasan Kandasamy, Gautam Dasarathy, Junier B Oliva, Jeff Schneider, and Barnabás Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems*, pages 992–1000, 2016.

[37] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[38] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[39] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[40] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pages 6638–6648, 2018.

[41] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 329. John Wiley & Sons, 2012.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[43] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[44] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[45] Wenxin Jiang. Some theoretical aspects of boosting in the presence of noisy data. In *Proceedings of the Eighteenth International Conference on Machine Learning*. Citeseer, 2001.

[46] A Van Den Bos. Complex gradient and hessian. *IEE Proceedings-Vision, Image and Signal Processing*, 141(6):380–382, 1994.

[47] Quazi Nafiul Islam. *Mastering PyCharm*. Packt Publishing Ltd, 2015.

[48] IDEA IntelliJ. the most intelligent java ide. *JetBrains [online].[cit. 2016-02-23]. Dostupné z: https://www. jetbrains. com/idea/# chooseYourEdition*, 2011.

[49] Manish Jangid. Kotlin the unrivalled android programming language lineage. *Imperial Journal of Interdisciplinary Research*, 3(8):256–259, 2017.

[50] Jetbrains. Kotlin for data science. URL: https://kotlinlang.org/docs/reference/data-science-overview.html.

[51] Jeffrey M Perkel. Why jupyter is data scientists' computational notebook of choice. *Nature*, 563(7732):145–147, 2018.

[52] Benjamin Muschko. *Gradle in action*. Manning, 2014.

[53] Inc Github. Github, 2016.

[54] ktlint. An anti-bikeshedding kotlin linter with built-in formatter, 2019. URL: https://ktlint.github.io/.

[55] Mateusz Kobos. Java wrapper for the fortran l-bfgs-b algorithm, 2016. URL: https://github.com/mkobos/lbfgsb_wrapper.

[56] Jason Brownlee. 10 standard datasets for practicing applied machine learning. URL: https://machinelearningmastery.com/standard-machine-learning-datasets.

[57] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1):559–563, 2017.

[58] Oracle. Oracle graalvm enterprise edition, 2019. URL: https://www.oracle.com/technetwork/graalvm/overview/index.html.

[59] Jetbrains. Intellij idea, 2019. URL: https://www.jetbrains.com/idea/.

[60] Canonical. Download ubuntu desktop, 2019. URL: https://ubuntu.com/download/desktop.

[61] Microsoft. Parameters tuning. URL: https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html.

[62] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of optimization theory and applications*, 115(3):549–570, 2002.

[63] Liming Zhu, Len Bass, and George Champlin-Scharff. Devops and its practices. *IEEE Software*, 33(3):32–34, 2016.