Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Mathematics and Data Analysis

Analysis of Bayesian Optimization components in terms of the properties of exploration and exploitation

Alicja Żebiałowicz

student record book number 327236

thesis supervisor

Katarzyna Woźnica, Ph.D.

consultation

Professor Przemysław Biecek, Ph.D., D.Sc.

WARSAW 2025

Abstract

Analysis of Bayesian Optimization components in terms of the properties of exploration and exploitation

Bayesian Optimization is an effective method for optimizing expensive black-box functions, particularly in machine learning applications such as hyperparameter tuning. Its efficiency largely depends on the balance between exploration - searching unknown regions of the input space - and exploitation - focusing on areas already known to yield good results.

This thesis investigates the impact of key components of Bayesian Optimization on the exploration - exploitation trade-off. The theoretical part provides an overview of surrogate models such as Gaussian Processes, Random Forests, Extra Trees and acquisition functions such as Expected Improvement, Probability of Improvement, and Lower Confidence Bound.

An experiment was conducted, in which various combinations of surrogate models and acquisition functions were evaluated across multiple datasets. The performance of Bayesian Optimization was compared to traditional methods such as Grid Search and Random Search. Results were analyzed based on convergence behavior and overall efficiency, with a particular focus on how individual components affect the search dynamics.

The findings confirm that well-chosen components, especially Gaussian Processes with the Lower Confidence Bound, lead to faster convergence and better optimization outcomes. This analysis contributes to a better understanding of how Bayesian Optimization methods manage the exploration- exploitation balance in practical machine learning scenarios.

Keywords: Bayesian Optimization, surrogate model, acquisition function, Gaussian Process, exploration vs. exploitation, hyperparameter tuning, machine learning.

Streszczenie

Analiza komponentów Optymalizacji Bayesowskiej w ujęciu eksploracji i eksploatacji

Optymalizacja Bayesowska jest skuteczną metodą optymalizacji kosztownych funkcji typu „czarna skrzynka", szczególnie w zastosowaniach związanych z uczeniem maszynowym, takich jak dostrajanie hiperparametrów. Jej efektywność w dużej mierze zależy od zachowania równowagi między eksploracją – czyli przeszukiwaniem nieznanych obszarów przestrzeni wejściowej – a eksploatacją, czyli koncentrowaniem się na obszarach już znanych jako obiecujące.

Niniejsza praca analizuje wpływ kluczowych komponentów Optymalizacji Bayesowskiej na kompromis pomiędzy eksploracją a eksploatacją. Część teoretyczna zawiera przegląd modeli zastępczych, takich jak procesy gaussowskie, lasy losowe (Random Forests), oraz drzewa losowe (Extra Trees), a także funkcji akwizycji, takich jak oczekiwane ulepszenie (Expected Improvement), prawdopodobieństwo ulepszenia (Probability of Improvement) oraz dolna granica ufności (Lower Confidence Bound).

Przeprowadzono eksperyment, w ramach którego oceniano różne kombinacje modeli zastępczych i funkcji akwizycji na wielu zbiorach danych. Wydajność Optymalizacji Bayesowskiej została porównana z metodami tradycyjnymi, takimi jak przeszukiwanie siatki (Grid Search) i losowe przeszukiwanie (Random Search). Analizowano zachowanie zbieżności oraz ogólną efektywność, ze szczególnym uwzględnieniem wpływu poszczególnych komponentów na dynamikę procesu poszukiwania.

Wyniki potwierdzają, że odpowiednio dobrane komponenty – zwłaszcza procesy gaussowskie w połączeniu z funkcją dolnej granicy ufności (Lower Confidence Bound) – prowadzą do szybszej zbieżności i lepszych wyników optymalizacji. Analiza ta przyczynia się do lepszego zrozumienia, w jaki sposób metody Optymalizacji Bayesowskiej zarządzają równowagą między eksploracją a eksploatacją w praktycznych zastosowaniach uczenia maszynowego.

Słowa kluczowe: Optymalizacja Bayesowska, model zastępczy, funkcja akwizycji, proces gaussowski, eksploracja vs. eksploatacja, dostrajanie hiperparametrów, uczenie maszynowe.

Contents

1. Introduction

Bayesian Optimization is a method commonly used to optimize objective functions that are expensive to evaluate and lack a known analytical form. These characteristics make such functions difficult to optimize using traditional methods, especially when evaluations are time-consuming or computationally intensive. Bayesian Optimization is particularly effective in scenarios where evaluation costs are high and the search space is complex, as is often the case in hyperparameter tuning in machine learning. The method relies on building a surrogate model of the objective function, which serves as an approximation that is cheaper to evaluate. Based on this model, the algorithm selects new points in the search space using an acquisition function, which quantifies the potential benefit of evaluating a given point. A key aspect of this optimization strategy is the balance between exploration - searching unexplored areas of the domain - and exploitation - focusing on regions already known to offer promising results. Maintaining this balance is crucial for efficient and effective optimization, as it allows the algorithm to discover high-performing regions without wasting resources on unproductive areas.

The purpose of this thesis is to analyze the components of Bayesian Optimization with respect to this exploration - exploitation trade-off. The first part of the work presents the theoretical background necessary to understand the algorithm, including concepts such as surrogate models and acquisition functions.

In the second part, an experiment is conducted to illustrate how it looks in practice. The implementation is carried out in Python and used to compare the optimization process under varying configurations. The experiment aims to demonstrate how the design of these components influences the overall performance and strategy of Bayesian Optimization compared to Grid Search and Random Search.

Through this analysis, the thesis seeks to highlight the role of each component in managing the balance between exploration and exploitation, and to provide insight into their practical application.

2. Bayesian probability theory

Bayesian probability theory is a mathematical framework for updating beliefs based on new evidence. It is rooted in Bayes' Theorem, which describes how to revise probabilities when given new data. The key idea is that instead of treating probabilities as fixed frequencies (as in classical statistics), Bayesian probability interprets them as degrees of belief that can change as more information becomes available.

2.1. Bayes' Theorem

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is the event space, and $\mathbb{P}$ is the probability measure.

Definition 2.1 (Conditional probability).
For two events $A, B \in \mathcal{F}$ with the unconditional probability of B being greater than zero (i.e. $\mathbb{P}(B) > 0$), the *conditional probability* of A given B is defined as:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

Theorem 2.2 (Bayes' Theorem).
Let A and B be events in a probability space where $\mathbb{P}(B) > 0$. Then the formula

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A) \cdot \mathbb{P}(A)}{\mathbb{P}(B)}$$

holds.

2.2. Bayesian inference

Bayesian inference distinguishes itself from the frequentist approach by interpreting observed quantities as realizations drawn from a family of probability distributions, parameterized through random variables.

The following paragraphs present the fundamental concepts underlying Bayesian inference.

Let us assume that the information about the parameter θ will be represented by the so-called *prior distribution* $\pi(\theta)$ defined on the parameter space Θ.

Given the prior distribution $\pi(\theta)$ and the sample distribution $p_\theta(\mathbf{x})$, we can determine *the posterior distribution* density from the formula

$$\pi(\theta \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \theta) \cdot \pi(\theta)}{\int_\Theta p(\mathbf{x} \mid \theta) \cdot \pi(\theta) d\theta}, \tag{2.1}$$

which follows from Bayes' Theorem 2.2.

To describe it more literally: A prior distribution expresses our belief about a parameter before seeing the data. A posterior distribution updates that belief based on the data we observe.

In the following interpretation the sample distribution $p(\mathbf{x} \mid \theta)$ is treated as function of parameter θ given fixed observed data $\mathbf{x}$ and is called *likelihood*. Note that while expressed using conditional probability notation, the likelihood is not a probability distribution over θ - it becomes one only through Bayes' Theorem 2.2, yielding the posterior $\pi(\theta \mid \mathbf{x})$.

From Equation 2.1 we have:

$$\pi(\theta \mid \mathbf{x}) \propto p(\mathbf{x} \mid \theta) \cdot \pi(\theta),$$

where $A \propto B$ means that A is proportional to B, i.e. there is a constant k such that $A = k \cdot B$.

In other words, we could say that a posteriori knowledge is proportional to product of likelihood and a priori knowledge.

3. Optimization in machine learning

Optimization plays a central role in both classical and modern data analysis, including machine learning and artificial intelligence. It provides a formal mathematical framework for improving models, systems and algorithms by systematically searching for the best possible configuration or solution according to a predefined criterion. In the context of machine learning, optimization is used at multiple levels: training models by minimizing a loss function, tuning hyperparameters to enhance generalization, and selecting features or models to improve performance. This chapter introduces the foundational concepts of optimization, explains the formal definitions of local and global optima, and illustrates how these ideas are applied to hyperparameter optimization in machine learning.

3.1. Fundamentals of optimization

An objective function is a mathematical representation of the goal to be achieved in a problem. It quantifies the quality or performance of a solution (e.g. machine learning model quality) by assigning a scalar value to a set of decision variables (inputs, parameters, or configurations). The purpose of optimization is to find the values of these variables that minimize or maximize the objective function, depending on the problem's requirements.

Throughout this thesis, we consider only single-objective optimization. All subsequent definitions and analyses apply solely to this case.

Definition 3.1 (The optimization problem).
Given function $f : A \to \mathbb{R}$, where $A \subset \mathbb{R}^n$, *an optimization problem* consist of finding $x^* \in A$ such that $f(x^*) \leqslant f(x)$ for all $x \in A$.

Definition 3.2 (The local minimum).
The function $f : A \to \mathbb{R}$ has *a local minimum* at the point x^* if there exists $\delta > 0$ such that for all $x \in A$ where $\|x - x^*\| \leqslant \delta$ the expression $f(x^*) \leqslant f(x)$ holds.

Definition 3.3 (The local maximum).

The function $f : A \to \mathbb{R}$ has *a local maximum* at the point x^* if there exists $\delta > 0$ such that for all $x \in A$ where $\|x - x^*\| \leqslant \delta$ the expression $f(x^*) \geqslant f(x)$ holds.

Definition 3.4 (The global minimum).

The function $f : A \to \mathbb{R}$ has *a global minimum* at the point x^* if for all $x \in A$ the expression $f(x^*) \leqslant f(x)$ holds.

Definition 3.5 (The global maximum).

The function $f : A \to \mathbb{R}$ has *a global maximum* at the point x^* if for all $x \in A$ the expression $f(x^*) \geqslant f(x)$ holds.

Figure 3.1 presents an exemplary function with marked local and global extrema.

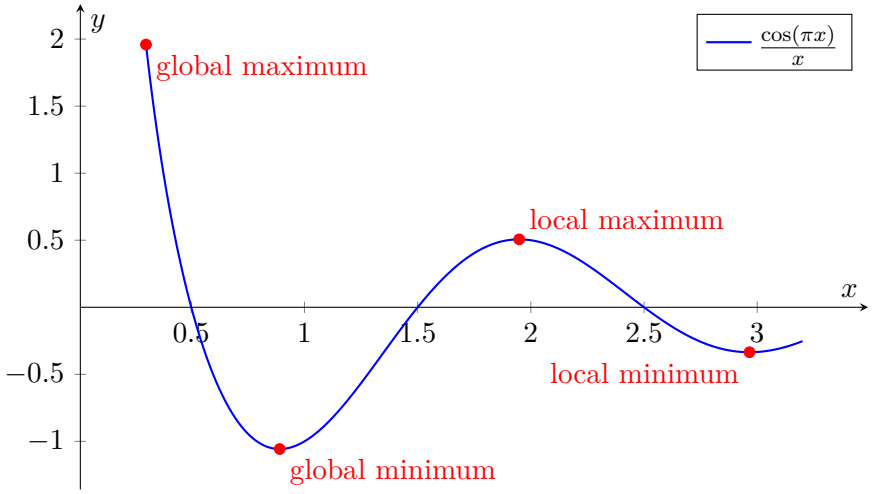

Figure 3.1: Plot of the function $\frac{\cos(\pi x)}{x}$ for $x \in [0.3,\ 3.2]$ with marked local and global extrema as an example.

3.2. Hyperparameter optimization in machine learning

Hyperparameter Optimization (HPO) in machine learning is the process of selecting the optimal values of hyperparameters (HPs) for a machine learning (ML) model to achieve the best performance on a validation dataset. The goal of selecting algorithms and optimizing machine learning models is to choose the model that gives the smallest generalization error. This section is based on [Bischl et al., 2023].

Definition 3.6 (Generalization error).

The generalization error (GE) is defined as the expected value error of the model on the distribution of data from which the samples are taken.

In practice, it is estimated by:

$$\widehat{GE}(\mathcal{I}_\lambda, \mathcal{D}, L) = \frac{1}{B} \sum_{b=1}^{B} L(y_{D_{test,b}}, \hat{y}_{D_{test,b}}).$$

Each component of this formula has a specific role in the estimation process:

- $\mathcal{I}$ is a learning algorithm. This represents the ML method trained on a training dataset. It could be, for example, a decision tree, a support vector machine, or a logistic regression. The learner receives training data and a set of HPs to construct a predictive model.

- λ is a vector of HPs. This refers to the configuration of model hyperparameters, which are fixed prior to training (e.g., the regularization strength, maximum depth of a tree, or learning rate). These values are not learned directly from the data, but can significantly affect model performance.

- $\mathcal{I}_\lambda$ is the learner instantiated with λ. Denotes the algorithm $\mathcal{I}$ configured with fixed vector of HPs λ.
  For example, $\mathcal{I}_\lambda$ could be a Random Forest with $\lambda = \{$`max_depth` $= 5$, `n_estimators` $= 100\}$.

- $\mathcal{D} = ((D_{train,1}, D_{test,1}), ..., (D_{train,B}, D_{test,B}))$ is a set of data partitions used in the resampling process. This denotes the collection of B data splits used to estimate performance, where B is the total number of splits. Each element of $\mathcal{D}$ is a pair consisting of a training set and a corresponding test set. These splits typically originate from a resampling technique such as k-fold cross-validation or bootstrap. They allow the model to be evaluated multiple times on different subsets of the data.

- L is a loss or evaluation metric: This is the function used to quantify the discrepancy between the true target values and the predicted values. Depending on the task, this could be the mean squared error (MSE), mean absolute error (MAE) in regression problems, accuracy in classification problems, or another relevant metric.

- $y_{D_{test,b}}$ is a vector of true target values related to $D_{test,b}$. These are the ground-truth outputs associated with the test set in the b-th data split. They represent the actual outcomes the model is expected to predict by model $\mathcal{I}_\lambda$.

- $\hat{y}_{D_{test,b}}$ are predicted target values by trained model $\mathcal{I}_\lambda$ for $D_{test,b}$. These are the model's predictions for the test set in the b-th split, generated after training on the corresponding training set $D_{train,b}$.

Some evaluation metrics are intended to be maximized, while others should be minimized, depending on the goal. For example, metrics such as accuracy, precision, recall, and AUC score are typically maximized, as higher values indicate better model performance. In contrast, metrics like log loss and mean squared error are minimized, since lower values reflect lower prediction error and better model fit.

Remark 3.7.
Unlike model parameters (which are learned during training), hyperparameters are set by the user before training begins and are not adjusted during the learning process. For example, a model parameter could be the coefficients in a logistic regression model - they are learned from the training data and a hyperparameter might be the regularization strength (C) in logistic regression - it is defined before training and influence how the model learns.
Most ML models are highly configurable by HPs, and their generalization performance usually depends on this configuration, as shown by Probst et al. [2019].

Definition 3.8 (The search space).
Let $\mathcal{I}$ be a learning algorithm with l hyperparameters, where each hyperparameter λ_i ($i = 1, \ldots, l$) has a domain Λ_i that can be continuous, discrete, or categorical. The *search space* $\Lambda \subset \mathbb{R}^l$ is defined as the Cartesian product of these domains under the assumption of hyperparameter independence:

$$\Lambda = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_l.$$

Each point $\lambda = (\lambda_1, \ldots, \lambda_l) \in \Lambda$ represents a unique hyperparameter configuration for $\mathcal{I}$.

The search space can also contain *dependent HPs*, leading to a hierarchical search space:
An HP λ_i is said to be *conditional* on λ_j if λ_i is only active when λ_j is an element of a given subset of Λ_j and inactive otherwise, i.e., not affecting the resulting learner. Conditional HP is, for example, regularization in linear regression.

Definition 3.9 (HPO problem).
The general HPO problem is defined as finding a hyperparameter configuration that minimizes

the estimated generalization error $\widehat{GE}$:

$$\lambda^* \in \underset{\lambda \in \Lambda}{\operatorname{argmin}}\ c(\lambda) = \underset{\lambda \in \Lambda}{\operatorname{argmin}}\ \widehat{GE}(\mathcal{I}_\lambda, \mathcal{J}, L),$$

where λ^* denotes the theoretical optimum, L is the loss function used to evaluate model performance and $c(\lambda)$ is a shorthand for the estimated generalization error of a learner $\mathcal{I}_\lambda$ with reference to an HPC λ, based on a resampling split $\mathcal{D} = ((D_{train,1}, D_{test,1}), ..., (D_{train,B}, D_{test,B}))$.

Remark 3.10.
Note that $c(\lambda)$ is a black-box, as it usually has no closed-form mathematical representation, and hence no analytic gradient information is available. Furthermore, the evaluation of $c(\lambda)$ can take a significant amount of time. Therefore, the minimization of $c(\lambda)$ forms an *expensive black-box* optimization problem.

3.3. Fundamental methods of HPO

Having introduced the importance of hyperparameter optimization, this section explores search-based methods, one of the fundamental strategies used to identify optimal hyperparameter configurations.

Definition 3.11 (Grid Search).
Grid Search is a brute-force HPO method that evaluates all hyperparameter combinations from a predefined grid.
Grid Search steps:

1. Discretize hyperparameter ranges into a finite set of values.

2. Train and evaluate the model for every combination.

3. Select the configuration with the best performance.

Grid Search offers several advantages. It is straightforward to implement and ensures that the best combination of hyperparameters within the defined search space is identified. However, this method also has its limitations. It can be highly computationally expensive, particularly when dealing with a large number of hyperparameters or a wide range of values. Moreover, Grid Search may be inefficient if certain hyperparameters have little influence on model performance, as it still evaluates all combinations regardless of their relevance (see Figure 3.2).

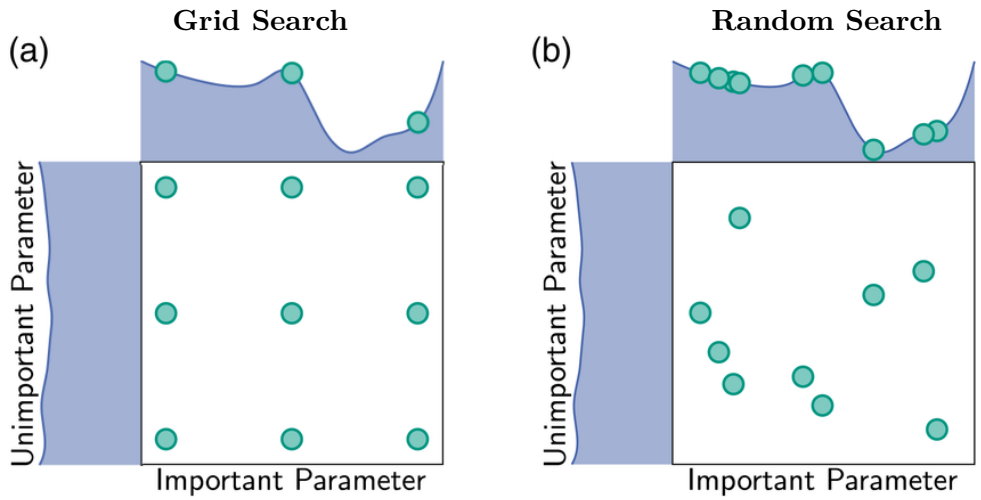

Figure 3.2: This figure presents a comparison of Grid Search and Random Search. It is adapted from [Meisenbacher et al., 2022]. It consists of a square representing the two-dimensional hyperparameter space. Within the square, points indicate the specific hyperparameter combinations evaluated during the search. Grid Search samples points in a systematic grid pattern, while Random Search selects points randomly across the space. Above and to the left of the square, marginal plots show the function's output (e.g. accuracy, loss) corresponding to individual hyperparameter values, illustrating how it changes along each hyperparameter dimension separately.

Definition 3.12 (Random Search).
Random Search samples hyperparameters randomly from predefined distributions over multiple iterations.

Random Search steps:

1. Define probability distributions for hyperparameters.
2. Sample a configuration randomly and evaluate the model.
3. Repeat for a fixed number of iterations.
4. Select the best-performing configuration.

Random Search presents several advantages over Grid Search, particularly in high-dimensional hyperparameter spaces. It is generally more efficient, as it does not waste resources evaluating combinations of hyperparameters that have little impact on model performance. Instead, it samples randomly, which increases the chances of discovering regions with high-performing configurations, especially when only a few hyperparameters are truly influential (see Figure 3.2). However, Random Search also has drawbacks. It does not guarantee that the global

optimum will be found, since the sampling is not exhaustive. Additionally, it may require careful manual tuning of the sampling distributions for each hyperparameter to be effective.

3.4. Selected machine learning algorithms

This section presents a detailed overview of the selected machine learning algorithms: Random Forest, Gradient Boosting, XGBoost and LightGBM. Each of these methods builds upon decision trees and uses ensemble learning techniques to improve predictive accuracy and robustness.

3.4.1. Random Forest

Random Forest is a bagging-based ensemble method that constructs multiple decision trees using random subsets of the training data and features. Each tree in the forest is trained independently, and the final prediction is obtained by aggregating their outputs using voting for classification or averaging for regression. Mathematically, the prediction $\hat{y}$ of a Random Forest model for an input x is given by:

$$\hat{y} = \begin{cases} \frac{1}{M} \sum_{m=1}^{M} d_m(x) & \text{for regression,} \\ \underset{c}{\operatorname{argmax}} \sum_{i=1}^{M} \mathbb{I}(d_m(x) = c) & \text{for classification,} \end{cases}$$

where M is the number of trees,$d_m(x)$ is the prediction of the m-th tree, c is class label and $\mathbb{I}$ is the indicator function.

The randomization introduced during tree construction helps to reduce overfitting and improves generalization. Random Forest is widely appreciated for its simplicity, robustness to noise, and strong baseline performance across a variety of tasks.

For Random Forest, some key hyperparameters control the structure and complexity of the individual decision trees. The `max_depth` limits how deep each tree can grow, preventing overly complex trees that may overfit the training data. The `n_estimators` determines how many trees form the ensemble, with more trees generally improving robustness but increasing computational cost. Additionally, `min_samples_split` and `min_samples_leaf` set thresholds for how many training samples are needed to create a split or to form a leaf node, respectively. These parameters help regulate the tree growth and reduce overfitting by ensuring that splits only happen when there is sufficient data support.

3.4.2. Gradient Boosting

Unlike Random Forest, which builds independent trees in parallel, *Gradient Boosting* constructs decision trees sequentially, with each new tree learning from the mistakes of the previous ones. Instead of averaging multiple models, it incrementally improves predictions by focusing on residual errors - the differences between the current model's predictions and the true values. This approach allows Gradient Boosting to gradually refine its accuracy, often achieving higher performance than bagging-based methods like Random Forest. However, it requires careful tuning to avoid overfitting, as the sequential nature of training makes it more sensitive to noise in the data.

Key hyperparameters influencing gradient boosting models include the `learning_rate`, which controls the size of updates at each step and balances convergence speed with stability, and the `n_estimators`, which sets how many trees are added to the model. To avoid overfitting, gradient boosting frameworks often use subsampling - training each tree on a random fraction of the data - and apply regularization techniques such as L2 penalties on leaf weights, which help control model complexity.

3.4.3. XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful gradient boosting algorithm designed for performance and efficiency. It builds decision trees sequentially, where each new tree attempts to correct the errors made by the previous ones. One of XGBoost's key strengths is its use of both L1 (Lasso) and L2 (Ridge) regularization to control model complexity and prevent overfitting. This makes the algorithm more robust, especially when dealing with noisy data or high-dimensional features. Additionally, XGBoost supports parallel processing, handles missing values natively, and includes various optimization techniques to improve training speed.

In the case of XGBoost, the learning process is influenced by the `learning_rate`, which controls the step size during each model update, balancing between rapid convergence and stable learning. Like Random Forest, `max_depth` restricts the depth of each individual tree, while `n_estimators` specifies the total number of trees added sequentially to correct previous errors. The `subsample` parameter introduces randomness by using only a fraction of the training data for each tree, which helps prevent overfitting. Furthermore, `reg_lambda` applies L2 regularization on the leaf weights to control model complexity and improve generalization. During training, model performance is monitored using an `eval_metric` such as logloss for classification, guiding the optimization process.

3.4.4. LightGBM

LightGBM (Light Gradient-Boosting Machine) is a gradient boosting framework developed by Microsoft, optimized for speed and memory efficiency. It uses a histogram-based approach for splitting nodes, which significantly accelerates training, especially on large datasets. LightGBM builds trees leaf-wise (i.e., it grows the leaf with the highest loss reduction first), which often leads to deeper trees and better accuracy. Like XGBoost, LightGBM also includes regularization mechanisms, but it primarily uses L2 (Ridge) regularization. It is particularly effective for large-scale learning tasks and handles categorical features natively, without requiring one-hot encoding.

LightGBM shares many of these hyperparameters with XGBoost but includes some additional ones tailored for its architecture. The `learning_rate` and `n_estimators` similarly control the pace and extent of boosting iterations. The `subsample` parameter again controls the fraction of data used per tree to reduce overfitting. The `reg_lambda` parameter introduces L2 regularization on leaf weights to prevent overly complex models. Unique to LightGBM, `num_leaves` sets the maximum number of leaves a tree can have, directly influencing the shape and depth of trees grown leaf-wise. Moreover, `force_col_wise` forces the use of a column-wise data format, which can improve training efficiency in certain scenarios. Lastly, `min_gain_to_split` establishes the minimum gain required to perform a split at a tree node, helping to avoid unnecessary splits that do not significantly improve the model.

4. Bayesian Optimization

Bayesian Optimization is an efficient method for optimizing costly or complex functions where evaluations are expensive or time-consuming. It leverages probabilistic surrogate models - most notably Gaussian Processes, which are discussed in more detail - to approximate the objective function, along with acquisition functions that decide where to sample next. This chapter introduces the overall workflow of Bayesian Optimization, explains the role of surrogate models including a more in-depth treatment of Gaussian Processes, and details acquisition strategies that guide the search for optimal solutions.

4.1. Gaussian Processes

Gaussian Processes are fundamental tools in Bayesian Optimization and machine learning for modeling unknown continuous functions. They provide a flexible, probabilistic framework that allows efficient exploration and exploitation of complex search spaces. In this chapter, we introduce their key properties, including mean and covariance functions, and explain the mathematical foundations underlying Gaussian Processes.

Definition 4.1 (The canonical Gaussian distribution).
The canonical Gaussian distribution γ_d in $\mathbb{R}^d$ is the distribution of a d-dimensional random vector $\mathbf{X} = (X_1, ..., X_d)^T$ with independent coordinates, distributed according to the standard normal distribution.
γ_d has a density relative to Lebesgue measure $\gamma_d(dx) = (2\pi)^{-\frac{d}{2}} \exp(-\frac{x^T x}{2})dx$,
where $\mathbf{x} = (x_1, ..., x_d) \in \mathbb{R}^d$.

Remark 4.2.
Key properties of the canonical Gaussian distribution, where $\mathbf{X} \sim N(0, \mathbf{I}_d) = \gamma_d$, are:

- $\mathbb{E}[\mathbf{X}] = 0$, where $\mathbb{E}[\mathbf{X}]$ is expected value of $\mathbf{X}$,
- $\Sigma_X := \mathrm{Cov}(\mathbf{X}) = \mathbf{I}_d$, where $\mathrm{Cov}(\mathbf{X})$ is covariance matrix of $\mathbf{X}$,

- $M_{\mathbf{X}}(t) = \exp(\frac{1}{2}t^T t)$, where $M_{\mathbf{X}}$ is a moment-generating function of $\mathbf{X}$

Definition 4.3 (Gaussian distribution).
A measure μ on $\mathbb{R}^d$ is called a *Gaussian distribution* in $\mathbb{R}^d$ if every linear combination of the coordinates of a random vector $\mathbf{X} \sim \mu$ is normally distributed, that is, for every $v \in \mathbb{R}^d$, the real-valued random variable $v^T\mathbf{X}$ follows a one-dimensional Gaussian distribution.

Definition 4.4 (Gaussian vector).
A random vector $\mathbf{X}$ in $\mathbb{R}^d$ is called *Gaussian vector* if its distribution is Gaussian, i.e. if

$$X \overset{d}{=} A\mathbf{Y} + a$$

for some $A \in \mathbb{R}^{d\times m}, a \in \mathbb{R}^d, \mathbf{Y} \sim \gamma_m$.

Lemma 4.5.
If $\mathbf{X}$ is a Gaussian vector and $\mathbf{X} \overset{d}{=} A\mathbf{Y} + a$ (where $A \in \mathbb{R}^{d\times m}, a \in \mathbb{R}^d, \mathbf{Y} \sim \gamma_m$), then:

- $\mathbb{E}[\mathbf{X}] = a$,
- $\Sigma_{\mathbf{X}} := \mathrm{Cov}(\mathbf{X}) = AA^T$,
- $M_{\mathbf{X}}(t) = \exp(t^T a + \frac{1}{2}t^T\Sigma_{\mathbf{X}}t)$.

Lemma 4.6.
Let $X = (X_1, ..., X_d)'$ be a Gaussian vector in $\mathbb{R}^d$. Then the random variables $X_1, ..., X_d$ are independent if and only if they are not correlated.

Theorem 4.7.
Let $\mathbf{X}$ and $\mathbf{Y}$ be jointly Gaussian random vectors

$$\begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix} \sim N\left(\begin{bmatrix}\mu_X\\ \mu_Y\end{bmatrix}, \begin{bmatrix}A & C\\ C^\top & B\end{bmatrix}\right) = N\left(\begin{bmatrix}\mu_X\\ \mu_Y\end{bmatrix}, \begin{bmatrix}\bar{A} & \bar{C}\\ \bar{C}^\top & \bar{B}\end{bmatrix}^{-1}\right),$$

then the marginal distribution of $\mathbf{X}$ and the conditional distribution of $\mathbf{X}$ given $\mathbf{Y}$ are

$$\mathbf{X} \sim N(\mu_{\mathbf{X}}, A) \quad \text{and} \quad \mathbf{X} \mid \mathbf{Y} \sim N\left(\mu_X + CB^{-1}(\mathbf{Y} - \mu_Y),\ A - CB^{-1}C^\top\right),$$
$$\text{or} \qquad \mathbf{X} \mid \mathbf{Y} \sim N\left(\mu_X - \bar{A}^{-1}\bar{C}(\mathbf{Y} - \mu_Y),\ \bar{A}^{-1}\right).$$

Definition 4.8 (Finite-Dimensional Distribution).
Let $(X_t)_{t\in T}$ be a stochastic process with values in $\mathbb{R}$. For $\{t_1, ..., t_n\} \subseteq T$ formula

$$\nu_{t_1,...,t_n}(A) := \mathbb{P}((X_{t_1}, ..., X_{t_n}) \in A),$$

where $A \subseteq \mathbb{R}^n$, defines the *finite-dimensional distribution* of the process corresponding to $t_1, ..., t_n$.

Definition 4.9 (Gaussian Process).
If $T \neq \emptyset$, then stochastic process $(X_t)_{t\in T}$ is a *Gaussian Process*, when all its finite-dimensional distributions are Gaussian.

Gaussian Process is an example of L^2-process, which is defined as stochastic process $(X_t)_{t\in T}$ if for all $t \in T$: $\mathbb{E}[X_t^2] < \infty$.

Definition 4.10 (Mean function of L^2-process).
A mean function of L^2-process $(X_t)_{t\in T}$ is a function $m : T \to \mathbb{R}, m(t) = \mathbb{E}[X_t]$.

Definition 4.11 (Covariance function of L^2-process).
A covariance function of L^2-process $(X_t)_{t\in T}$ is a function $K : T \times T \to \mathbb{R}, K(s,t) = Cov(X_s, X_t)$

Theorem 4.12.
Let $T \neq \emptyset$ be an arbitrary set. For every function $m : T \to \mathbb{R}$ and every symmetric, non-negative definite function $K : T \times T \to \mathbb{R}$, there exists a Gaussian Process $(X_t)_{t\in T}$ such that for all $s, t \in T$:

$$E[X_t] = m(t), \quad Cov(X_s, X_t) = K(s,t)$$

This process is determined with precision up to finite-dimensional distributions.

When making predictions at new input points using Gaussian Processes, Theorem 4.13 provides the posterior predictive distribution, which combines noisy training observations and prior information to estimate the latent function value at any test location. Theorem 4.13 is based on Section 2.2 and Section 2.7 in [Rasmussen and Williams, 2006]

Theorem 4.13.
Let $(X_t)_{t\in T}$ be a Gaussian Process with mean function $m : T \to \mathbb{R}$ and covariance function $K : T \times T \to \mathbb{R}$, i.e. $X \sim \mathcal{GP}(m, K)$.

Suppose we have access to noisy observations of the form $y_i = X_{t_i} + \varepsilon_i$, where $t_i \in T$, and the noise terms ε_i are independent and identically distributed as $N(0, \sigma^2)$, for $i = 1, \ldots, n$. Let $\mathbf{y} = (y_1, \ldots, y_n)^T$, and define the vectors:

$$\mathbf{m} = (m(t_1), \ldots, m(t_n))^T, \quad K_* = (K(t_1, t_*), \ldots, K(t_n, t_*))^T,$$

and the matrix:

$$K_n = [K(t_i, t_j)]_{i,j=1}^n,$$

where $t_* \in T$ is an arbitrary test point at which we wish to predict the function value.

Then, the conditional distribution of X_{t_*}, given the observations $\mathbf{y}$, is Gaussian:

$$X_{t_*} \mid \mathbf{y} \sim N(\mu_*, \sigma_*^2),$$

where

$$\mu_* = m(t_*) + K_*^T (K_n + \sigma^2 I)^{-1} (\mathbf{y} - \mathbf{m}),$$

$$\sigma_*^2 = K(t_*, t_*) - K_*^T (K_n + \sigma^2 I)^{-1} K_*.$$

4.2. Bayesian Optimization Workflow

Bayesian Optimization (BO) is a sequential design strategy for global optimization of black-box functions that are expensive to evaluate. The method is particularly useful in scenarios where each evaluation of the objective function is costly in terms of time or computational resources. This section outlines the step-by-step procedure followed during the BO process.

1. **Initialization**

   The process begins by selecting an initial set of input points within the domain of interest. These points are often chosen using a random sampling method or a space-filling design strategy. The objective function is then evaluated at these initial points, providing the first observations that will serve as training data for the surrogate model.

2. **Surrogate model construction**

   Once initial evaluations are collected, a surrogate model is employed to approximate the true objective function. This model serves as a computationally efficient substitute, allowing the algorithm to estimate how the objective function behaves across the input space without requiring costly evaluations at every step. At this stage, the surrogate does not need to be perfect - rather, its role is to capture the general trends and uncertainties that inform the decision-making process in subsequent steps.

3. **Acquisition function optimization**

   The surrogate model is then used to construct an acquisition function, which acts as a decision-making tool to determine the next most promising point to evaluate. The acquisition function balances two competing goals: sampling areas with high predicted performance (so called exploitation) and exploring uncertain regions where the model has limited information (so called exploration). Instead of evaluating the true objective function everywhere, the optimization process focuses only on points suggested by the acquisition function, thus making the search more efficient.

4. **Selection of the next evaluation point**

   The input point that maximizes the acquisition function (or minimizes - depends on the description of the specific acquisition function) is selected as the next evaluation point. This point is expected to provide useful information that will improve the surrogate model and, eventually, lead to the global optimum of the objective function.

5. **Objective function evaluation**

   The actual objective function is evaluated at the selected point. Although this evaluation may be computationally expensive, it provides an accurate function value that is added to the existing dataset.

6. **Surrogate model update**

   The new observation is incorporated into the surrogate model. The model is retrained with the updated dataset, improving its accuracy and predictive uncertainty for the next iteration.

7. **Termination**

   The iterative process continues until a stopping criterion is met. Common termination conditions include reaching a maximum number of iterations, exceeding a time budget, or observing no significant improvement over a number of iterations. The best solution found during the optimization is returned as the final result.

4.3. Surrogate models

Surrogate models are used to replace expensive, time-consuming simulations or calculations while preserving key behaviors.

Definition 4.14 (Surrogate model).

A *surrogate model* is a simplified, computationally efficient approximation of a complex model. Let $f : \mathbb{R}^d \to \mathbb{R}$ denote an expensive-to-evaluate function, representing a process, where $\mathbf{x} \in \mathbb{R}^d$ is a vector of input parameters. Surrogate model constructs an approximation $\hat{f}(\mathbf{x}) \approx f(\mathbf{x})$ that can be evaluated with minimal computational cost.

Surrogate models are trained on input-output data from the original system, allowing for faster predictions, sensitivity analysis, or optimization — for example, by approximating a computationally expensive function.

In the context of Bayesian Optimization, output of the surrogate model serves as approximations of the true objective function $f(\mathbf{x})$, which is typically black-box function. The optimization procedure relies on a surrogate to make informed decisions about where to sample next, using an acquisition function (see Section 4.4).

In the context of surrogate models, x typically represents the input argument to the function being approximated - in our case, this would be a hyperparameter or a vector of hyperparameters (e.g., the number of trees in an ensemble model, learning rate, etc.).

Among the most commonly used surrogate models are Gaussian Processes, Random Forests and Extra Trees. Both Random Forests and Extra Trees are particular types of Tree Parzen Estimators. A key advantage of these models is their ability to quantify prediction uncertainty, which helps guide decision-making and improve model reliability, especially in high-stakes or data-scarce scenarios.

4.3.1. Gaussian Process as surrogate model

A Gaussian Process (GP) defines a flexible prior over functions, making it particularly useful as a surrogate model in Bayesian Optimization. The GP is specified by:

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), K(\mathbf{x}, \mathbf{x}')),$$

where $\mu(\mathbf{x})$ is the mean function and $K(\mathbf{x}, \mathbf{x}')$ is the covariance kernel function. These functions define a distribution over the unknown function f and encode prior beliefs about its properties, such as smoothness, variability, and expected values across the input space.

As discussed in Section 4.1 and in Theorem 4.13, given observed data $\mathcal{H}_n = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^n$, we can compute the posterior distribution at any new point $\mathbf{x}^*$:

$$\hat{f}_n(\mathbf{x}^*) \sim N(\mu_n(\mathbf{x}^*), \sigma_n^2(\mathbf{x}^*)).$$

The posterior mean $\mu_n(\mathbf{x}^*)$ and variance $\sigma_n^2(\mathbf{x}^*)$ are derived analytically from the kernel matrix K_n (which is the covariance matrix constructed by evaluating the kernel function K at all pairs

of observed input points: $K_n = [K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n$) and the observed data as in Theorem 4.13. The mean represents our best estimate of the function value, while the variance quantifies the uncertainty in our prediction.

The covariance kernel K is crucial, as it determines the smoothness and properties of the functions we can model.

Gaussian Processes offer key advantages as surrogate models, including natural uncertainty quantification expressed through the covariance function, analytical posterior computation, and modeling flexibility enabled by kernel functions that encode assumptions about smoothness, periodicity, or other structural properties of the objective function. Their ability to incorporate prior knowledge via kernel selection makes them particularly valuable for Bayesian Optimization.

However, GPs also present notable limitations in the context of Bayesian Optimization. The most significant drawback is their poor scalability due to $\mathcal{O}(n^3)$ computational complexity. Additionally, standard GP implementations are inherently limited to continuous parameter spaces, making them unsuitable for problems involving categorical or discrete variables without special modifications. The performance of GPs is also highly sensitive to the choice of kernel function and its hyperparameters, which often require careful tuning.

Despite these limitations, GPs remain particularly effective for Bayesian Optimization in low-to-moderate dimensional continuous spaces, where their strengths in uncertainty-aware modeling are most valuable.

4.3.2. Random Forest as surrogate model

A Random Forest (RF) is an ensemble of decision trees trained via bootstrap aggregation (see Subsection 3.4.1), making it a robust and flexible non-parametric model. As a surrogate model in Bayesian Optimization, RF approximates the unknown objective function by averaging predictions from individual trees:

$$\hat{f}_n(\mathbf{x}^*) = \frac{1}{T} \sum_{t=1}^{T} f_t(\mathbf{x}^*),$$

where T is the number of trees and $f_t(\cdot)$ denotes the prediction of tree t. To estimate uncertainty, the empirical variance across trees is used:

$$\sigma_n^2(\mathbf{x}^*) = \frac{1}{T-1} \sum_{t=1}^{T} (f_t(\mathbf{x}^*) - \hat{f}_n(\mathbf{x}^*))^2.$$

Although this uncertainty estimate is heuristic, it often performs well in practice for exploration-exploitation trade-offs.

RF models are particularly suitable for high-dimensional problems, scale linearly with the number of data points, and naturally handle both continuous and categorical variables. They are less sensitive to hyperparameters and robust to overfitting.

While Random Forests offer scalability and robustness, they lack the principled uncertainty quantification of Gaussian Processes and may be less effective for smooth, low-dimensional functions.

4.3.3. Extra Trees as surrogate model

Extra Trees (ET, Extremely Randomized Trees) are an ensemble method similar to Random Forests but introduce additional randomness during training. Instead of searching for optimal split points, ET selects split thresholds at random, further decorrelating the trees and often improving generalization.

As a surrogate model, ET approximates the unknown function via the average prediction of T randomized trees:

$$\hat{f}_n(\mathbf{x}^*) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}^*),$$

with predictive uncertainty estimated through the variance across tree outputs:

$$\sigma_n^2(\mathbf{x}^*) = \frac{1}{T-1}\sum_{t=1}^{T}(f_t(\mathbf{x}^*) - \hat{f}_n(\mathbf{x}^*))^2.$$

4.4. Acquisition functions

This chapter is based on chapter *5. Decision theory for optimization* from [Garnett, 2023]

Acquisition function is a mathematical tool central to Bayesian Optimization, used to determine the next point to evaluate when optimizing an expensive objective function. It balances exploration (sampling uncertain regions) and exploitation (focusing on promising areas) by assigning a "score" to potential observation points. This decision-making process is guided by a surrogate model that approximates the true objective function based on past observations.

Definition 4.15 (Acquisition function).
Denoted as $a : \mathcal{X} \times \mathcal{H} \to \mathbb{R}$, *the acquisition function* maps each point x in the domain $\mathcal{X}$ to a real number, reflecting its utility for optimization. This score depends on optimization history $\mathcal{H}$.

Remark 4.16.
Preferences for candidate points are shaped by the posterior belief $p(f|\mathcal{H})$, a probabilistic model

of the objective function f conditioned on observed data. The acquisition function leverages this posterior to quantify uncertainty and expected improvement.

The next evaluation point is selected by maximizing (or - for, among others, Lower Confidence Bound - minimizing) the acquisition function:

$$x^* \in \text{argmax}_{x' \in \mathcal{X}} \; a(x', \mathcal{H}).$$

Though this is a global optimization problem, acquisition functions are designed to be computationally cheap and analytically differentiable, enabling efficient optimization with standard tools.

Bayesian Optimization appears paradoxical: it solves a hard global optimization problem by repeatedly solving simpler global optimization subproblems (maximizing $a(x, \mathcal{H})$. The key is that acquisition functions are far cheaper to evaluate than the original objective function, making the approach practical.

4.4.1. Examples of acquisition functions

Prominent acquisition functions include Lower Confidence Bound, Expected Improvement and Probability of Improvement. They are included in package `scikit-optimize` in Python.

Remark 4.17.
Note that in the following subsections $a(x, \mathcal{H})$ is described as $a(x)$. In definition of acquisition function it is emphasized that the function depends on the data $\mathcal{H}$. In practice (e.g. in `scikit-optimize`) the surrogate model is already trained on data $\mathcal{H}$, so $\mu(x)$ and $\sigma(x)$, which are estimated by the surrogate model, take into account $\mathcal{H}$ implicitly.

The Lower Confidence Bound (LCB) is a commonly used acquisition function in Bayesian Optimization. Its purpose is to determine the next point for evaluation by balancing exploration and exploitation. It does so by penalizing the predicted mean by a multiple of the predicted standard deviation. The role of this acquisition function is to select the next evaluation point so as to minimize the objective function by selecting evaluation points with a potentially low true function value and/or high uncertainty.

Definition 4.18 (Lower Confidence Bound (LCB)).
The LCB acquisition function is formally defined as:

$$a_{\text{LCB}}(x) = \mu(x) - \kappa \cdot \sigma(x),$$

where:

- $\mu(x)$ is the surrogate model's predicted mean at x,
- $\sigma(x)$ is the surrogate model's predicted standard deviation at x,
- $\kappa \geqslant 0$ is a user-defined parameter controlling the exploration-exploitation trade-off.

Expected Improvement (EI) is another widely used acquisition function in Bayesian Optimization. Its goal is to choose the next point to evaluate by maximizing the expected improvement over the currently best observed function value. To compute this expectation EI uses both the predicted value and uncertainty of the surrogate model, and it is particularly useful when we want to balance taking advantage of known good areas and exploring uncertain regions that might yield better results. The function rewards points that are likely to outperform the current best value, while also considering the potential size of the improvement.

Definition 4.19 (Expected Improvement (EI)).
The EI acquisition function is formally defined as:

$$a_{\text{EI}} = \begin{cases} (f_{\min} - \mu(x) - \xi) \cdot \Phi(Z) + \sigma(x) \cdot \phi(Z) & \text{for } \sigma(x) > 0, \\ 0 & \text{for } \sigma(x) = 0, \end{cases},$$

where:

- $Z = \dfrac{f_{\min} - \mu(x) - \xi}{\sigma(x)}$,
- $\mu(x)$ is the surrogate model's predicted mean at x,
- $\sigma(x)$ is the surrogate model's predicted standard deviation (uncertainty) at x,
- $f_{\min}$ is the current best observed value of the objective function,
- $\xi \geqslant 0$ is an exploration threshold,
- Φ is the cumulative distribution function (CDF) of the standard normal distribution,
- ϕ is the probability density function (PDF) of the standard normal distribution.

Probability of Improvement (PI) is an acquisition function used in Bayesian Optimization that seeks to maximize the probability that the function value at a new point will outperform the current best observed value by a specified threshold. The PI function focuses on selecting points that are likely to result in an improvement over the current best value, considering the

uncertainty in the model's predictions. By maximizing PI, the optimizer aims to prioritize points that have a higher likelihood of providing a better outcome, based on the surrogate model's predictions.

Definition 4.20 (Probability of Improvement (PI)).
The PI acquisition function is formally defined as:

$$a_{\mathrm{PI}} = \Phi\left(\frac{f_{\min} - \mu(x) - \xi}{\sigma(x)}\right),$$

where:

- $\mu(x)$ is the surrogate model's predicted mean at x,
- $\sigma(x)$ is the surrogate model's predicted standard deviation (uncertainty) at x,
- $f_{\min}$ is the current best observed value of the objective function,
- $\xi \geqslant 0$ is an exploration threshold,
- Φ is the CDF of the standard normal distribution, which converts the standardized improvement score into a probability.

5. Experiment

In order to better understand the practical implications of Bayesian Optimization, an experiment was conducted to investigate how its individual components affect the balance between exploration and exploitation during hyperparameter tuning. The experiment was implemented in Python using libraries such as `scikit-optimize` [Head et al., 2021] (commonly reffered to as `skopt`) and `scikit-learn` [Pedregosa et al., 2011]. The code for this experiment is available in the repository [1]. Particular attention was given to comparing the Bayesian Optimization algorithm (called Bayesian Search) with more traditional approaches like Grid Search and Random Search. The aim was to evaluate the extent to which different elements of Bayesian Optimization - including the surrogate model and acquisition function - influence the algorithm's behavior in navigating the hyperparameter space.

5.1. Datasets and cross-validation

In order to evaluate the generalizability of the findings, the optimization methods were tested on a diverse set of datasets. The datasets were selected from the OpenML CC18 benchmark [Bischl et al., 2019] suite (a standardized collection commonly used for fair and reproducible comparison of ML algorithms). The chosen subset covers a range of domains and data complexities, particularly in terms of feature dimensionality and class distribution, while keeping the number of observations in a relatively similar scale to ensure computational feasibility, but also making it suitable for studying the balance between exploration and exploitation under different conditions.

Table 5.1 summarizes the datasets used in the experiment, including their OpenML ID, number of observations, and number of features.

[1] `https://github.com/montaluta/Bayesian-Optimization-Bachelor-Thesis`

Table 5.1: Summary of datasets used in the experiment.

ID	Name	Observations	Features
23381	dresses-sales	500	12
1063	kc2	522	21
6332	cylinder-bands	540	37
40994	climate-model-simulation-crashes	540	18
1510	wdbc	569	30
1480	ilpd	583	10
29	credit-approval	690	15
15	breast-w	699	9
1464	blood-transfusion-service-center	748	4
37	diabetes	768	8
50	tic-tac-toe	958	9
1494	qsar-biodeg	1055	41
1068	pc1	1109	21
1462	banknote-authentication	1372	4
1049	pc4	1458	37
1050	pc3	1563	37
1067	kc1	2109	21
1487	ozone-level-8hr	2534	72
1485	madelon	2600	500
3	kr-vs-kp	3196	36

5.2. Validation of machine learning models

To better evaluate the model, prevent overfitting and optimize the selection of HPs, we used 5-fold cross-validation in the experiment.

Definition 5.1 (k-fold cross-validation).
k-fold cross-validation is a validation technique where the original dataset is randomly divided into k equal, pairwise disjoint parts (folds). The model is trained on $k-1$ folds and validated on the remaining one. This process is repeated k times, so that each fold serves as the validation set once, providing a reliable estimate of the model's performance and reducing the risk of overfitting.

To provide a clearer illustration of the k-fold cross-validation process, Figure 5.1, which is taken from documentation [2], illustrates 5-fold cross-validation.

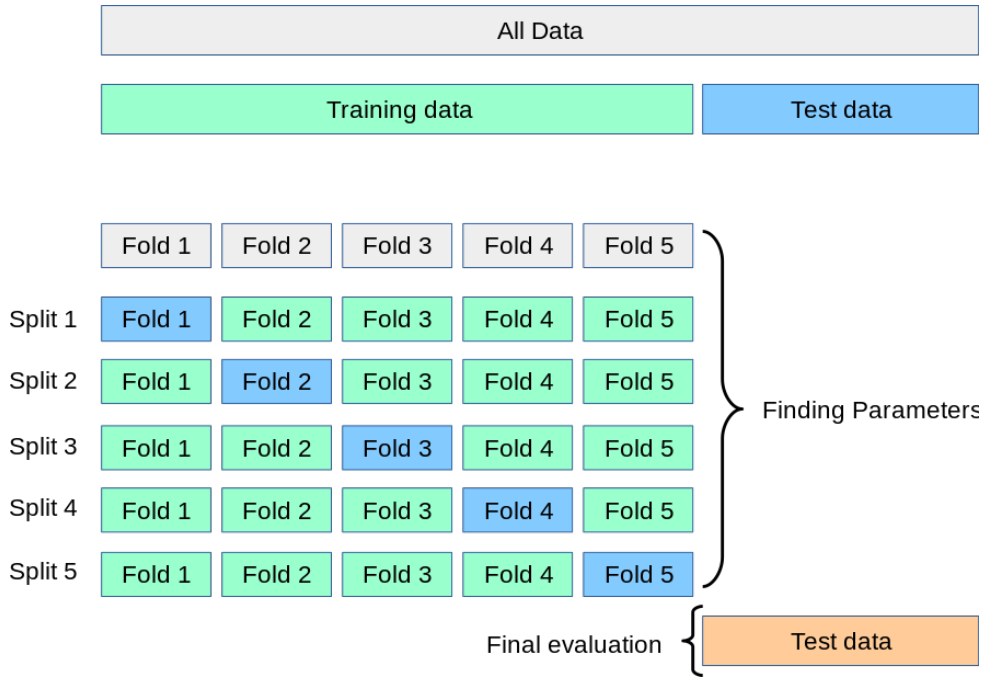

Figure 5.1: 5-fold cross-validation schema. The diagram illustrates the typical workflow of 5-fold cross-validation in model training. It begins with the entire dataset, which is divided into multiple folds. In each iteration, one fold (the blue one) is used as the test set, while the remaining folds (green) serve as the training set. This process is repeated for each fold, ensuring that every data point is used for training $k-1$ times and for testing exactly once. The performance metrics from each iteration are then averaged to provide a more reliable estimate of the model's performance. Note that the "Test data" shown in the schema are not part of the 5-fold cross-validation itself. They represent an optional, external test set that can be used for final model evaluation after k-fold cross-validation is complete.

Since AUC is used as the evaluation metric in our experiment, the following definitions briefy introduce the concepts of the Receiver Operating Characteristic Curve (ROC) and the AUC to clarify their interpretation and relevance in this context.

To construct the ROC curve, the True Positive Rate (TPR) and the False Positive Rate (FPR) are calculated at various classification thresholds. *The decision threshold* is the value that determines the cutoff point at which the model assigns a positive or negative label to a given instance. The TPR represents the proportion of actual positives correctly identified by the model ($\text{TPR} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$), while the FPR reflects the proportion of actual

[2] `https://scikit-learn.org/stable/modules/cross_validation.html`

negatives that are incorrectly classified as positives (FPR = $\frac{\text{False Positives}}{\text{False Positives+True Negatives}}$).

Definition 5.2 (Receiver Operating Characteristic (ROC) Curve).
The ROC Curve is a graphical representation that illustrates the diagnostic ability of a binary classifier system as its decision threshold is varied.
It plots the True Positive Rate against the False Positive Rate at various threshold settings.
A classifier that makes random guesses produces a ROC curve that lies along the diagonal (from bottom-left to top-right), whereas a model with perfect discrimination reaches the top-left corner of the plot.

Definition 5.3 (Area Under the Receiver Operating Characteristic Curve (AUC)).
The AUC quantifies how well the model can separate the positive and negative classes across all possible threshold values. It corresponds to the area under the ROC curve and ranges from 0 to 1.
An AUC of 0.5 indicates performance no better than random guessing, whereas an AUC of 1.0 signifies perfect classification. In practice, higher AUC values reflect better overall model performance and greater robustness to threshold selection.

5.3. Combinations of surrogate models, acquisition functions and hyperparameters

In order to evaluate the performance of Bayesian Optimization across different configurations, we investigate a wide range of combinations involving surrogate models, acquisition functions, and model-specific hyperparameters.

Surrogate models used in the experiment are:

- Gaussian Process (GP),
- Random Forest (RF),
- Extra Trees (ET),

which are described in detail in Section 4.3. Default hyperparameter settings provided by the implementation are used for all surrogate models.

Each surrogate model is paired with different acquisition functions, namely:

- Lower Confidence Bound (LCB),
- Expected Improvement (EI),
- Probability of Improvement (PI),

which are described in detail in Section 4.4. Default settings are also used for all acquisition functions.

To test the impact of search strategies, each base model is optimized using three different hyperparameter tuning approaches: Bayesian Search, Grid Search and Random Search.

We apply these methods to three ML models: XGBoost, LightGBM and Random Forest. Each of these models has its own set of tunable HPs, which are adapted for each optimization strategy.

Before we describe the search spaces, we need to describe the HPs used:

- `learning_rate` is the step size at each iteration while updating the model - used in XGBoost and LightGBM,
- `max_depth` is the maximum depth of a single decision tree - used in XGBoost and Random Forest,
- `n_estimators` is the number of trees in the ensemble model - used in XGBoost, LightGBM, and Random Forest,
- `subsample` is the fraction of samples used to train each individual tree - used in XGBoost and LightGBM,
- `reg_lambda` is the L2 regularization term on weights, which penalizes large coefficients to control model complexity - used in XGBoost and LightGBM,
- `num_leaves` is the maximum number of leaves in a single tree - used in LightGBM,
- `min_samples_split` is the minimum number of samples required to split an internal node - used in Random Forest,
- `min_samples_leaf` is the minimum number of samples required to be at a leaf node - used in Random Forest,
- `eval_metric` is a metric used to evaluate model performance during training, e.g., `logloss` for binary classification - used in XGBoost,

- `force_col_wise` is used in LightGBM, forces the use of column-wise data format, which can improve efficiency in certain cases - used in LightGBM,

- `min_gain_to_split` is the minimum gain needed to perform a split in a tree node - used in LightGBM.

We define search spaces as in Tables 5.2, 5.3, 5.4. Additionally, we assume the following default settings for the models:
- for XGBoost `eval_metric = 'logloss'`,
- for LightGBM `force_col_wise = True` and `min_gain_to_split = 0.01`.

Table 5.2: Hyperparameter search space for XGBoost.

XGBoost					
Search	**learning_rate**	**max_depth**	**n_estimators**	**subsample**	**reg_lambda**
Bayesian	[0.01, 0.3] (log)	[3, 9]	[50, 300]	[0.6, 1.0]	[0, 10]
Random	[0.01, 0.8] (log)	[3, 10]	[50, 200]	[0.6, 1.0]	[0, 10]
Grid	{0.01, 0.1, 0.2}	{3, 6, 9}	{100, 200}	{0.6, 0.8, 1.0}	{0, 5, 10}

Table 5.3: Hyperparameter search space for LightGBM.

LightGBM					
Search	**learning_rate**	**num_leaves**	**n_estimators**	**subsample**	**reg_lambda**
Bayesian	[0.01, 0.3] (log)	[31, 100]	[50, 300]	[0.6, 1.0]	[0, 10]
Random	[0.01, 0.8] (log)	[31, 100]	[50, 300]	[0.6, 1.0]	[0, 10]
Grid	{0.01, 0.1, 0.3}	{31, 50, 100}	{100, 200, 300}	{0.6, 0.8, 1.0}	{0, 5, 10}

Table 5.4: Hyperparameter search space for Random Forest.

Random Forest				
Search	**n_estimators**	**max_depth**	**min_samples_split**	**min_samples_leaf**
Bayesian	[50, 300]	[3, 20]	[2, 20]	[1, 20]
Random	[50, 300]	[3, 20]	[2, 20]	[1, 20]
Grid	{100, 200, 300}	{3, 6, 9, None}	{2, 5, 10}	{1, 2, 4}

5.4. Main loop of the experiment

The main loop of the experiment involves identifying the best-performing hyperparameter configuration for each dataset listed in Table 5.1 and for each optimization method.

Specifically, for every dataset, the hyperparameter search is conducted separately using three optimization strategies: `BayesSearchCV`, `GridSearchCV`, and `RandomizedSearchCV`. Within each method, 5-fold cross-validation is applied to estimate model performance in a robust and comparable way. The evaluation metric used throughout all experiments is AUC.

For `BayesSearchCV` and `RandomizedSearchCV`, the search is limited to a maximum of 100 iterations to control computational cost, while `GridSearchCV` exhaustively explores all parameter combinations in the specified grid, resulting in 162 iterations for XGBoost, 243 iterations for LightGBM, and 108 iterations for Random Forest.

For every combination of model, dataset, and optimization method, the configuration achieving the highest mean cross-validated AUC (see Section 5.2) is selected as the best. This experimental design enables a comprehensive comparison of how different optimization methods perform across models and datasets, particularly in terms of their ability to balance exploration and exploitation in hyperparameter tuning.

5.5. Analysis of method convergence plots

To illustrate the results of the conducted experiment, convergence plots were created for each combination of dataset and ML model. These plots show how the performance of different hyperparameter optimization methods evolves as the number of iterations increases. They provide a visual summary of both the efficiency and stability of each method across the search process.

5.5.1. Convergence of optimization methods

Due to the large number of possible model-dataset combinations, including all plots in the thesis would be impractical and could negatively impact clarity and readability. Therefore, only a selected subset of plots is included for illustration. The chosen examples highlight the most representative or distinctive behaviors observed during the experiment.

The x-axis indicates the iteration number (`n_iter`), while the y-axis shows the mean AUC score from cross-validation obtained during each step of the optimization process. Each line represents a distinct optimization method, with visual differences such as color and marker style to facilitate comparison. To additionally highlight the stability and variability of results, shaded

regions around each line represent the standard deviation (`std`) of the scores. This allows not only for evaluating average performance but also understanding how consistent each method is across repeated evaluations.

It is important to note that for the Bayesian Optimization and for Random Search the search process was constrained by an iteration limit of 100, but Grid Search has more iterations. Also note that the iterations start from 1, even though the x-axes on the plots are signed from 0.

The dataset with ID 37 was selected for detailed analysis of the corresponding plots. Figure 5.2 shows that for XGBoost, Grid Search exhaustively explores the hyperparameter space, and in this case, it required about 160 iterations to find the optimal configuration. Although Random Search is faster than Grid Search, its random sampling introduces fluctuations in the results, making it less predictable, as indicated by the higher standard deviation. In contrast, Bayesian methods leverage probabilistic models to intelligently select promising hyperparameter combinations, resulting in rapid convergence - they achieved high AUC scores within just 30–50 iterations.

Figure 5.3 illustrates that, for LightGBM Grid Search also slowly explores the HP space, it required about 240 iterations to find the optimal configuration. Random Search has unpredictable results. Bayesian methods achieve high AUC scores rapidly again, but are less stable that Grid Search.

Figure 5.4 conveys that for Random Forest regardless of the method, the level of stability of the results is similar, as are the results themselves. Grid Search needs about 110 iterations to converge. Random Search is not predictable in terms of when it will reach a high AUC score. Bayesian methods, on the other hand, have high results even after about 20 iterations.

Conclusions. Bayesian methods, particularly those using Gaussian Processes with LCB acquisition (`Bayes_GP_LCB`), achieve high AUC scores significantly faster - often within 20–50 iterations - whereas Grid Search requires a substantially higher number of iterations, and Random Search results remain highly variable and unpredictable.

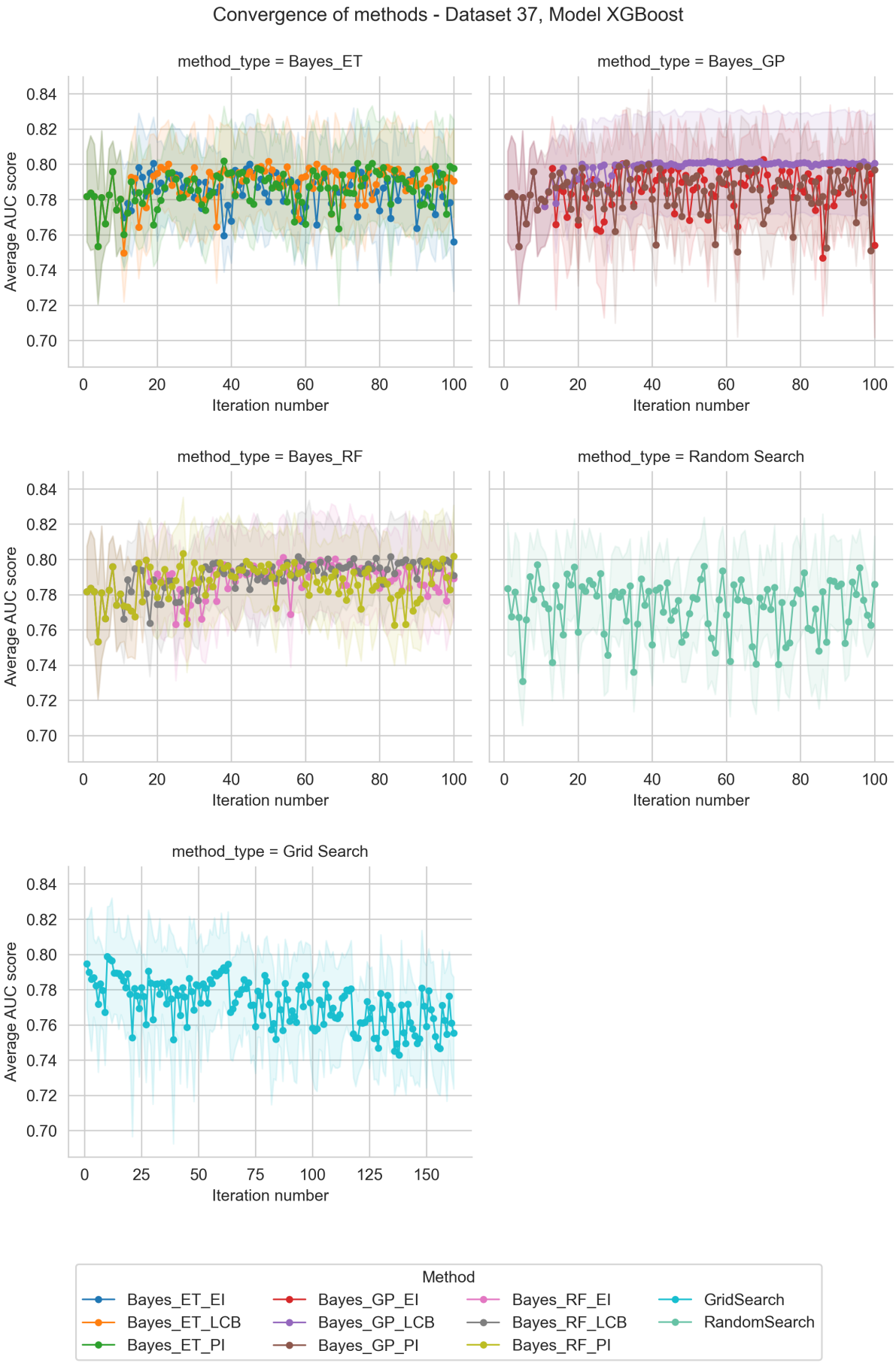

Figure 5.2: Convergence of methods for XGBoost model on dataset with ID 37.

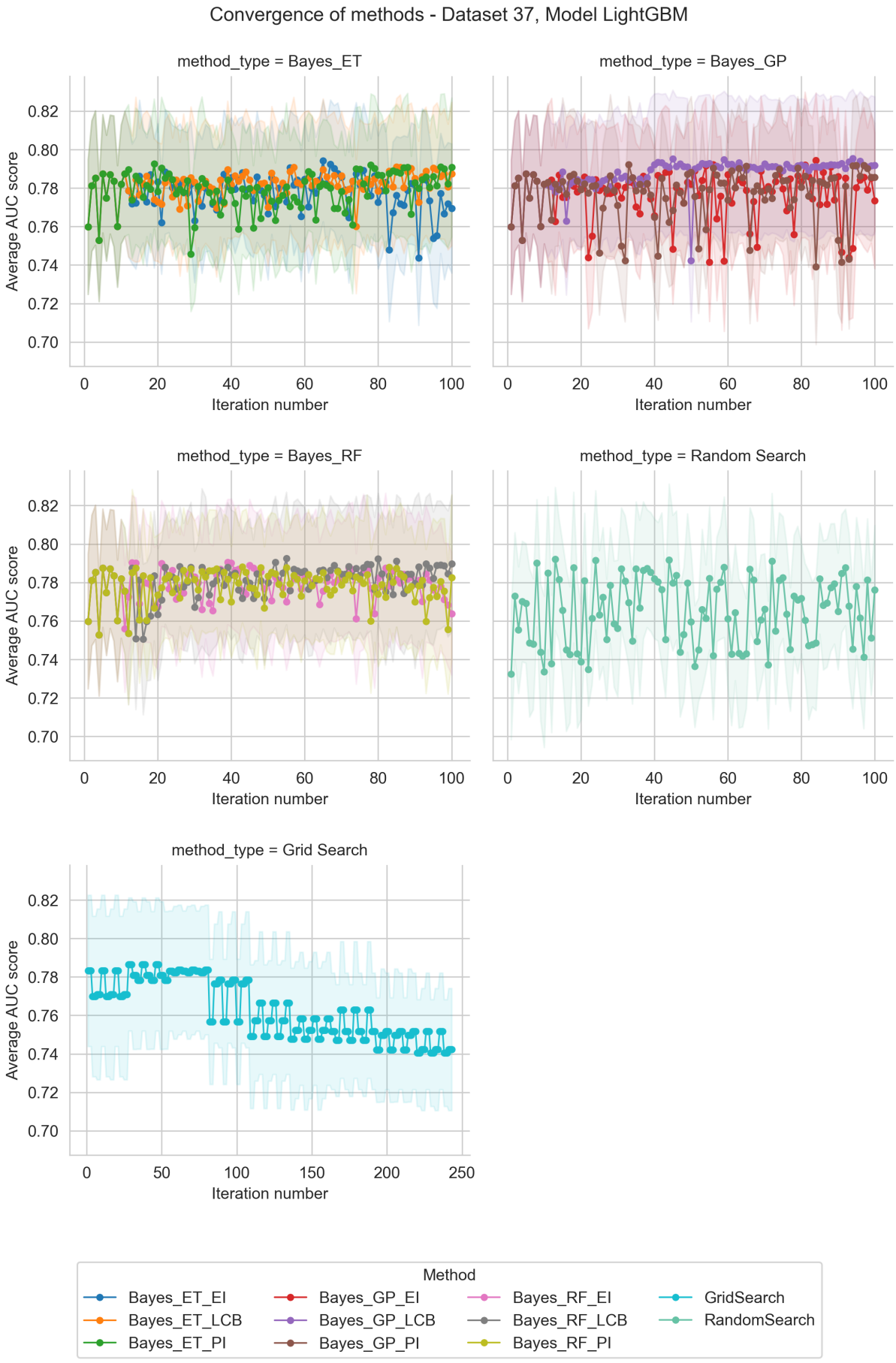

Figure 5.3: Convergence of methods for LightGBM model on dataset with ID 37.

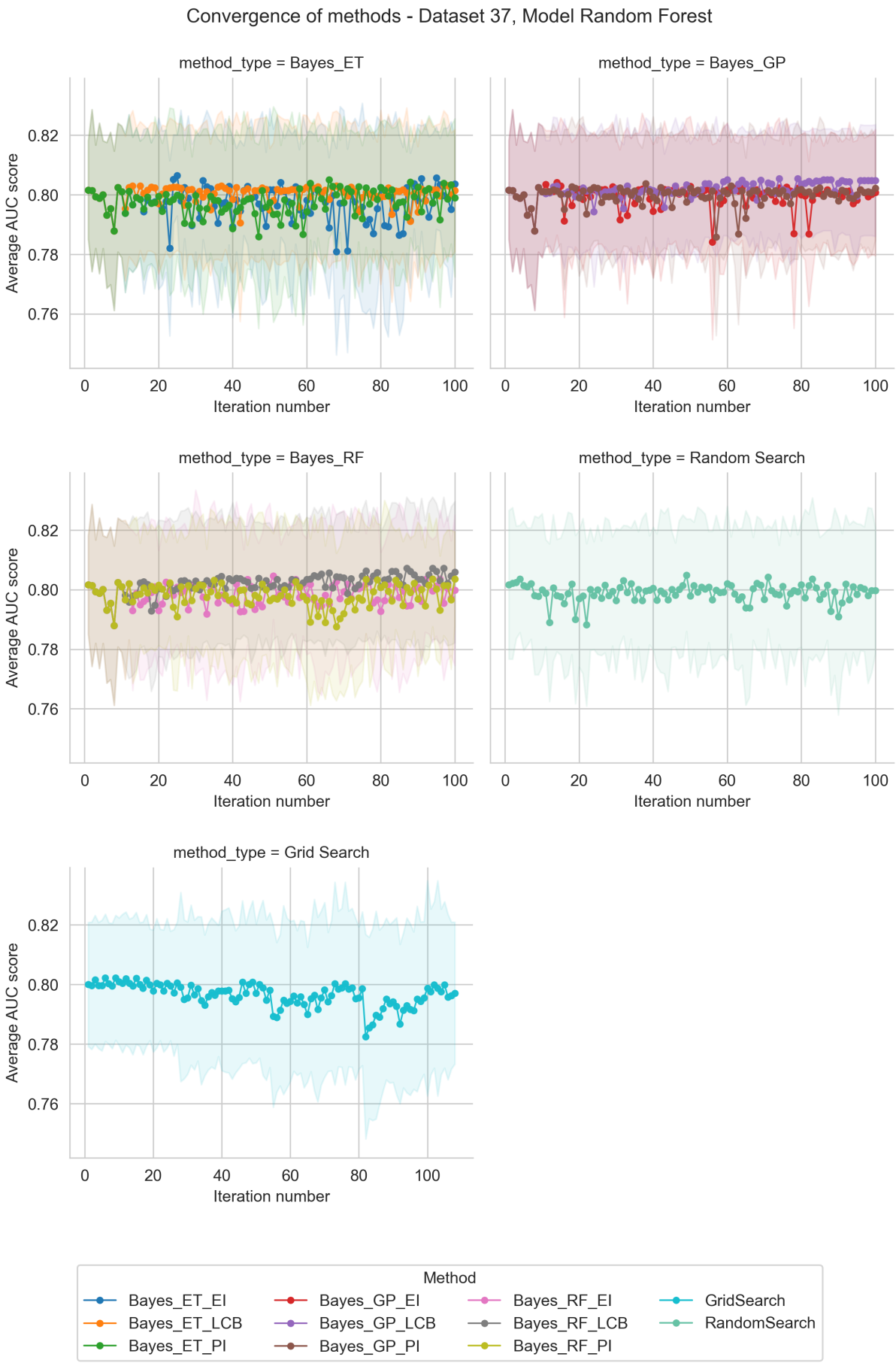

Figure 5.4: Convergence of methods for Random Forest model on dataset with ID 37.

5.5.2. Aggregating Convergence Results using Average Distance to Maximum

In order to draw general conclusions from the experiment, it is essential to aggregate the results across all datasets. By combining the data in this way, we can identify overarching patterns and trends that would otherwise be difficult to discern when examining individual cases. This aggregation allows us to summarize the performance of different optimization methods in a way that is both comprehensive and interpretable. It ensures that the findings are not biased by the specifics of any single dataset, but rather reflect the overall effectiveness and behavior of the methods across various scenarios.

First, we need to introduce the concept of Average Distance to Maximum (ADTM), which serves as a metric for evaluating the effectiveness of hyperparameter tuning strategies across a set of meta-test datasets.

Definition 5.4 is based on [Woźnica et al., 2024].

Definition 5.4 (Average Distance to Maximum (ADTM)).
Let $\mathbf{D}_{meta-test}$ be a set of all test datasets on which the effectiveness of the hyperparameter tuning strategy is evaluated, and f is AUC measure. The set of hyperparameter configurations generated by a given strategy until `n_iter = N` is Λ_N. Then *ADTM (Average Distance to Maximum)* is defined as

$$ADTM(\mathbf{D}_{meta-test}, \Lambda_N) = \frac{1}{|\mathbf{D}_{meta-test}|} \sum_{i \in \mathbf{D}_{meta-test}} \min_{\lambda \in \Lambda_N} \frac{f_i^{max} - f_i(\lambda)}{f_i^{max} - f_i^{min}},$$

where $|\mathbf{D}_{meta-test}|$ is a cardinality of $\mathbf{D}_{meta-test}$.

To evaluate the performance of the analyzed hyperparameter optimization methods, the ADTM metric was computed. The calculation was performed for each method across all available meta-test tasks, where each task is defined as a unique dataset.

To visualize the results, the plots were generated showing how the ADTM score changes over iterations for each method. The x-axis represents the number of iterations, while the y-axis shows the ADTM value - a lower value indicates better performance. Each method is represented by a separate line, allowing for a direct comparison of their convergence behavior and overall effectiveness. Note that the iterations start from 1, even though the x-axes on the plots are signed from 0.

It is also worth noting that in the initial iterations, the results of the Bayesian methods in ADTM are identical, causing their lines on the plot to completely overlap. This behavior is due to the fixed random seed and identical initial conditions, which lead these algorithms to follow the exact same sequence of decisions at the beginning. Only in subsequent iterations, when

the methods start to differ more distinctly in effectiveness, can the individual behavior of each algorithm be clearly observed.

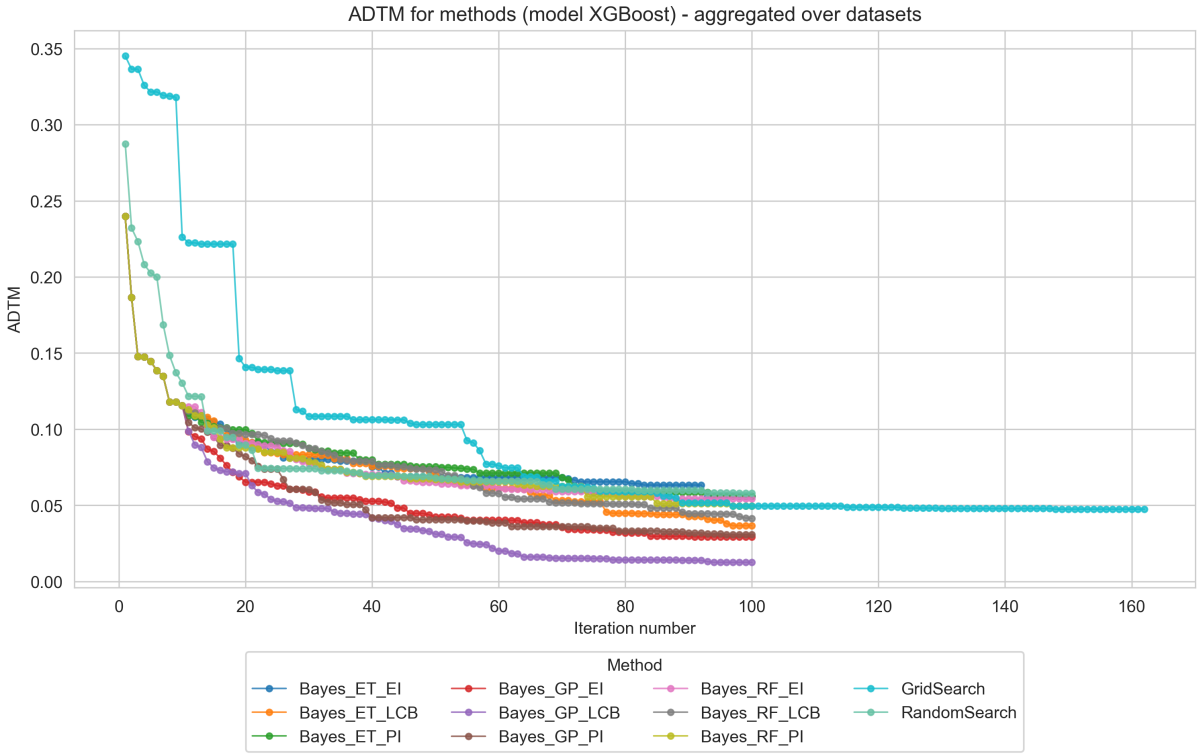

Figure 5.5: ADTM for each method for XGBoost model.

Figure 5.5 clearly shows that for XGBoost the ADTM decreases consistently as the number of iterations increases. This indicates that as the optimization process proceeds, the model parameters are tuned more effectively, leading to an improvement in performance. Notably, a rapid improvement is observed during the early iterations, after which the improvements slow down and the performance remains relatively stable, showing similar results over the iterations. This stabilization signifies that the optimization process approaches convergence once marginal improvements become minimal. It is evident that most Bayesian Optimization methods reach lower ADTM values faster than Grid Search and Random Search. This suggests that Bayesian methods, which intelligently explore and exploit the search space using probabilistic modeling, are more efficient and yield better performance results compared to the less targeted strategies used by Grid Search and Random Search. Multiple lines for Bayesian approaches indicates subtle differences in performance based on the choice of surrogate model and acquisition function. These differences imply that while Bayesian Optimization, in general, outperforms traditional methods. In this case `Bayes_GP_LCB` is the best in terms of ADTM. By aggregating results over multiple datasets, the plot suggests that the advantages observed in Bayesian Optimization methods are

robust across various conditions. This reinforces the idea that employing advanced optimization strategies like Bayesian methods can significantly reduce the computational budget required for hyperparameter tuning while consistently achieving a lower ADTM (i.e., better performance).

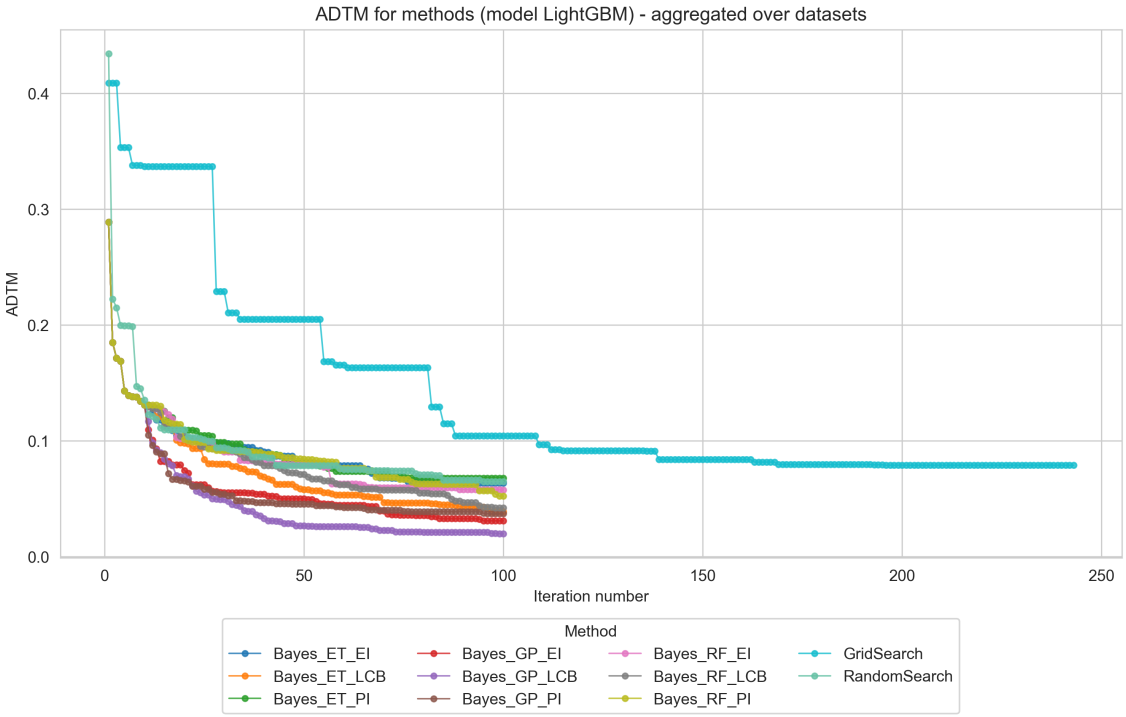

Figure 5.6: ADTM for each method for LightGBM model.

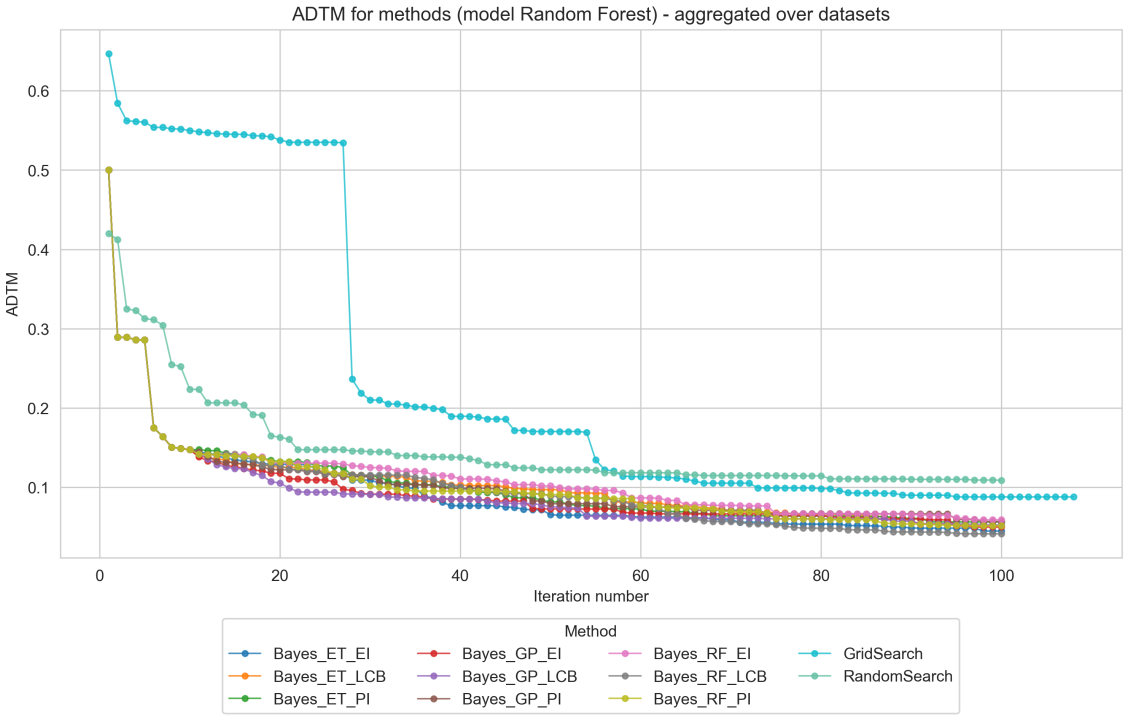

Figure 5.7: ADTM for each method for Random Forest model.

Conclusions drawn from Figure 5.6 are similar as from Figure 5.5. Also `Bayes_GP_LCB` is in this case the best.

Findings from Figure 5.7 are consistent with those from Figures 5.5 and 5.6, but here it is difficult to distinguish the best Bayesian method because the lines overlap.

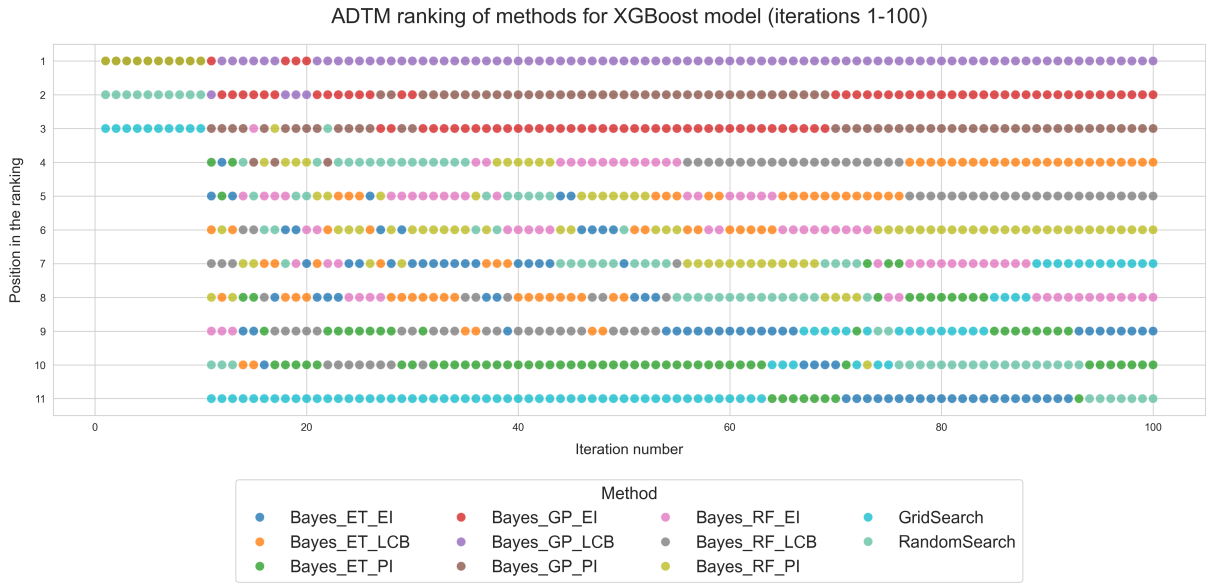

Figure 5.8: ADTM ranking for all methods over iterations 1-100 for XGBoost model.

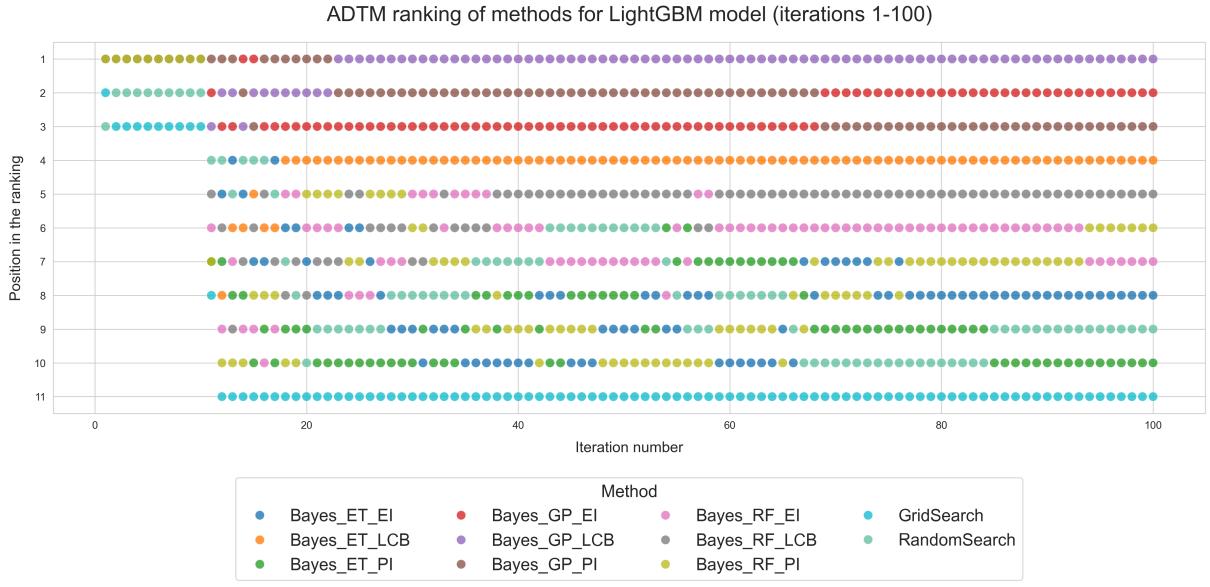

Figure 5.9: ADTM ranking for all methods over iterations 1-100 for LightGBM model.

In addition to visualizing the evolution of the ADTM metric over iterations, we also constructed a ranking of methods based on their ADTM values at each iteration step from iteration 1 to 100 (because there is no point in comparing above the hundredth iteration - only Grid

Search has such). This ranking provides a more interpretable comparison, showing which hyperparameter optimization strategies performed best at different stages of the optimization process. By analyzing these rankings, we can identify not only which methods converge faster but also which consistently achieve better performance throughout the tuning procedure. Note that the iterations start from 1, even though the x-axes on the plots are signed from 0.

Figure 5.8 confirms what Figure 5.5 showed us, but with more details. During the initial 10 iterations, the Bayesian methods produce identical results, leading to tied rankings and overlapping lines on the plot due to the fixed random seed and identical initial conditions. Grid Search consistently performs poorly, often occupying the lowest positions. Random Search demonstrates high variability and is frequently ranked near the bottom. In contrast, the `Bayes_GP_LCB` method clearly outperforms the others, maintaining the top position for the majority of the iterations. The `Bayes_GP_PI` and `Bayes_GP_EI` methods also show strong performance, typically ranking in second or third place.

Figure 5.9 shows that during the initial 10 iterations of LightGBM optimization, the Bayesian methods produce identical results like in Figure 5.8. Grid Search consistently ranks among the lowest-performing approaches. Random Search exhibits significant variability, generally fluctuating between the 8th and 10th positions. In contrast, the `Bayes_GP_LCB` method demonstrates clear superiority, consistently occupying the top rank throughout most iterations. The `Bayes_GP_PI` and `Bayes_GP_EI` methods also perform well, typically maintaining second or third place in the rankings.

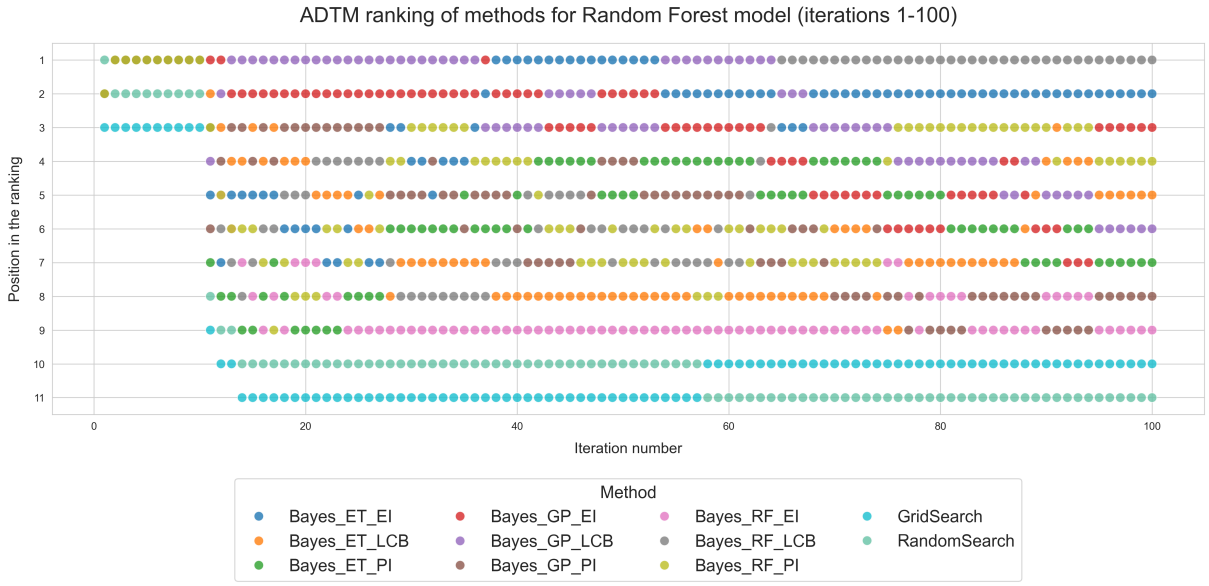

Figure 5.10: ADTM ranking for all methods over iterations 1-100 for Random Forest model.

Figure 5.10 illustrates that for the initial iterations (approximately the first 10–15), the performance of Bayesian methods is identical, consistent with the trends observed in Figures 5.8 and 5.9. As the number of iterations increases, the differences in performance become more pronounced. Grid Search consistently exhibits poor rankings throughout the optimization process. Random Search also performs poorly, with rankings fluctuating near the bottom. In contrast, `Bayes_GP_LCB` consistently achieves the highest ranking, frequently occupying the top position. Other Bayesian strategies, such as `Bayes_GP_EI` and `Bayes_GP_PI`, also perform strongly, and these two mentioned ranking within the top three methods.

Conclusions. The ADTM plots confirm that Bayesian Optimization reduces model tuning time while improving performance. `Bayes_GP_LCB` consistently achieves the lowest ADTM values for XGBoost and LightGBM, indicating that it is the most effective strategy in this context. For Random Forest, although the Bayesian advantage is less pronounced, these methods still perform competitively.

Ranking plots confirm that Bayesian methods dominate the top positions across 100 iterations. `Bayes_GP_LCB` emerges as the most robust performer, followed closely by `Bayes_GP_EI` and `Bayes_GP_PI`. Grid Search and Random Search remain consistently among the lowest-ranked strategies.

5.6. Comparison of Bayesian methods using Observation Traveling Salesman Distance

In this section, we analyze how different Bayesian methods explore the search space over time. To do this, we use the Observation Traveling Salesman Distance (OTSD) [Papenmeier et al., 2025] - a measure that reflects how directly an algorithm moves through the space of configurations. Intuitively, lower values of OTSD suggest that the method explores in a more structured way, while higher values may indicate redundant or scattered movement.

For comparison, we also include Random Search, which samples configurations uniformly at random. This allows us to benchmark the exploration behavior of Bayesian methods against a baseline that does not use any adaptive search strategy.

By tracking how OTSD evolves with each optimization step, we can observe patterns in the search behavior of different algorithms. Averaging results across multiple datasets allows us to compare searchers more robustly and draw conclusions about their overall efficiency.

Before analyzing the results, we first introduce the concept of OTSD, which serves as the basis for evaluating how efficiently search algorithms explore the hyperparameter space.

Definition 5.5 is based on [Papenmeier et al., 2025].

Definition 5.5 (Observation Traveling Salesman Distance (OTSD)).
Observation Traveling Salesman Distance (OTSD) measures the minimal total Euclidean distance required to traverse a set of observation points exactly once, formulated as a Traveling Salesman Problem. Given a sequence of observations $X_t = \{x_t\}_{i=1}^{t}$ in a d-dimensional space, OTSD is defined as:

$$OTSD(X_t) := \min_{\tau \in S_t} \left(\sum_{i=1}^{t} \|x_{\tau(i)} - x_{\tau(i+1)}\| \right),$$

where τ is a permutation of $\{1, 2, ..., t\}$ representing a tour (with $\tau(t+1) = \tau(1)$), S_t is the set of all possible permutations of t elements and $\|\cdot\|$ denotes the Euclidean distance (so for $x = (x_1, ..., x_d)^T \in R^d : \|x\| = \sqrt{\sum_{i=1}^{d} x_i^2}$).

To ensure comparability across different hyperparameter spaces and scales, all hyperparameter configurations evaluated during the optimization process were first normalized using min-max scaling. This step standardizes the range of each hyperparameter to the interval $[0, 1]$, preventing any single parameter from dominating the distance calculations due to its numerical scale.

After normalization, we computed the OTSD for each sequence of evaluated configurations, capturing how efficiently methods explore the configuration space. This allowed us to track and compare the exploration behavior of different Bayesian methods across iterations and datasets in a consistent and scale-invariant manner. To visualize differences, we created a series of plots that show the OTSD values over optimization steps.

Note that the OTSD is monotonically non-decreasing as more points are added. Since finding the exact solution to the Traveling Salesman Problem is computationally expensive for larger sets, we employ a heuristic algorithm to approximate the OTSD efficiently.

For illustration purposes, we show the OTSD trends on dataset with ID 3, which serves as a representative example of the observed patterns. Note that the iterations start from 1, even though the x-axes on the plots are signed from 0.

Figure 5.11 illustrates that the highest OTSD values - indicating a broader or more extensive exploration of the search space - were observed for all methods based on the GP surrogate model. They were almost always higher than OTSD values for Random Search. In contrast, methods with ET and RF as surrogate models achieved lower values than Random Search and have relatively similar results to each other, suggesting more structured exploration behavior. The surrogate model appears to have a stronger influence on the exploration pattern than the acquisition function, as methods using the same model exhibit similar OTSD trends regardless of the acquisition strategy.

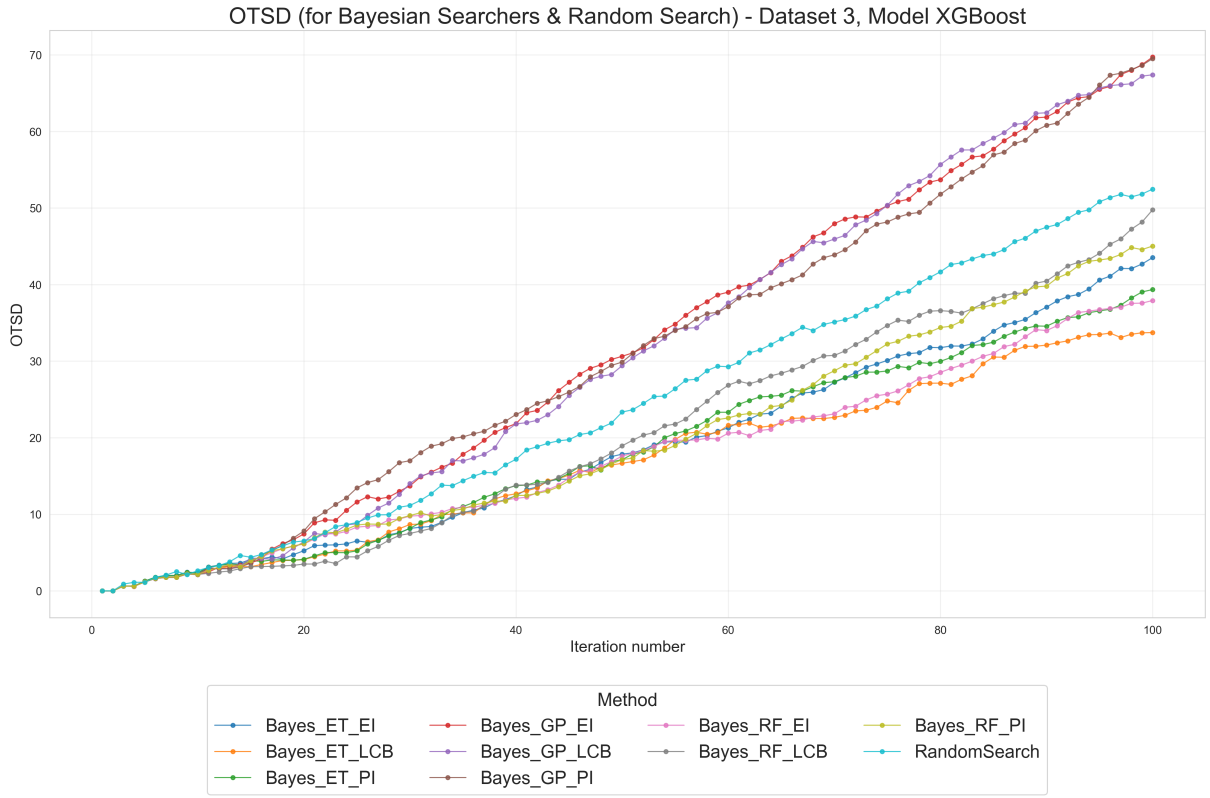

Figure 5.11: OTDS for XGBoost model on dataset with ID 3.

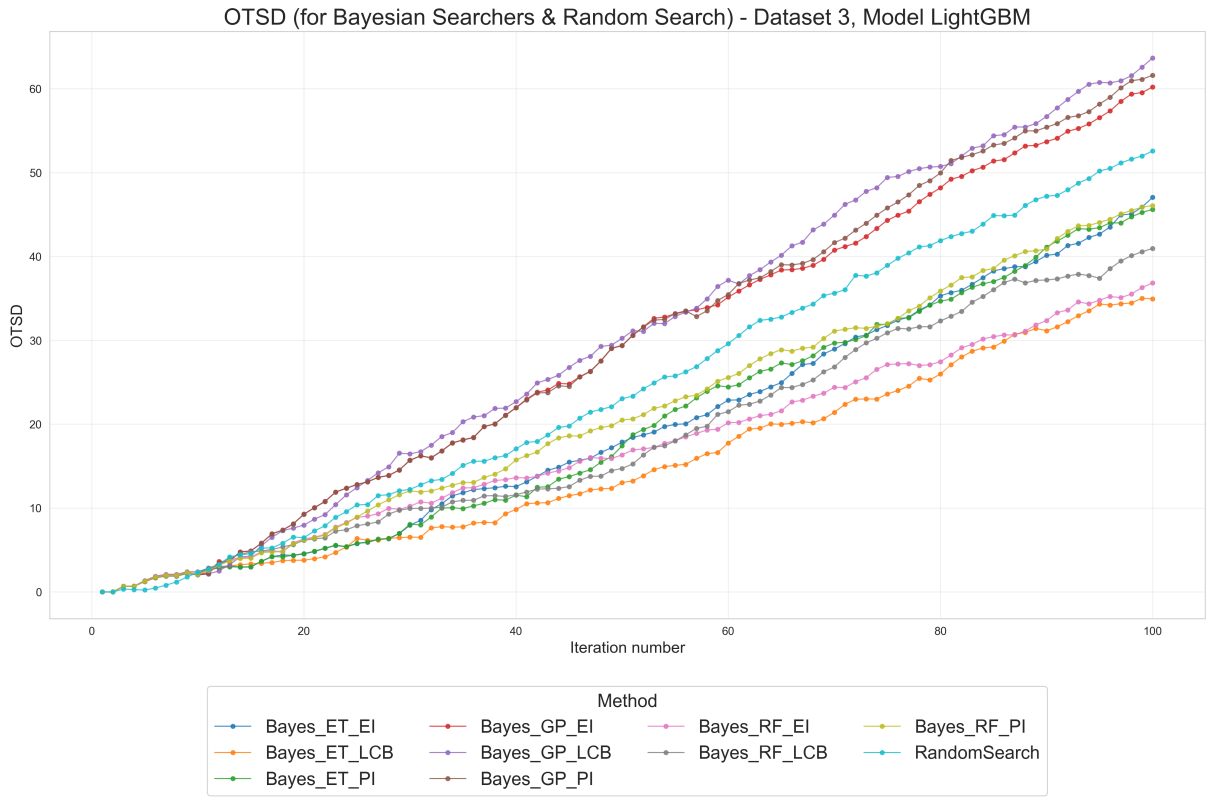

Figure 5.12: OTDS for LightGBM model on dataset with ID 3.

Similar to Figure 5.11, Figure 5.12 shows that the highest OTSD values - indicating a broader or more extensive exploration of the search space - were observed for all methods based on the GP surrogate model. Between methods based on GP surrogate model and other Bayesian methods are values for Random Search. Methods using ET and RF surrogate models achieved lower and relatively similar OTSD scores, suggesting a more structured exploration behavior.

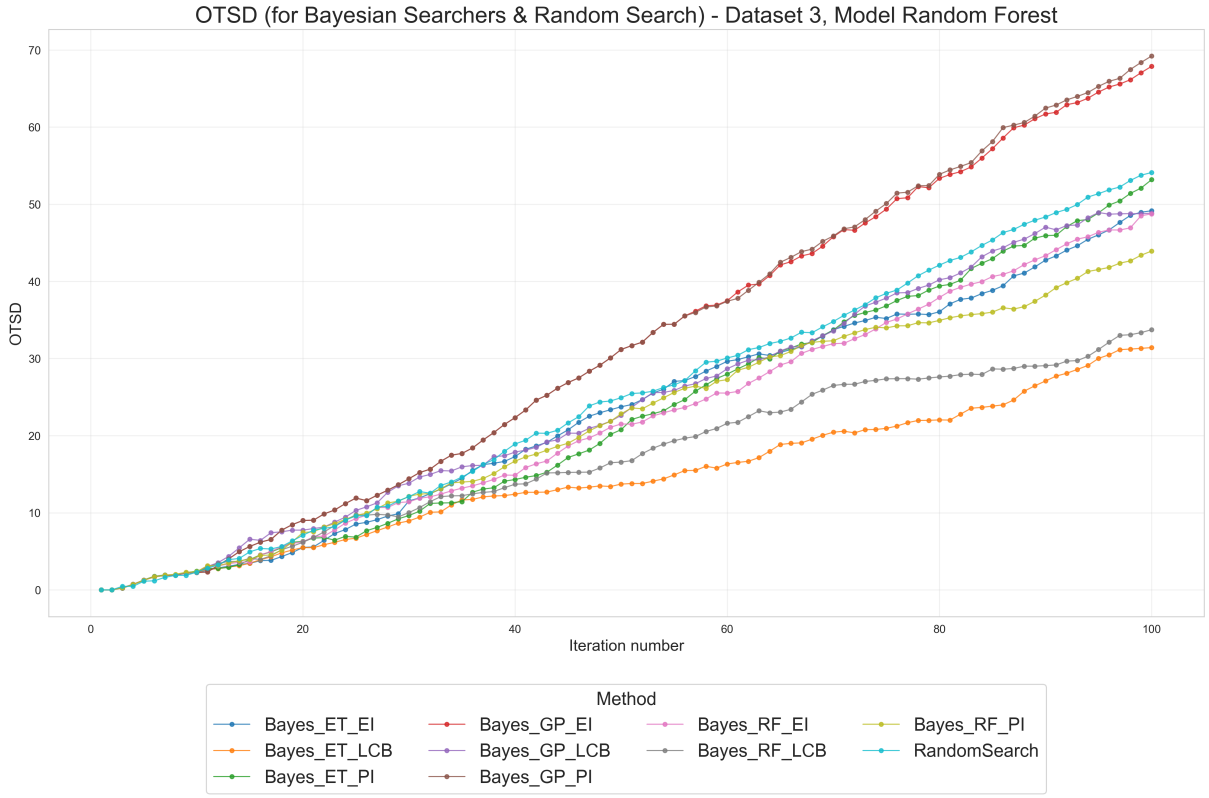

Figure 5.13: OTDS for Random Forest model on dataset with ID 3.

As showed in Figure 5.13, methods `Bayes_GP_EI` and `Bayes_GP_PI` have the highest OTSD values. While `Bayes_ET_LCB` has the lowest OTSD value. A little bit higher value is achieved by `Bayes_RF_LCB`. The remaining methods have similar results, falling between those mentioned above.

To comprehensively compare the performance of different optimization methods across multiple datasets, we performed a aggregation of OTSD (across datasets and models). The aggregated approach provides a more robust assessment of each method's effectiveness than initial analysis for individual datasets.

The OTSD results, aggregated across datasets and models, were computed as the arithmetic mean along with the corresponding standard deviation. On the resulting plot, the x-axis represents the iteration number, while the y-axis shows the OTSD values (mean $\pm$ standard deviation). Each Bayesian method is represented by a line indicating the mean OTSD across it-

erations. Surrounding each line is a shaded area, which visualizes the standard deviation, thereby illustrating the variability of the results. Note that the iterations start from 1, even though the x-axes on the plots are signed from 0.

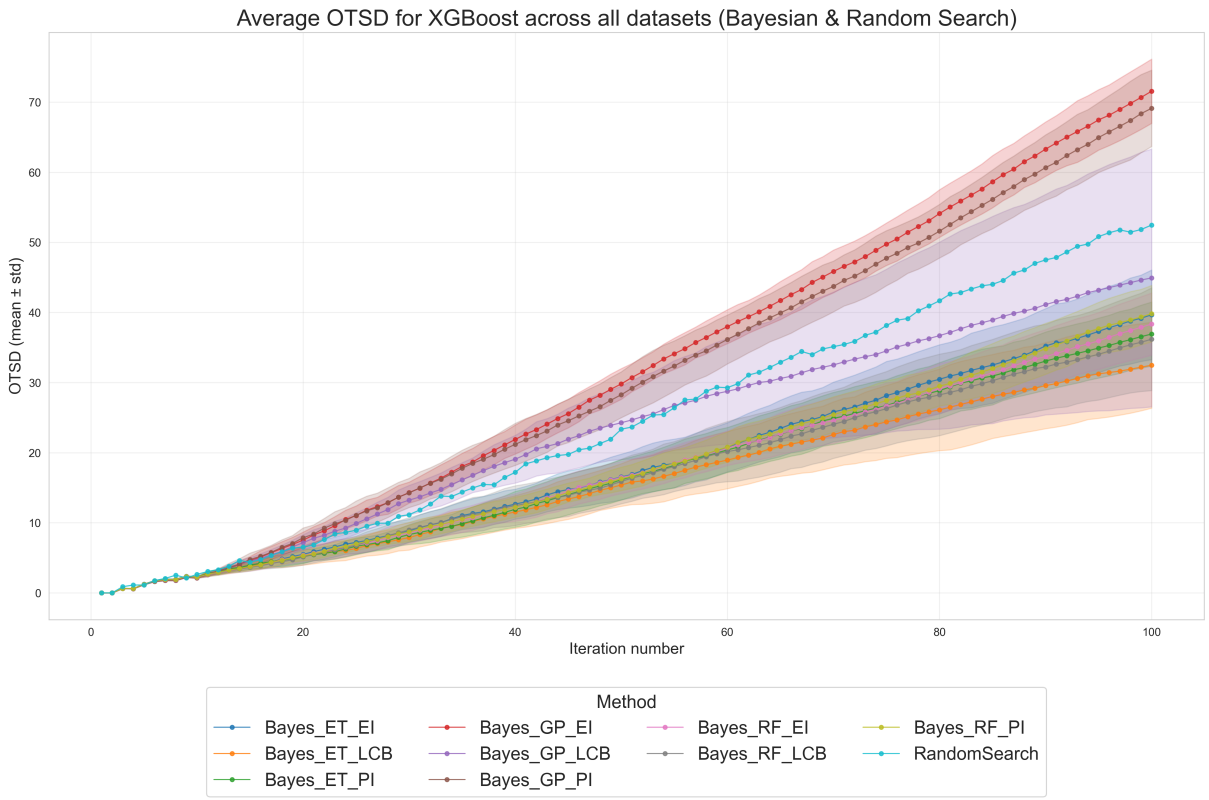

Figure 5.14: Agreggated OTSD for XGBoost model.

Figure 5.14 shows that for XGBoost the best Bayesian method in the context of OTSD is `Bayes_ET_LCB` and the worst are `Bayes_GP_EI` and `Bayes_GP_PI`. Methods with GP as surrogate models have weaker results compared to the rest (leaving out `Bayes_GP_LCB` from about iteration 55 compared to Random Search).

Figure 5.15 illustrates that for LightGBM the best Bayesian method in the context of OTSD is `Bayes_ET_LCB` and the worst are `Bayes_GP_EI` and `Bayes_GP_PI`. Methods with GP as surrogate models have much weaker results compared to the rest.

Figure 5.16 conveys that for Random Forest the best Bayesian method in the context of OTSD are `Bayes_ET_LCB` and `Bayes_RF_LCB` and the worst are `Bayes_GP_EI` and `Bayes_GP_PI`. Methods with GP as surrogate models have much weaker results compared to the rest (leaving out `Bayes_GP_LCB` compared to Random Search because they have similar results - sometimes one has higher scores, sometimes the other).

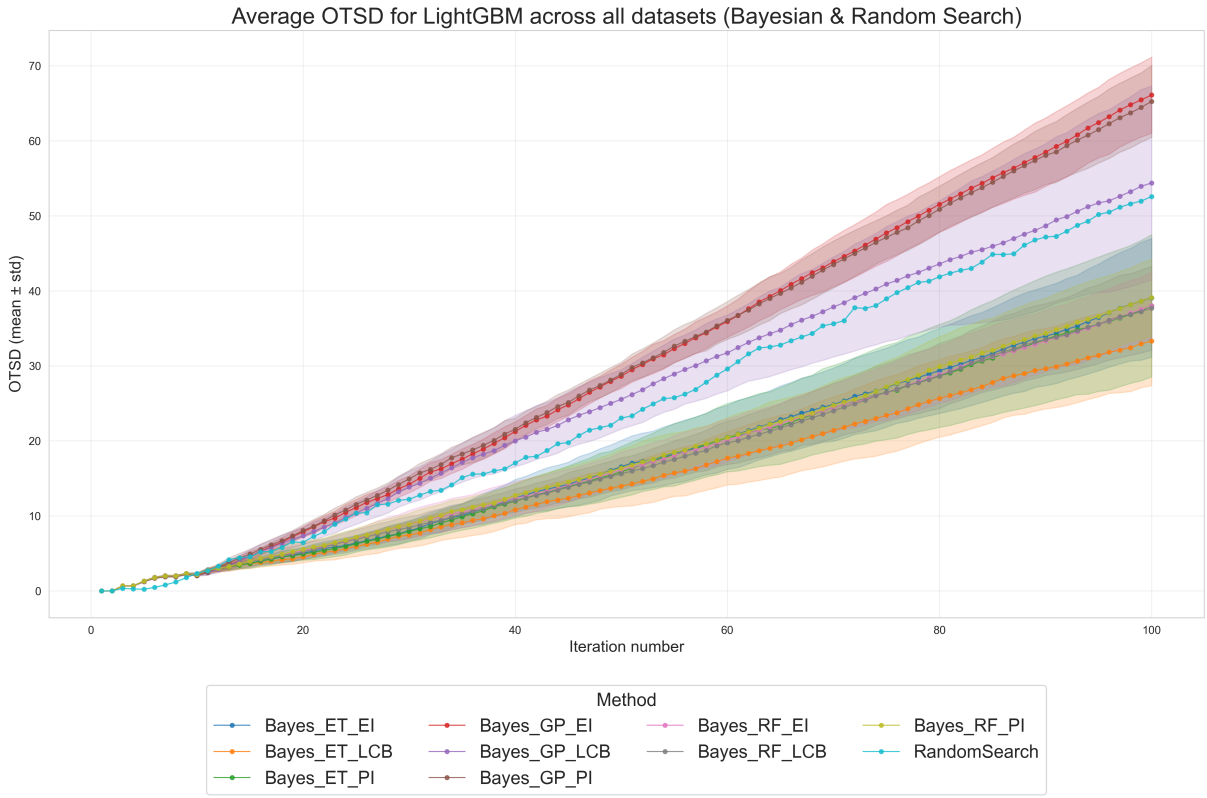

Figure 5.15: Agreggated OTSD for LightGBM model.

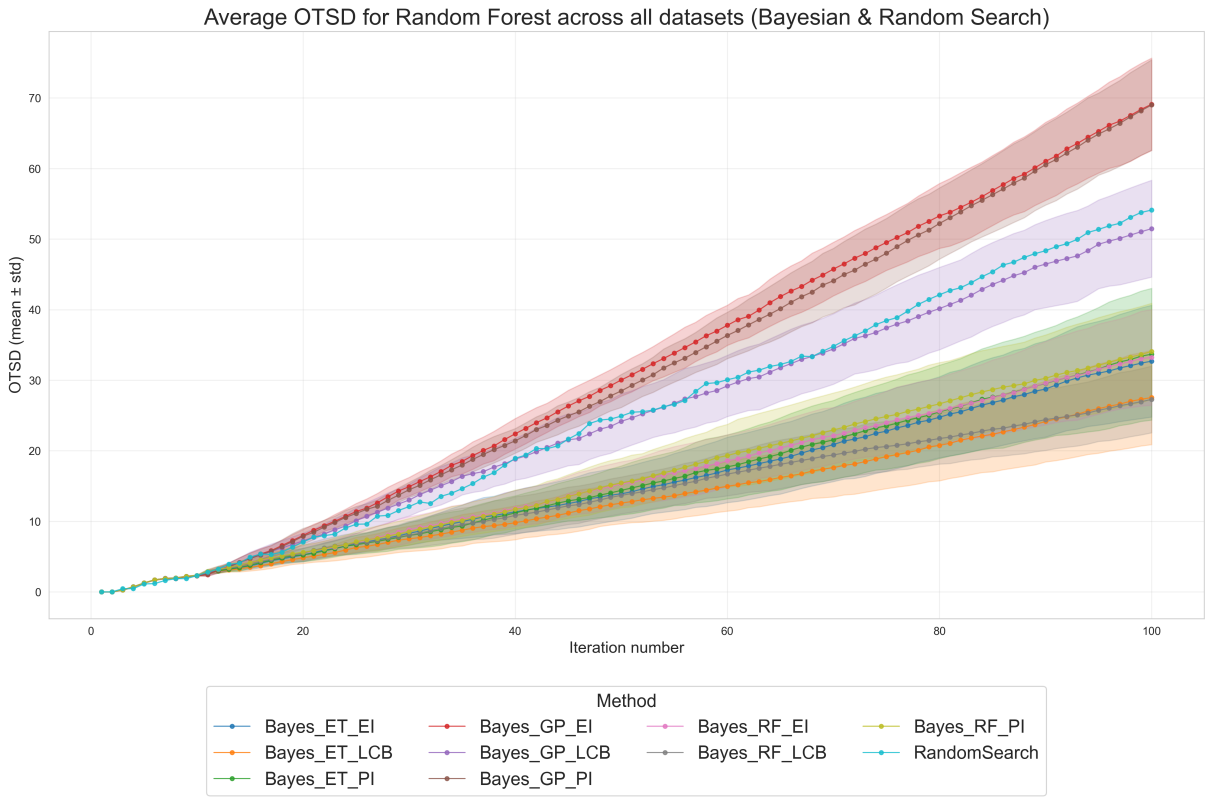

Figure 5.16: Agreggated OTSD for Random Forest model.

Conclusions. When analyzing OTSD, which captures the spread and structure of the exploration process, methods using GP surrogate models tend to have higher OTSD than Random Search (least structured exploration), while `Bayes_ET_LCB` and `Bayes_RF_LCB` exhibit lower OTSD, suggesting more focused and consistent search patterns. This contrast reflects different exploration strategies and their potential trade-offs with convergence speed.

5.7. General conclusions of the experiment

The conducted experiment consistently demonstrate that Bayesian Optimization methods outperform traditional hyperparameter tuning techniques such as Grid Search and Random Search across all evaluated models (XGBoost, LightGBM and Random Forest) and evaluation metrics (AUC, ADTM, OTSD). This superiority manifests in several key aspects:

- faster convergence,
- efficiency in model tuning (ADTM),
- stable ranking performance for ADTM,
- more structured and model-aware exploration behavior (OTSD) - for methods based on Extra Trees and Random Forest surrogate models,
- robustness across models and datasets.

In summary, the analysis strongly supports the adoption of Bayesian Optimization, particularly with well-chosen surrogate models and acquisition functions, as a preferred strategy for hyperparameter tuning. These methods offer a favorable balance between exploration and exploitation, enabling quicker convergence to high-performance configurations with reduced computational cost.

6. Conclusions

This thesis has provided an analysis of the fundamental components of Bayesian Optimization and their influence on the exploration - exploitation trade-off in the context of optimizing expensive black-box functions. Through both theoretical investigation and extensive empirical evaluation, the study has demonstrated that Bayesian Optimization methods are not only theoretically sound but also practically effective tools for improving optimization efficiency in machine learning tasks, particularly hyperparameter tuning.

The experimental results clearly indicate that the choice of surrogate model and acquisition function plays a crucial role in the optimization process. Among the evaluated combinations, Gaussian Processes paired with the Lower Confidence Bound acquisition function consistently outperformed other configurations, leading to faster convergence and more reliable identification of optimal solutions. These findings corroborate the growing body of evidence supporting the superiority of Bayesian Optimization over traditional techniques such as Grid Search and Random Search, especially when computational resources are limited or function evaluations are costly.

Importantly, this work highlights that the effective management of the exploration–exploitation balance is central to achieving optimal performance. Bayesian Optimization's adaptive approach enables a more intelligent search strategy, efficiently exploring unknown regions of the parameter space while exploiting promising areas identified through prior evaluations. This balance significantly enhances the optimization process's speed and outcome quality, making these methods highly valuable for real-world applications.

Bibliography

Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2), 2023.

Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53), 2019.

Stefan Meisenbacher, Marian Turowski, Kaleb Phipps, Martin Rätz, Dirk Müller, Veit Hagenmeyer, and Ralf Mikut. Review of automated time series forecasting pipelines. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(6), 2022.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*, chapter 2. The MIT Press, 2006.

Roman Garnett. *Bayesian optimization*, chapter 5. Cambridge University Press, 2023.

Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. Scikit-optimize/scikit-optimize, 2021.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85), 2011.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML Benchmarking Suites. *preprint arXiv:1708.03731v2*, 2019.

Katarzyna Woźnica, Mateusz Grzyb, Zuzanna Trafas, and Przemysław Biecek. Consolidated

learning: a domain-specific model-free optimization strategy with validation on metamimic benchmarks. *Machine Learning*, 113(7), 2024.

Leonard Papenmeier, Nuojin Cheng, Stephen Becker, and Luigi Nardi. Exploring exploration in bayesian optimization. *preprint arXiv:2502.08208*, 2025.

List of symbols and abbreviations

BO	Bayesian Optimization
ML	machine learning
HPO	Hyperparameter Optimization
HP	hyperparameter
GE	generalization error
GP	Gaussian Process
RF	Random Forest
ET	Extra Trees
LCB	Lower Confidence Bound
EI	Expected Improvement
PI	Probability of Improvement
ROC	Receiver Operating Characteristic
AUC	Area Under the Receiver Operating Characteristic Curve
ADTM	Average Distance to Maximum
OTSD	Observation Traveling Salesman Distance

List of Figures

List of tables

List of appendices

1. Code of the experiment (file: `Experiment.ipynb`)