Automatic Relevance Detection (ARD) and Hyperparameter Tuning in the Training of Gaussian Process Regression-based Surrogate Models: A Machine Learning Approach

Md Shahnawaz Ahmed 2301640
Master's Degree IT: Master's Degree Programme in Information Technology, Computer Science
Supervisors: Anders Brink, Jerker Björkqvist
Faculty of Science and Engineering
Åbo Akademi University
2024-2025
Version: 1.0.0

Abstract

Gaussian Process Regressor (GPR) is a Bayesian optimization-based non-parametric regression technique that has emerged as a very successful modeling method for creating surrogate models in Machine Learning (ML), particularly where real-world experimental costs are very high and limited. The effectiveness of GPR hinges on precise hyperparameter calibration and the model's ability to discern and prioritize relevant input features. This thesis investigates GPR with a particular emphasis on hyperparameter tuning and integrating Automatic Relevance Determination (ARD) to assess the significance of input features and improve surrogate modeling. A series of controlled experiments is conducted on generated synthetic datasets with diverse dimensionality, noise levels, signal variances, and feature length scales, enabling a systematic evaluation of model behavior under different training circumstances. Implemented using Python and the scikit-learn library, the experimental framework facilitates practical assessment of GPR's sensitivity to initialization, sample size, and kernel configurations. Model performances are evaluated for comparison using multiple optimization strategies under different settings. Although these optimization strategies have various algorithms, they still serve a few optimization goals, like maximizing the likelihood and minimizing the error/loss function. The proposed alternate strategies improve the existing model's drawbacks, model generalization, reduce overfitting, and decrease computational overhead. These findings contribute to developing robust and interpretable surrogate models, with implications for scientific computing, optimization tasks, and engineering applications.

Keywords: Gaussian Process Regression, surrogate modeling, hyperparameter optimization, kernel selection, log marginal likelihood, conditional lml, ARD, model training, machine learning, regression.

Acknowledgment

The results were achieved in the Flexible Clean Propulsion Technologies (Flex-CPT) project (ref. 10526/31/2023) that was co-funded by Business Finland.

Flex-CPT is a research initiative coordinated by the University of Vaasa and co-founded by Business Finland, along with 17 consortium members. Flex-CPT develops flexible, fuel-agnostic, zero-emission powertrain technologies to tackle the major obstacles of decarbonizing off-road and marine transportation.

I am grateful for the opportunity to contribute to this important effort toward a sustainable and economically viable future for the Finnish powertrain industry. The research was carried out at Åbo Akademi University, and I am grateful to the institution for providing the resources and academic environment required for this study.

Additionally, I would like to convey my gratitude to my supervisor, who guided me throughout this study. Also, thanks to my research colleagues and all other Flex-CPT project participants for their support, direction, and insightful conversations during this work. Finally, I want to cordially thank my family for their support and understanding, which helped me to focus on this study.

Contents

List of Abbreviations and Symbols

ℓ *Lengthscale* 24

$\ell_{feature}$ *Feature Length Scale of the Underlying Data* 18, 25–29, 31, 61–63

ℓ_{init} *Given initial kernel length scale* 61, 63

$\ell_{learned}$ *Learned length scale* 27, 28, 31, 63

AI *Artificial intelligence* 2

ARD *Automatic Relevance Detection* 2, 3, 7, 17, 34, 36, 38, 40, 42–44

CI *Confidence Interval* 7, 63

CLML *Conditional Log Marginal Likelihood* 15, 27, 28, 34, 36, 38, 40–42, 45

GP *Gaussian Process* 1, 8, 14, 39

GPR *Gaussian Process Regressor* 1–3, 5, 7, 11–14, 18–20, 30, 31, 36, 38, 40, 43, 45, 49, 62, 63

GPs *Gaussian Processes* 17

LML *Log Marginal Likelihood* 14, 15, 26–28, 33, 34, 36, 38, 40–42, 45

LOO *Leave-One-Out* 27, 28, 34, 36, 38, 40–42, 45

LOO-CV *Leave-One-Out Cross Validation* 15

MAE *Mean Average Error* 27

ML *Machine Learning* 2, 3

NLML *Negative Log Marginal Likelihood* 26

PRG *Polynomial Regression* 6

RBF *Sqaured Exponential* 9

RBF *Radial Basis Function* 19, 45, 62

RMSE *Root Mean Squared Error* 27

SE *Squared Exponential* 19

TP *Training points* 24, 27, 28, 34, 41

CHAPTER 1

Introduction

Human intelligence is exceptionally skilled at creating comprehensive mental models from incomplete data. Recognizing a popular face from a partial or obscured image with some hints demonstrates how the human brain fills in the missing information using prior knowledge. As Forrester et al. (2008) illustrate, "The key to such apparently impressive feats is that we actually know a great deal about the obscured parts". Prior data, like a famous human face, a roughly symmetrical image, the center point being the nose, etc., reduces the search space needed to formulate a decision. Surrogate modeling works on a similar principle of constructing effective assumptions of an unknown function with the help of a few known data points (p. ix) [1].

Simulations have become increasingly sophisticated as scientific models and engineering systems have progressed over the decades. However, they also become computationally expensive, requiring hours of processing for a single run. These simulations require a good amount of computational resources or extensive physical testing, resulting in the need for adequate substitutes for high-fidelity simulations, particularly when multiple iterations or sensitivity studies are needed. Surrogate models can cut computational costs by approximating complex systems without sacrificing accuracy. As Forrester et al. (2008) point out, "The basic idea in the 'surrogate model' approach is to avoid the temptation to invest one's computational budget in answering the question at hand and, instead, invest in developing fast mathematical approximations" (p. xiii) [1].

Among various surrogate models, those built with a Gaussian Process are widely popular and often used as the underlying model in Bayesian optimization for approximation tasks [2]. A Gaussian Process Regressor (GPR) is a well-known non-parametric and probabilistic Bayesian approach to regression that is ideal for surrogate modeling [3, 4]. According to Rasmussen and Williams (2006), GPR quantifies uncertainty for each prediction and estimates output values by treating predictions as distributions rather than fixed values. This is essential for risk-aware decision-making [3].

A GPR model's performance is primarily determined by how its internal similarity or covariance function is configured and by related hyperparameters or key settings. These parameters control the ability of the model to fit observed data and generalize. Bishop

(2006) and Lotfi et al. (2022) highlight that inadequate hyperparameter selection can result in overfitting or underfitting, reducing the model's trustworthiness [5, 6].

Automatic Relevance Detection (ARD) enhances the interpretability and relevance of GPR models in high-dimensional spaces. By giving each input dimension its hyperparameters (i.e., length scale), ARD expands on standard internal similarity functions, allowing the model to identify the significance of each feature and disregard those that are not relevant [5, 7]. In case of having multidimensional features without any proper indicators of significance, ARD is useful to determine the similarity and extract important features.

In a study by Liu et al. (2020) predict the calendar aging of lithium-ion batteries is predicted using GPR with an Automatic Relevance Detection (ARD) covariance function. The model can automatically determine which factors, such as temperature, state-of-charge (SOC), and storage time, impact battery degradation the most by using ARD to give each input feature its length scale. This selective sensitivity enhances the interpretability and performance of the model, especially when there is a shortage of training data compared to other models. The researchers demonstrate that the accuracy of their ARD-GPR approach is higher than other machine learning methods and conventional GPR [8].

This thesis will systematically compare the performance of different optimization strategies (maximizing the log marginal likelihood, conditionally maximizing the log marginal likelihood, and minimizing the error) in learning the underlying length scale of the multi-dimensional input data and explore a machine learning framework that gives the best model performance with a low cost. Moreover, this study will evaluate how different sizes of data, features, and ARD affect the training efforts of constructed models. The results are intended to aid in developing surrogate modeling methods that are more precise, comprehensible, and computationally efficient.

1.1 Motivation

Expensive simulations or costly experimental data collection are frequently required to develop and validate physical systems in many real-world engineering and scientific problems. In this era of Machine Learning and Artificial intelligence, with a large amount of computing resources and scalable infrastructure, computer-aided simulations have reached unprecedented sophistication and are reducing bottlenecks like complexity, time, and cost. Surrogate models effectively approximate the behavior of complex systems, cutting those bottlenecks. Using Gaussian Process Regressor (GPR), as a forefront of surrogate modeling techniques, with the help of Automatic Relevance Detection (ARD) for significant dimensions identification, can help to potentially reduce computational complexity with-

out sacrificing accuracy [1, 3].

Even with these developments, there remain questions regarding the optimal strategies for hyperparameter optimization in GPR models with ARD and various sizes of datasets. This study attempts to close this knowledge gap by methodically assessing various optimization techniques under various data scenarios. However, the right choice of hyperparameters, which control the internal similarity function's behavior and the model's capacity for generalization, is crucial to its performance. Optimizing the balance between model fit and predictive uncertainty is frequently not possible with traditional default or manual settings.

The study's motivation is to systematically investigate hyperparameter tuning strategies by conducting experiments on GPR datasets of various shapes and dimensions and evaluating how ARD enhances model performance and interpretability in diverse scenarios.

1.2 Research Questions

1. How can automatic relevance detection and hyperparameter tuning based on similar data decrease the effort of training?

2. What effects does dataset size have on the efficacy of hyperparameter tuning?

3. Which methods for optimizing hyperparameters offer the best balance between model performance and computational expense?

CHAPTER 2

Background

2.1 Surrogate Modeling

In engineering, surrogate models serve as simplified approximations of complex systems. Particularly, when direct evaluations are computationally intensive or infeasible. Simulations are frequently employed to evaluate the connection among different parameters, such as constraints, objectives, and design variables. Since high-fidelity simulations can be time-consuming and resource-intensive, surrogate models provide a practical alternative by approximating outcomes at significantly lower computational cost. These models approximate intricate and computationally costly simulations. As highlighted by Forrester et al. (2008), with much lower computational costs, surrogate models offer a useful way to carry out sensitivity analysis, uncertainty quantification, and optimization [1].

Common surrogate modeling techniques include artificial neural networks, support vector machines, radial basis functions, and polynomial regression models, each suited to different types of approximation problems [4]. However, Gaussian Process Regressor (GPR) is notable for its prediction and uncertainty estimation capacity. Due to this, GPR is highly in demand for those areas where both characteristics are required [3].

2.2 Parametric model vs Non-Parametric model

Parametric models have fixed parameters, while non-parametric models work with infinite parameters. Fixed parameters are explicitly defined in the model. A good example of a parametric model that assumes a fixed nonlinear form with parameters is polynomial regression, which usually estimates coefficients for powers of input variables. On the other hand, GPR is a non-parametric model. Instead, it infers the function shape directly from the data with the help of a kernel or covariance function [3].

- **Parametric model:**

The Polynomial Regression (PRG) parametric model can be expressed as [4]:

$$f_i(z) = \sum_{j=1}^{N_{\text{PRG}}} \beta_{\mathbf{j}} z_j^{(i)} + \varepsilon_i \tag{2.1}$$

with:

$$\mathbb{E}[\varepsilon_i] = 0, \quad \mathbb{V}[\varepsilon_i] = \sigma^2$$

- $f_i(z)$: predicted output for the i^{th} data point.
- $z_j^{(i)}$: the j^{th} feature (input variable) of the i^{th} sample.
- β_j: **parameters** (regression coefficients) of the model to be learned.
- ε_i: random noise or error term.
- $\mathbb{E}[\varepsilon_i] = 0$: error has zero mean.
- $\mathbb{V}[\varepsilon_i] = \sigma^2$: error has constant variance σ^2.

- **Non-Parametric model:**

  A GP represents a non-parametric probabilistic model that defines a distribution over possible functions and can be formally described as follows [3]:

$$f(x) \sim \mathbf{GP}(m(x), k(x,x')) \tag{2.2}$$

  - $f(x)$: a random function taken from a GP.
  - $m(x)$: the mean function.
  - $k(x,x')$: the covariance function.

  This implies that each finite set of function values $f(x_1), f(x_2), \ldots, f(x_n)$ has a multivariate normal distribution.:

$$[f(x_1), \ldots, f(x_n)]^\top \sim \mathrm{N}(m, K) \tag{2.3}$$

  where m is the mean vector and K is the covariance or kernel matrix as determined by the kernel function..

2.3 Gaussian Process Regressor (GPR)

Gaussian Process Regressor has evolved and risen as a powerful non-parametric probabilistic machine learning algorithm in statistical learning and engineering design (p. xiii). The foundational work by Rasmussen and Williams (2006) contributed to the groundwork for GPR to present it as a versatile non-parametric method that offers not only forecasts, but also estimates of uncertainty [3]. Their work expanded on a previous tutorial by MacKay (1998) that linked GPR to Bayesian frameworks and contributed to the popularization of these techniques [9]. Rasmussen and Williams showed how GPR may capture intricate patterns with comparatively few hyperparameters by formalizing it inside a Bayesian framework. This groundwork considerably expanded the approach of GPR, especially in the fields of hyperparameter adjustment and feature selection [3].

Bishop (2006) further explained GPR's relationship to other kernel techniques and Bayesian approaches by placing it inside a larger probabilistic modeling framework. He characterizes GPR as a generalized form of Bayesian linear regression, enhanced by the use of an infinite set of basis functions. He gives a detailed explanation of how the kernel function encodes prior beliefs about function attributes and links GPs to other kernel approaches [5].

Connecting surrogate modeling with GPR, Forrester, Sóbester, and Keane (2008) present experimental design guidelines on surrogate modeling focusing on GPR as a powerful method for creating computationally effective metamodels that simulate costly simulations or testing [1].

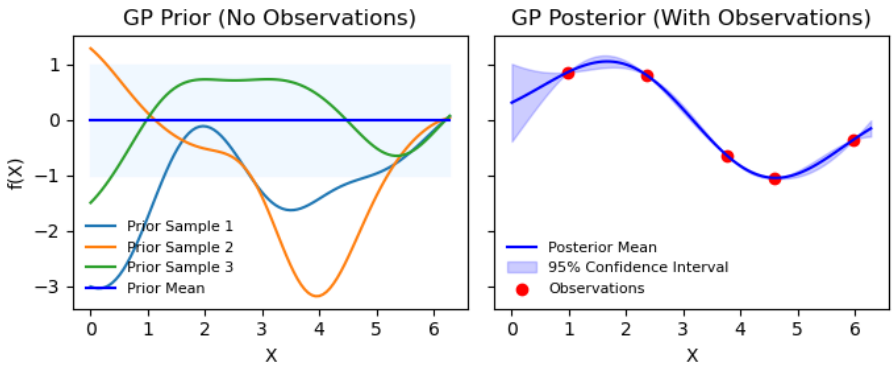

Figure 2.1: Using a two-column layout, GPR visualizations display (left) the prior distribution with three sampled functions and the mean, which illustrates the model's assumptions before data observation, and (right) the posterior distribution with the mean, 95% Confidence Interval, and five noisy observations, which represents the updated model following data fitting.

Based on Bayesian statistics, Gaussian Process Regressor creates a posterior by up-

dating a prior over functions with observed data. This probabilistic framework allows GPR to generate point predictions and corresponding confidence intervals. In Figure 2.1, the prior samples show the GP's natural variability based on the RBF kernel's variance, with the mean representing the expected function before seeing any data. After observing data, the posterior mean adapts to fit it, while the 95% confidence interval reflects prediction uncertainty, wider due to the noise variance (σ_n^2). The difference between prior and posterior shows how data reduces uncertainty near known points while maintaining smoothness.

GPR performs inference by modeling a distribution over functions. As outlined in Chapter 2.2 of Rasmussen and Williams (2006), a Gaussian Process is fully defined by its mean function $m(x)$ and covariance function $k(x,x')$ [3]. Beyond the formulation in Equation 2.2, GPR makes predictions for a test input x_* by computing the posterior distribution, also Gaussian, conditioned on the training data $\{X,y\}$. Here, $X \in \mathbb{R}^{n\times d}$ denotes the input matrix with n samples and d features, and $y \in \mathbb{R}^n$ represents the corresponding target outputs. This approach allows GPR not only to predict the mean but also to estimate the variance, offering insight into the model's uncertainty about its predictions.

$$f_* \mid X, y, X_* \sim \mathrm{N}(\bar{f}_*, \mathrm{cov}(f_*)) \tag{2.4}$$

Mean: $\bar{f}_* = K(X_*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1}\mathbf{y}$
Covariance: $\mathrm{cov}(f_*) = K(X_*,X_*) - K(X_*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1}K(X,X_*)$

where

- $K(X,X)$: the kernel matrix of the training inputs.
- $K(X_*,X)$: covariance between test inputs and training inputs.
- $K(X_*,X_*)$: covariance of test inputs with themselves.
- σ_n^2: noise variance.
- I: identity matrix (of size $n \times n$).
- y: observed output values (training labels).

2.4 Covariance Functions

Covariance Function or Kernel exhibits the similarity between observations (input samples). According to Rasmussen & Williams (2006) in Chapter 2.2, "The choice of covari-

ance function encodes assumptions about the function being modeled, such as smoothness, periodicity, and amplitude." [3].

The "Kernel Cookbook" by David Duvenaud (2014) [10] is a noteworthy addition to kernel approaches, especially regarding Gaussian processes. Duvenaud sees kernel design as building with blocks in that basic kernels are parts that can be joined to form more powerful models. The main contrast of this cookbook is that each kernel reflects a particular assumption about the function's behavior.

In machine learning, a kernel or covariance function is a mathematical tool that determines how similar two data points are. It compares two input vectors (e.g., x, x') and returns a single number value indicating their similarity. This number assists the algorithm in determining how closely connected the two points are in a higher-dimensional (feature) space, even if they do not appear similar at first glance.

The key idea is to perform computations in a high-dimensional feature space without explicitly mapping the data into that space. This concept is known as the *kernel trick*, which enables significant computational savings when dealing with high-dimensional representations. A kernel function implicitly incorporates prior assumptions about the target function's characteristics, such as stationarity, periodicity, and smoothness [3].

There are some common kernels, such as Sqaured Exponential (RBF) kernel, Matern kernel, Rational Quadratic kernel, Periodic kernel, and others. Selecting the right kernel is essential to the model's functionality since it controls the GP's behavior and capacity for generalization. David Duvenaud (2014) described different covariance functions, such as [10, 11]:

Squared Exponential (SE) kernel

The SE kernel, sometimes referred to as the Radial Basis Function (RBF) kernel, is well-liked for its limitless differentiability and smoothness:

$$k_{\text{SE}}(\mathbf{x},\mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}\right) \tag{2.5}$$

- σ_f^2: signal variance, calculates the function's average deviation from its mean.
- ℓ: length scale, controls the frequency of the curve of the underlying function..

With **Automatic Relevance Determination (ARD)**, each input dimension d receives its own length scale ℓ_d:

$$k_{\text{SE-ARD}}(\mathbf{x},\mathbf{x}') = \sigma_f^2 \exp\left(-\sum_{d=1}^{D}\frac{(x_d - x_d')^2}{2\ell_d^2}\right) \tag{2.6}$$

Rational Quadratic (RQ) kernel

The Rational Quadratic kernel is a scale mixture of SE kernels with different length scales:

$$k_{\mathrm{RQ}}(\mathbf{x},\mathbf{x}') = \sigma_f^2 \left(1 + \frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{2.7}$$

where α is the scale mixture parameter. As $\alpha \to \infty$, this kernel approaches the SE kernel.

Periodic kernel

The Periodic kernel captures repeating patterns with a given period p:

$$k_{\mathrm{Per}}(\mathbf{x},\mathbf{x}') = \sigma_f^2 \exp\left(-\frac{2\sin^2(\pi\|\mathbf{x}-\mathbf{x}'\|/p)}{\ell^2}\right) \tag{2.8}$$

This kernel is suitable for time-series or seasonal data with known cycles.

Hyperparameters of kernels [10]:

Table 2.1: Associated hyperparameters with different kernels

Kernel Name	**Hyperparameters**
Squared Exponential (RBF)	Lengthscale (l), Signal variance (σ_f^2)
Matérn	Lengthscale (l), Signal variance (σ_f^2), Smoothness parameter (ν)
Rational Quadratic	Lengthscale (l), Signal variance (σ_f^2), Scale mixture parameter (α)
Linear (Non-Stationary)	Variance (σ^2)
Polynomial (Non-Stationary)	Coefficient (c), Degree (d)
Periodic	Lengthscale (l), Period (p), Signal variance (σ_f^2)
Constant	Constant value (c)

2.5 Hyperparameter Tuning in GPR

The hyperparameters affect the model's accuracy and efficiency, so selecting the appropriate ones is crucial. Otherwise, it may learn the training data well but perform poorly on new test data, or it may not learn enough to perform well. In contrast to traditional parameters that are learned directly from data, GPR hyperparameters are optimized by maximizing the marginal likelihood, balancing data fit and model complexity[2].

According to MacKay (1999), who made links between maximal likelihood estimation (MLE) and Bayesian model selection principles, early methods of hyperparameter optimization mostly depended on MLE.

Rasmussen and Williams (2006) established Marginal likelihood maximization as the primary method for GP hyperparameter optimization in their work. Moreover, they presented analytical gradients of the marginal likelihood concerning hyperparameters, which allowed for the adoption of gradient-based optimization approaches. The research provided practical advice on starting tactics and optimization processes, such as employing several restarts to prevent local optima and optimizing specific parameters in log-space [3].

Aligning with Rasmussen and Williams (2006), Bishop (2006) offers a logical method for estimating maximum likelihood, in which the marginal likelihood (or evidence) function is maximized to adjust hyperparameters. Occam's razor is naturally used in this strategy, which balances data fit and model complexity without the need for additional validation data. Bishop highlights that this optimization method offers an intuitive Bayesian framework for automatically discovering the ideal kernel parameters and noise levels, even though it can occasionally be difficult due to several local optima [5]. Getting the best out of the model, hyperparameter tuning will help to select the optimal values.

The role of hyperparameters in GPR

Hyperparameters in the GPR include the noise variance, the kernel's length-scales, and signal variance:

- **Noise variance:**
  In order to assist the model differentiate between signal and noise and improve generalization and uncertainty estimates, the noise variance (σ_n^2) in GPR represents the amount of noise or error that is anticipated in the observed data.

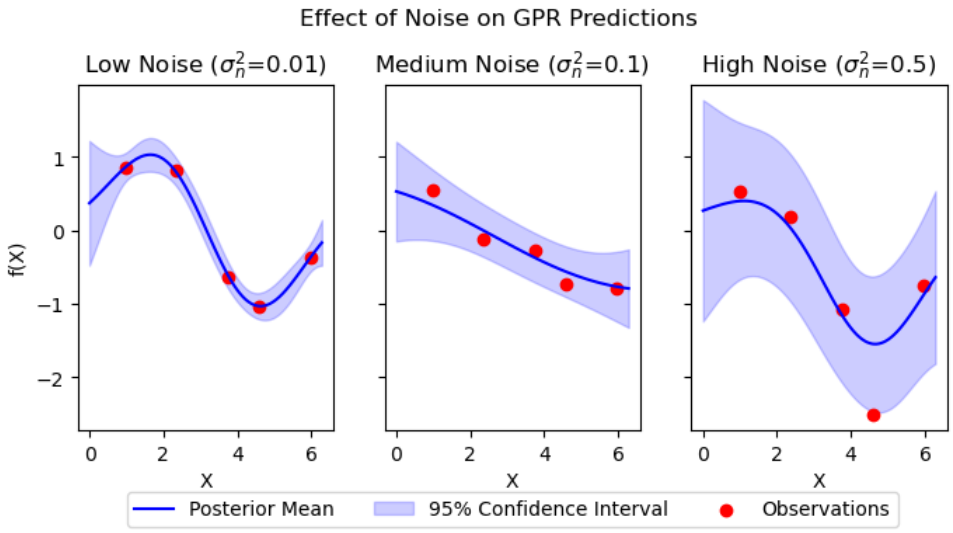

Figure 2.2: Illustration of the effect of noise on GPR predictions, presented in three panels: (left) low noise ($\varepsilon = \sigma_n^2 = 0.01$) with a tight fit to observations, (middle) medium noise ($\varepsilon = \sigma_n^2 = 0.1$) showing increased uncertainty, and (right) high noise ($\varepsilon = \sigma_n^2 = 0.5$) with a smoother posterior mean and wider confidence intervals, each displaying the posterior mean, 95% confidence interval, and observed data points. Where ε represents the data noise and σ_n^2 represents the assumed noise in the model.

- **Length scales:**
  The length scale (ℓ) is a hyperparameter in covariance or kernel functions such as the SE kernel or the RBF. It establishes how rapidly the values of the kernel function can alter in response to changes in the input. Whereas a lengthy length scale suggests smoother, slower variation throughout input space, a small length scale allows quick changes (high change).

- **Signal variance:**
  Signal variance (σ_f^2) in GPR refers to the variability in observed data or latent function values, which is commonly represented by the covariance function or kernel's variance parameter, which regulates the amplitude of the function's fluctuations. High variance might imply noisy data or complicated patterns, which affect GPR's predicting uncertainty and performance.

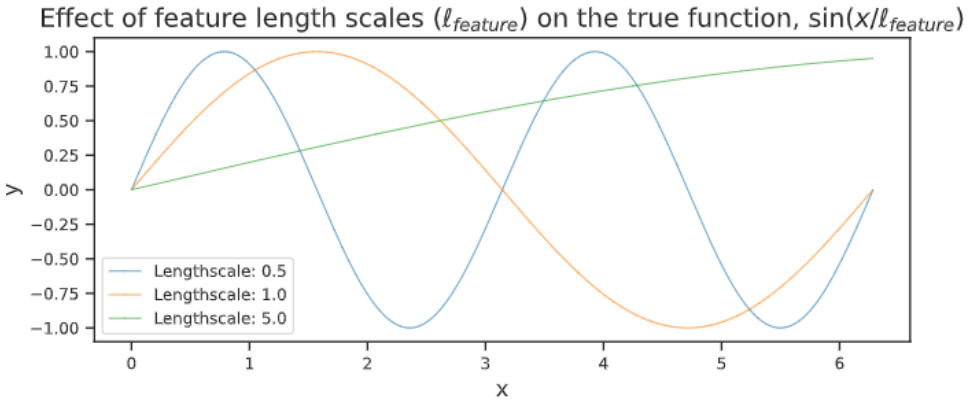

Figure 2.3: Illustration of the effect of feature length scales (ℓ_{feature}) on the true function $\sin(x/\ell_{\text{feature}})$ in Gaussian Process Regression, depicted with three curves: length scale 0.5 (blue) showing rapid oscillations, length scale 1.0 (orange) with moderate periodicity, and length scale 5.0 (green) exhibiting a near-linear trend, highlighting the influence of length scale on function smoothness.

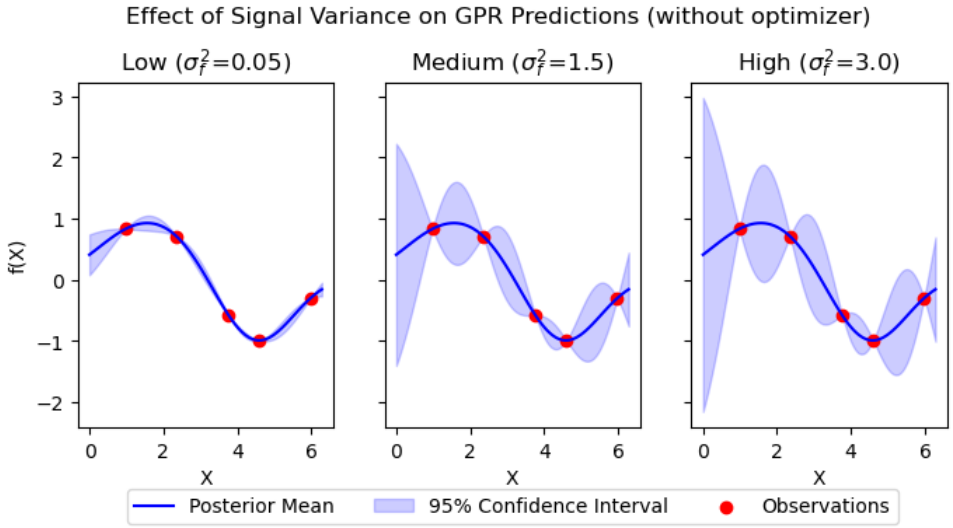

Figure 2.4: Demonstration of the effect of signal variance (σ_f^2) on GPR predictions without using any optimizer, shown in three panels: (left) low variance ($\sigma_f^2 = 0.05$) with a tight fit to observations, (middle) medium variance ($\sigma_f^2 = 1.5$) showing balanced flexibility, and (right) high variance ($\sigma_f^2 = 3.0$) with increased uncertainty, each depicting the posterior mean, 95% confidence interval, and observed data points.

Hyperparameter optimization methods

Various methods exist for optimizing hyperparameters in GPR, each offering different trade-offs between computational efficiency, accuracy, and robustness. A few of those are used in this thesis to compare the performance of the models. Such as:

- **Log Marginal Likelihood:**
  "Gaussian Processes for Machine Learning" by Rasmussen and Williams (2006) is arguably the most thorough explanation of hyperparameter optimization concerning Gaussian processes. In this literature, authors present a hyperparameter tuning approach where hyperparameters are usually chosen by maximizing the Log Marginal Likelihood (LML) or the model evidence. This approach balances model fit and intricacy of a complex model by applying Occam's razor and penalizing (Ch. 5.4) [3].

  The Log Marginal Likelihood for a Gaussian process is described as:

$$\log p(\mathbf{y} \mid \mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi \tag{2.9}$$

  Where $K_y = K(X,X) + \sigma_n^2 I$ is the covariance matrix determined by hyperparameters θ and n is the number of observations. $-\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y}$ denotes the data-fit term, while a penalty term is formulated with $-\frac{1}{2}\log|K_y|$. A normalization constant $-\frac{n}{2}\log 2\pi$ is added to the end.

  Bishop (2006) further supports Rasmussen and Williams' view by showing how Occam's razor is naturally embodied by the marginal likelihood, which automatically balances data fit and model complexity (Ch. 3 & 4.4). Additionally, the author points out that gradient-based optimization methods can effectively maximize this likelihood (Ch. 6.4.3) [5].

  Forrester et al. (2008) in "Engineering Design via Surrogate Modelling" focus on surrogate-based optimization techniques for engineering design. These techniques are especially valuable for expensive black-box functions. The authors demonstrate how GP surrogates can help find optimal hyperparameters for such expensive evaluation. Moreover, authors pointed out the benefits of automatic hyperparameter estimation using marginal likelihood, which can make reliable models without manual tweaking. Notably, the challenges related to small sampling observations (i.e., 3, 4) in a surrogate model may be inaccurate (Ch. 3.2.1) [1].

- **Conditional Log Marginal Likelihood:**
  Lotfi et al. (2022) also highlighted similar findings. Authors have pointed out that

although Occam's razor is automatically incorporated into the marginal likelihood by punishing model complexity, it can be sensitive to presumptions and may not accurately forecast generalization performance. Chapter 4.2 of the paper [6] shows how Marginal likelihood optimization for hyperparameter learning can be overfit or underfit by ignoring uncertainty or selection of the hyperparameter [6].

To overcome this problem, authors introduce the Conditional Log Marginal Likelihood (CLML), which splits observations into subsets and conditions the marginal likelihood. It helps to reduce the influence of prior assumptions and improve the model's prediction capability on unseen data. Using LML decomposition and training and testing on available and unseen data, authors formulate CLML as:

$$\sum_{i=m}^{n} \log p(D_i \mid D_{<i}, M) = \log p(D_{\geq m} \mid D_{<m}) \tag{2.10}$$

where $m \in 1, ..., n$ is the cutoff number, $D_{\leq m}$ is the set of observations from D_m to D_n. M is the given model.

In Appendix B of Lotfi et al. (2022), the authors mentioned that CLML has a dependency on datapoints ordering; thus, doing the average of CLML across all possible orderings makes it independent. Due to a large number of permutations, instead of doing all permutations, the authors suggest doing an approximation as:

$$\frac{1}{|\hat{S}|} \sum_{\sigma \in \hat{S}} \sum_{i=m}^{n} \log p(D_{\sigma(i)} \mid D_{\sigma(1)}, \ldots, D_{\sigma(i-1)}, M) \tag{2.11}$$

where $\hat{S} \subset S_n$ is a subset of several random permutations acquired from all permutations, $\sigma \in \hat{S}$ is a single permutation from that subset [6].

- **Leave-One-Out Cross Validation (LOO-CV):**
  Leave-One-Out Cross Validation (LOO-CV) is another hyperparameter optimization method. Rasmussen and Williams (2006), Forrester et al. (2008), Bishop (2006), and others discussed this method in their literature. Rasmussen and Williams (2006) presented LOO-CV based on the inverse of the kernel matrix, which allows the errors to be calculated with a single inversion. While Bishop (2006) highlighted the benefits of avoiding the over-optimism of training error, the author also warned of some setbacks, such as higher variance in error estimating due to extreme sensitivity of individual observation [1, 3, 5].

Hyperparameter search strategies

There are several search strategies for determining the best hyperparameters in GPR. Every strategy is customized to meet particular modeling requirements and computational limitations. Some of them are:

Gradient-Based Optimization: Uses analytical gradients of the log marginal likelihood (e.g., Quasi-Newton method) to find optimal hyperparameters [12].

- **Pros:** Utilizing smooth and differentiable objectives, such as those in GPR, it is quick and scalable.
- **Cons:** Requires differentiability; may converge to local optimalities.
- **Best suited for:** Gaussian Process models and moderate dataset sizes with analytically tractable likelihoods.

Bayesian Optimization: Builds a surrogate probabilistic model (e.g., Gaussian Process) of the objective function and selects new points by balancing exploration and exploitation.

- **Pros:** Sample-efficient; suitable for expensive evaluations.
- **Cons:** Overhead of maintaining the surrogate model; scales poorly with many hyperparameters.
- **Best suited for:** Expensive or black-box objective functions with limited evaluations.

Grid Search: This method evaluates all possible combinations of hyperparameter values within specified ranges.

- **Pros:** Simple to implement; exhaustive and interpretable.
- **Cons:** Inefficient in high dimensions; computationally expensive.
- **Best suited for:** Low-dimensional problems or when parameter ranges are well understood.

Random Search: Randomly samples combinations from the hyperparameter space instead of exhaustively evaluating all of them.

- **Pros:** More efficient than grid search; better exploration in high dimensions.
- **Cons:** Still requires many evaluations; lacks systematic guidance.
- **Best suited for:** High-dimensional problems where only a few parameters are critical.

.

2.6 Automatic Relevance Detection (ARD)

The conceptual underpinnings of ARD were presented in MacKay's earlier work (1998), which demonstrated how it could automatically select pertinent input dimensions. ARD was first created using distinct weight variance hyperparameters, and Neal (1996) linked it to his research on Bayesian neural networks [9, 13].

Later, Williams and Rasmussen (1996) adapted and applied ARD principles to Gaussian Processes (GPs), including dimension-specific length-scale parameters into the squared exponential kernel, for feature selection and managing high-dimensional inputs [3]. Because it skillfully incorporates feature importance assessment into the model training process without requiring additional validation data, ARD has established itself as a standard strategy for feature selection in GPR.

In the squared exponential kernel, ARD is introduced by modifying the length scale for each dimension:

$$k(x,x') = \sigma^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2\ell_d^2}\right) \tag{2.12}$$

ARD improves both interpretability and performance in high-dimensional settings, making it a valuable tool in model refinement and feature selection.

2.7 Synthetic Data

Working with *Synthetic* data has gained so much popularity among researchers nowadays due to the ability to simulate real data, augmentation, analysis, and privacy. With the help of *Synthetic* data, researchers can evaluate the model or process performance by generating datasets in a controlled setting.

According to this study in the article "Synthetic Data — what, why and how?" by James Jordon et al.[14], *Synthetic Data* is broadly explained, including potential applications and related challenges.

> "Synthetic data is data that has been generated using a purpose-built mathematical model or algorithm, with the aim of solving a (set of) data science task(s)." [14]

The article by R. L. Harrison explains how Monte Carlo methods use random sampling to approximate numerical outcomes to construct a procedure to produce synthetic data. When creating samples for stochastic (random) systems, Monte Carlo simulations frequently begin with uniform random numbers [15].

2.8 True or latent function

Sinusoidal test function

In regression and machine learning research, a sine wave test function is a frequently used synthetic function, especially for assessing how well models such as GPR perform. It has the following features and is based on the mathematical sine function:

$$y = \sin\left(\frac{x}{\ell_{feature}}\right) + \varepsilon, \tag{2.13}$$

where x is the input domain, $\ell_{feature}$ is the feature length scale that controls the frequency of the wave, and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ represents optional Gaussian noise.

CHAPTER 3

Methodology

This chapter describes the approach taken to investigate how hyperparameter tuning and automatic relevance detection (ARD) affect Gaussian Process Regression (GPR) models. The main objective is to understand the effects of various tuning techniques on input relevance identification and model performance. Python is the primary language used in the study, which uses synthetic datasets for controlled experimentation.

3.1 Gaussian process modeling framework

Gaussian Process Regressor

Several libraries and packages are available on the internet based on the Gaussian Process Regression. This package allows researchers to apply the Gaussian Process Regression to their dataset. Also, other libraries, like GPy, GPflow, GPytorch, PyStan, etc., are present based on the Python programming language. In this study, a class called 'GaussianProcessRegressor' was used from the Scikit-Learn software package for the Python language. This class is constructed based on Algorithm 2.1 from the book of Rasmussen and Williams (2006) [3]. It is used to fit and predict the training and test data, where an optimizer can be used to optimize the kernel parameters.

RBF kernel and hyperparameter settings

In this study, the Squared Exponential (SE) kernel, sometimes referred to as Radial Basis Function (RBF) kernel [10], is selected as the primary kernel for Gaussian Process Regressor due to its flexibility, effectiveness, and the capability to explain radial characteristics in the data. When enough data is available, it works well for modeling continuous, non-linear functions. The RBF kernel is perfect for hyperparameter tweaking and Bayesian optimization, especially with limited datasets, because it can also adjust to different patterns [3, 5].

In the multi-dimensional setup (starting from 2D input data), ARD is implemented by

applying separate length scales for each dimension in the kernel. In most of the experiments done in this study, a few common hyperparameter settings are used, besides those experiments where it requires changing the values, such as

- **Feature length scale of latent function:** A value of 1.0 is set to the feature length scale, where no emphasis is given on the effect of the feature length scale in the experiments by setting a value to it.
- **Initial kernel length scale:** for each dimension, a length scale 1.0 is used. For certain experiments, different values are assigned.
- **Signal variance:** Except specific scenarios where the effect of signal variance is shown, a common value of 1.0 is used with bounds of 1.0 (for both high bound and low bound).
- **Noise:** Almost no noise is added to the input data and model assumption.

Hyperparameter tuning strategies

- **Log Marginal Likelihood in Gaussian Process Regression**

  In GPR, model selection and hyperparameter optimization are typically performed by maximizing the log marginal likelihood (LML). Equations 5.8 and 5.9 in Chapter 5.4.1 of Rasmussen, C. E., & Williams, C. K. (2006) illustrate the maximization of the marginal likelihood to set the hyperparameters in a model [3].

$$\theta^* = \arg\max_\theta \log p(\mathbf{y} \mid \mathbf{X}, \theta) = \arg\max_\theta \left[-\frac{1}{2}\mathbf{y}^\top (K_\theta + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log\left|K_\theta + \sigma_n^2 I\right| - \frac{n}{2}\log 2\pi \right] \tag{3.1}$$

  where θ represents the hyperparameters (e.g., length-scales ℓ, signal variance σ_f^2, and noise variance σ_n^2), and $\mathbf{K}_\theta$ is the covariance matrix computed using a kernel function parameterized by θ

- **Conditional Log Marginal Likelihood**

  An alternative to LML is the *Conditional Log Marginal Likelihood* (CLML), which aims to optimize hyperparameters based on conditional distributions of subsets of the data. CLML is particularly useful when assessing the importance of individual features or performing structured model selection. A shuffling method is used to

generate the subsets, where random seeds (ranging from 20 to 100) of the length of the intended number of shuffles are generated. Using each seed value, random permutations of positions of the training data length are drawn to shuffle the training data based on the positions. CLML is applied to the full and shuffled input data.

$$\theta^* = \arg\max_{\theta} \text{CLML}(X, y, X_m, y_m) = \arg\max_{\theta} \left[\log p(y|X,\theta) - \frac{1}{S}\sum_{i=1}^{S} \log p(y_m|X_m,\theta) \right] \tag{3.2}$$

where the term S indicates the number of shuffled subsets. Also, X_m, y_m are the shuffled subsets of the training data. Like log marginal likelihood, a difference of $\log p(y|X,\theta)$ is calculated over full training points and shuffled subsets.

- **Leave One Out Minimizing Errors**

The *Leave-One-Out* (LOO) method evaluates generalization performance by iteratively removing one data point, training on the remaining data, and predicting the held-out point. The predictive log-likelihood or squared error is computed for each left-out observation, and the average across all n points gives a robust estimate of model performance.

The minimized sum of mean squared errors ($(\sum MSE)_{min}$) when leaving out training case i for testing is,

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{n} \left(\frac{1}{n} \sum_{j=1}^{m} \left(y_i - \hat{y}_i^{(-i)} \right)^2 \right) \tag{3.3}$$

where θ represents hyperparameters of the model (e.g., length-scale ℓ, signal variance σ_f^2, noise variance σ_n^2). θ^* represents the optimal set of hyperparameters that minimizes the LOO error. n is the total number of training data points. y_i is the true value of the i-th training output. $\hat{y}_i^{(-i)}$ depicts the predicted value of y_i when the model is trained without the i-th point. $\left(y_i - \hat{y}_i^{(-i)} \right)^2$ represents the squared prediction error for point i in leave-one-out setting.

3.2 Algorithms

GPR training and prediction algorithm

Algorithm 1 Gaussian Process Regression, by Rasmussen and Williams (2006)

Require: Training inputs $X = \{x_i\}_{i=1}^n$, training targets $y = \{y_i\}_{i=1}^n$, test inputs X_*, covariance function $k(x, x')$

Ensure: Predictive mean $\bar{f}_*$ and variance $\text{Var}(f_*)$

1: Compute the covariance matrix K such that $K_{ij} = k(x_i, x_j)$
2: $L := \text{cholesky}(K + \sigma_n^2 I)$
3: $\alpha := L^{-\top}(L^{-1}y)$
4: $\bar{f}_* := k_*^\top \alpha$
5: $v := L^{-1}k_*$
6: $V[f_*] := k(x_*, x_*) - v^\top v$
7: $\log p(y|X) := -\frac{1}{2}y^\top \alpha - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$
8: **return** $\bar{f}_*$ (mean), $V[f_*]$ (variance), $\log p(y|X)$ (log marginal likelihood)

LML optimization algorithm

Algorithm 2 Hyperparameter Tuning via Log Marginal Likelihood

Require: Input data $\mathbf{X} \in \mathbb{R}^{n \times d}$, targets $\mathbf{y} \in \mathbb{R}^n$, kernel function k_θ, noise variance σ_n^2

Ensure: Optimal hyperparameters θ^*

1: Define the kernel matrix K_θ where $[K_\theta]_{ij} = k_\theta(\mathbf{x}_i, \mathbf{x}_j)$
2: Compute the log marginal likelihood:

$$\log p(\mathbf{y} \mid \mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi$$

3: Use an optimization routine to find:

$$\theta^* = \arg\max_\theta \log p(\mathbf{y} \mid \mathbf{X}, \theta)$$

return θ^*

CLML algorithm

Algorithm 3 Compute Conditional Log Marginal Likelihood (CLML)

Require: Training data $(X_{\text{train}}, y_{\text{train}})$, kernel $k(\cdot,\cdot)$ with length scale ℓ, noise variance σ_n^2, number of shuffles S

Set cutoff size $m = \lfloor 0.2 \cdot |X_{\text{train}}| \rfloor$

Initialize GPR model with kernel k and noise variance σ_n^2

Fit GPR on $(X_{\text{train}}, y_{\text{train}})$

Compute log marginal likelihood on full data:

$$\text{LML}_{\text{full}} = \log p(y_{\text{train}} | X_{\text{train}}, \theta)$$

Initialize Total LML shuffled $\leftarrow 0$

for $i = 1$ to S **do**

    Generate random seed and shuffle indices of training data

    Select first m points: (X_m, y_m)

    Fit GPR on (X_m, y_m)

    Compute log marginal likelihood:

$$\text{LML}_i = \log p(y_m | X_m, \theta)$$

    Total LML shuffled $\leftarrow$ Total LML shuffled $+ \text{LML}_i$

end for

Compute average LML of shuffled data:

$$\text{LML}_{\text{avg\_suf}} = \frac{\text{Total LML shuffled}}{S}$$

Compute Conditional Log Marginal Likelihood:

$$\text{CLML} = \text{LML}_{\text{full}} - \text{LML}_{\text{avg\_suf}}$$

return CLML

LOO error minimization algorithm

Algorithm 4 Leave-One-Out Error Minimization

Require: Dataset (X, y) with n samples, kernel parameters θ

Initialize total error $\mathscr{E} \leftarrow 0$

for $i = 1$ to n **do**

  Exclude the i-th data point: (x_i, y_i)

  Define training data: $X_{-i} = X \setminus \{x_i\}, \quad y_{-i} = y \setminus \{y_i\}$

  Train GPR model f_{-i} using (X_{-i}, y_{-i}) and kernel parameters θ

  Predict mean $\mu_i = f_{-i}(x_i)$ and variance σ_i^2

  Compute the mean squared error for i-th point:

$$\text{MSE}_i = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

  Accumulate: $\mathscr{E} \leftarrow \mathscr{E} + \text{MSE}_i$

end for

Return sum of MSE: $\mathscr{E}$

3.3 Dataset & preprocessing

To simulate the real-world data, various generated test datasets were used in this study to understand the process of Gaussian Regression and learn to optimize hyperparameters. Different Lengthscale (ℓ)(s), different test functions, and equal or random distances within observations were used to generate the one and multidimensional test datasets. Varying the number of observations helped produce significant results for this study.

Synthetic data generation

In a controlled environment, the models are assessed using synthetic datasets. These datasets are produced in different dimensions (from 1D to 4D) using already known functions, such as sinusoidal nonlinear functions. To mimic the variety found in the real world, Gaussian noise is used when required.

Training and testing points

The fundamental task of a model is to train on training points and assess accuracy on testing points. In this study, from 3 to 50 training points (TP) are used to train up the model, while around 500 to 1000 test points are used to evaluate the model accuracy. Also, this high number of points helps to visualize at a higher resolution.

Sampling domain

Input samples are randomly selected from the [0,2π] domain. This domain is selected because the objective function for simulation is a sine function, which is periodic with a fundamental period of 2π. By using this interval, the entire sine wave cycle is recorded, resulting in representative behavior of the function during its period. The model's capacity to generalize over known function structures and learn periodic patterns is best assessed in this environment. However, a few experiments in this study show that it creates an aliasing effect when the number of samples is relatively low (between 3 and 5) or when some samples are positioned at the beginning or end. To avoid this problem, a uniform distribution range was adjusted to $[0+0.5, 2\pi-0.5]$ when needed.

Monte Carlo sampling

Monte Carlo sampling is used to create the input data from a uniform distribution across the input domain for the surrogate models' training and testing [16]. This method prevents bias in the function approximation and guarantees generality.

The input values (X) are sampled uniformly from the interval [0, 2π]. This domain ensures that the periodic nature of the sine function is fully captured. The inputs are then sorted to aid in visualization or interpolation. However, a few experiments in this study showed that it created an aliasing effect when the number of samples was minimal or some samples were at the beginning or end. To avoid this problem, a uniform distribution range was adjusted to $[0+0.5, 2\pi-0.5]$ where needed.

True or latent function:
The target outputs ($f(X, \ell_{feature})$) are generated using a sinusoidal function that was used with a scaling factor named Feature Length Scale of the Underlying Data ($\ell_{feature}$).

$$f(X, \ell_{feature}) = \sin\left(\frac{X}{\ell_{feature}}\right) \tag{3.4}$$

where

- X is the independent observations or training points.
- $\ell_{feature}$ is the length scale of the underlying data (controlled synthetic input feature).

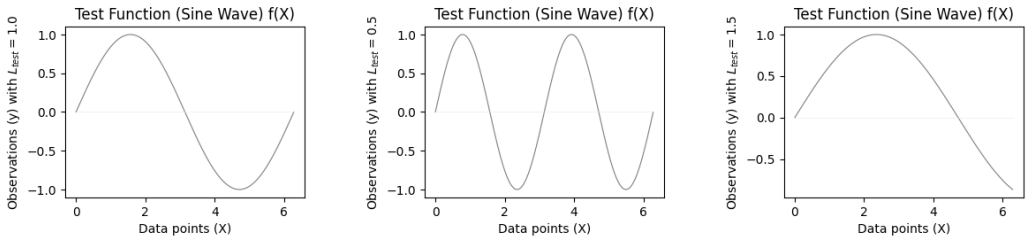

Figure 3.1: The plot displays three sine wave functions with varying amplitudes. Each subplot illustrates a sine wave across the sampling domain. Furthermore, a scaling factor named $\ell_{feature}$ controls the wavelength or the frequency of the periodicity.

In the Figure 3.1, left, center, and right subplots show a sine wave with $\ell_{feature}$ values of 1.0, 0.5, and 1.5, respectively, demonstrating how the sine wavelength changes proportionally to the $\ell_{feature}$ value.

The feature length scale of the underlying data is essential for managing the true function's variability. It makes it possible to modify the trends of the input data, offering a versatile and regulated setting for creating synthetic data with various properties. Experiments can yield datasets with a variety of trend behaviors by altering the feature length scale ($\ell_{feature}$). By suitably modifying their hyperparameters, models trained on these samples can then learn the underlying patterns by appropriately adjusting their hyperparameters [5].

3.4 Evaluation Metrics

The following metrics were used to evaluate model performance:

- **Negative Log Marginal Likelihood (NLML)**: Used as the objective for LML and CLML. Minimizer tries to minimize the NLML, which maximizes the Log Marginal Likelihood.
- **Root Mean Squared Error (RMSE)**: Measures prediction accuracy on the test set.
- **Mean Absolute Error (MAE)**: Provides a more interpretable error measure.
- **Learned length scale:** Compare models on how well they learned the input feature length scale.
- **ARD Lengthscales:** Used to rank the relevance of each input dimension.

CHAPTER 4

Results and Discussions

This chapter reports on experiments done to assess the performance of Gaussian Process Regression (GPR) models constructed by three optimization methods, LML, CLML, and LOO. Its main topics are Automatic Relevance Determination (ARD) and hyperparameter tuning using various hyperparameter optimization techniques. The aim is to evaluate how ARD affects GPR's interpretability and predictive accuracy and how various optimization techniques enhance performance.

4.1 One-dimensional hyperparameter tuning

Model performance evaluation with different observation sizes

This experiment assesses the predictive errors and learned length scale of optimization methods over various numbers of training points and noise levels.

Table 4.1 compares Root Mean Squared Error (RMSE) and Mean Average Error (MAE) for LML, CLML, and LOO methods (average of 10 runs) among increasing training points (TP = 10, 25, 50,100) with feature length scale of 1.0, signal variance of 1.0, number of restarts of 1, and almost no noisy data. According to the result, LML consistently achieved the lowest RMSE and MAE error values, particularly where TP were less (around 10). With a larger number of TP (more than 25), all methods converge to low RMSE and MAE error levels.

Table 4.2 compares the learned length scales for the LML, CLML, and LOO methods, using a true feature length scale ($\ell_{feature}$) of 1.0, in both noiseless and noisy scenarios (noise levels of 0.1). In the absence of noise, CLML and LOO converge toward the true feature length scale ($\ell_{feature}$) as the number of training points (TP) increases, whereas LML consistently overestimates the length scale, indicating unduly smooth function fits. All approaches tend to overestimate the learned length scale ($\ell_{learned}$) when noise is added; LOO stays more conservative, while LML and CLML approach values close to 1.77. This demonstrates how noise, particularly for LML and CLML, causes the models to interpret the underlying function as smoother.

Table 4.1: Performance comparison of model selection methods (LML, CLML, LOO) with varying number of training points.

Models' performance			
$\ell_{feature}$ = 1.0, Signal Variance = fixed, Noise=1e-10			
Method	**TP**	**RMSE**	**MAE**
LML	10	0.007654748846002417	0.0032475816526273725
	25	7.868033887070962e-05	2.3175824445591902e-05
	50	2.5539376268566242e-05	7.26487183731833e-06
	100	1.1399874637132645e-06	7.856934095710494e-07
CLML	10	0.03193955936502695	0.015304528863826886
	25	0.006112697941251576	0.0019735994513036424
	50	0.0009695078412425359	0.00032828407920262485
	100	5.810412165240707e-05	3.3712669171100434e-05
LOO	10	0.06441228805671287	0.02983249572073058
	25	0.004889201034083178	0.0015926262230617281
	50	0.0007514407910041975	0.0002294331073810647
	100	5.810271735398184e-05	3.371157080158845e-05

Table 4.2: Learned Length Scale ($\ell_{learned}$) by model selection method and number of training points under no noise and added noise conditions.

Models' captured length scale			
$\ell_{feature}$ = 1.0, Signal Variance = fixed			
Method	**TP**	**Learned Lengthscale (No Noise)**	**Learned Lengthscale (Noise = 0.1)**
LML	10	2.21359821957543	1.6862319955084466
	25	2.2677487543450696	1.6915396718245332
	50	2.3138344119741827	1.742200501800853
	100	2.3366260965370373	1.7714383647418939
CLML	10	2.142828335448164	1.7260308749405557
	25	1.050164608266219	1.7188033004378416
	50	1.287520071859623	1.7740613940858108
	100	1.0000093099870722	1.7798498872068969
LOO	10	1.316590119200558	1.4260246216416486
	25	1.5192539507939942	1.0350522809262366
	50	1.3674334911624277	1.1633541219465222
	100	1.0000044303882052	1.3310662920366763

Model performance with different hyperparameter settings

The model's ability to learn the $\ell_{feature}$ is displayed in line plots for three methods over different settings of hyperparameters, such as several training points, noise levels, number of restarts, various feature length scales, signal variances, etc.

Learned Length Scale Over Number Of Points

Figure 4.1 contrasts the behavior of various approaches to Gaussian Process Regression's length scale learning as training data increases (from 2 to 40). The conventional Log Marginal Likelihood method (Plot a) produces stable but consistently overestimated values, which converge above 2. However, with small datasets, the Conditional Log Marginal Likelihood (Plot b) and Leave-One-Out Cross-Validation (Plot c) approaches exhibit greater variability, although they converge closer to the feature length scale of 1.0, at about 1.5. This implies that these alternative approaches might be more resistant to overestimation, despite being more unpredictable initially. Every method exhibits sensitivity to dataset size, particularly when there are only a few data points (roughly 10).

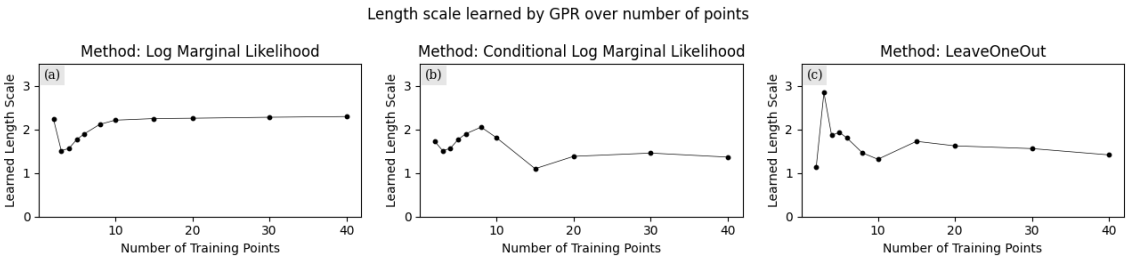

Figure 4.1: Behavior of length scale estimation across methods as training data increases

Experiments using different levels of noise

Figure 4.2 illustrates how the learned length scale in GPR behaves across three optimization approaches as the training size increases (up to 40 points) and under varying Gaussian noise levels (σ_n^2 = 1e-10, 0.01, 0.1, 0.2). Given that the input feature length scale ($\ell_{feature}$) is 1.0, and the signal variance (σ_f^2) is fixed (variance 1.0 with bounds (1.0, 1.0)). The LML approach overestimates the true value even at low noise levels, demonstrating a sharp rise in the learned length scale with small sample sizes before stabilizing between 2.2 and 2.6. LOO exhibits high variance at small sample sizes but stabilizes at 20 points. LML consistently overestimates the true length scale, even though it provides stable estimates. On the other hand, conditional LML and LOO generate values that are more reliable when applied to realistic data because they are closer to the true parameter.

In general, as the quantity of training points grows, the learned length scale tends to stabilize, suggesting that more data provides a more consistent estimate of the underlying function's smoothness. The effect of noise on the learned length scale is also apparent, though the specific trends vary across the different model selection methods. This analysis highlights the importance of considering these factors when applying GPR to real-world

datasets.

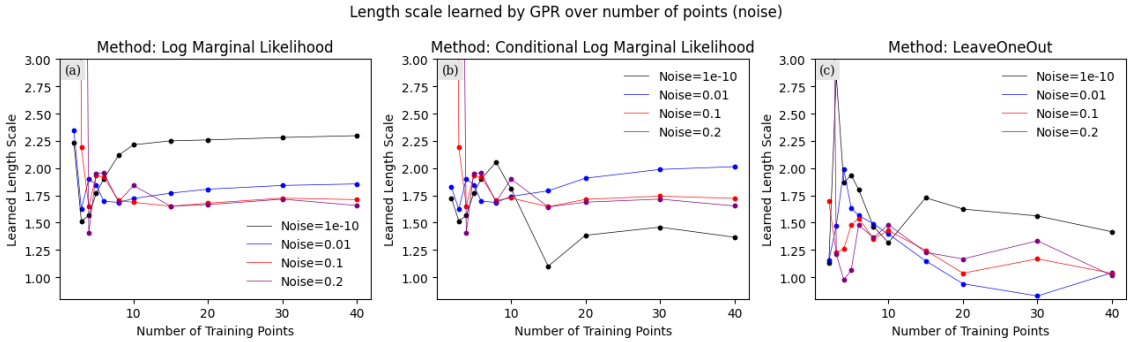

Figure 4.2: Learned length scale ($\ell_{learned}$) in GPR as a function of the number of training points and noise level, for three model selection methods. Subplots (a), (b), and (c) show results for LML, CLML, and LOO, respectively. Within each subplot, the learned $\ell_{learned}$ is plotted for four noise levels: 10^{-10} (black), 0.01 (blue), 0.1 (red), and 0.2 (purple). Each line connects data points representing the learned length scale for a given number of training points at the specified noise level.

Experiments using different numbers of restarts

Figure 4.3 shows how the learned length scale in GPR changes with the number of optimization restarts for three methods. LML is stable across different numbers of restarts, suggesting minimal benefit from multiple initializations. Conditional LML shows slight improvements in stability with more numbers of restarts, especially at lower sample sizes. LOO is the most sensitive to the number of restarts, with high variance at small sample sizes and notable stabilization as the number of restarts increases. At 10 number of restarts, LOO yields more consistent length scale estimates. These numbers of restarts improve robustness in Conditional LML and LOO, but have a limited effect on LML.

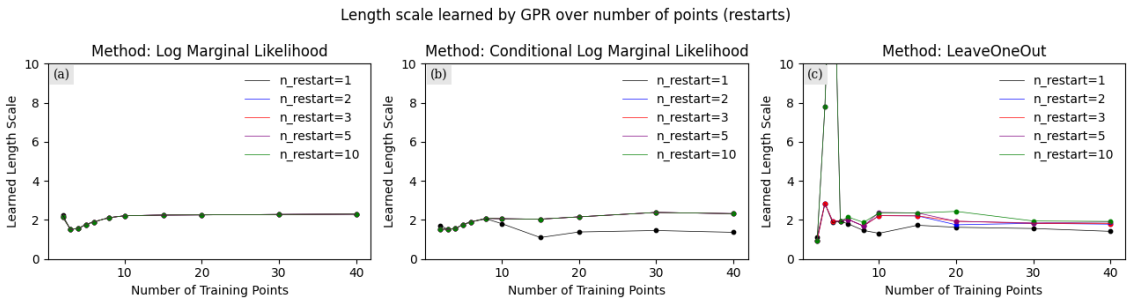

Figure 4.3: illustrates learned length scale ($\ell_{learned}$) of a GPR model as a function of the number of training points, for different numbers of optimization restarts. Subplots (a), (b), and (c) display results for the LML, CLML, and LOO methods, respectively. Within each subplot, lines represent different numbers of optimization restarts: 1 (black), 2 (blue), 3 (red), 5 (pink), and 10 (green).

Experiments using different feature length scales

Figure 4.4 illustrates how the learned length scale $\ell_{learned}$ in GPR varies with the true feature length scale $\ell_{feature}$ for each of the three estimation methods. When $\ell_{feature}$ is long (e.g., 10.0), all methods initially overestimate $\ell_{learned}$, particularly when the number of training points is limited. LML tends to overestimate early but stabilizes as more data becomes available. CLML shows more pronounced fluctuations and persistent overestimation, especially for long $\ell_{feature}$. LOO initially produces highly variable estimates, but its behavior becomes more stable and moderate with additional training points. Across methods, the precise recovery of $\ell_{feature}$ becomes increasingly difficult as its true value increases. The results suggest a strong influence of $\ell_{feature}$ on the learned length scale; models trained on smoother (longer-scale) features tend to infer smoother functions, especially with limited data.

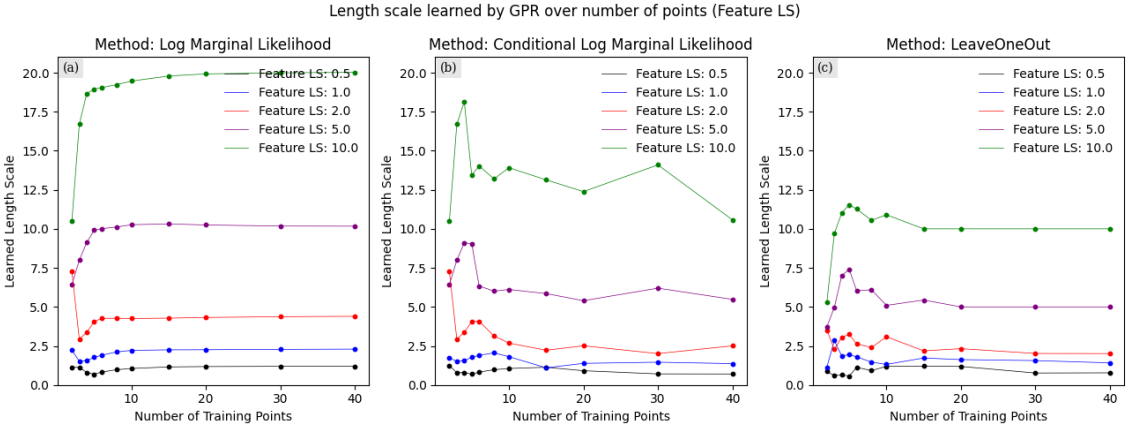

Figure 4.4: Learned length scale ($\ell_{learned}$) of a GPR model as a function of the number of training points, for different values of a feature length scale parameter, keeping noise to a minimum and variance fixed. Within each subplot, lines represent different feature length scale values: 0.5 (black), 1.0 (blue), 2.0 (red), 5.0 (purple), and 10.0 (green).

Experiments using different signal variances

In Figure 4.5, LML (panel a) produces consistent and interpretable length-scale estimates across different signal variances ($\sigma_f^2 = 0.5, 1.0, 3.0, 5.0$). As the number of training points increases, estimates stabilize around 15–20 points, showing early convergence. At TP = 40, LML maintains a clear separation between signal variances, indicating strong sensitivity to the underlying signal-to-noise structure. This makes LML statistically robust for hyperparameter tuning. In contrast, CLML (panel b) yields highly unstable results, length scale estimates vary unpredictably and often overlap across σ_f^2 values, even with more data. This suggests poor reliability and weak sensitivity to signal variance. LOO

(panel c) shows better performance than CLML, but still suffers from high variability with fewer than 15 training points. While estimates stabilize beyond 30 points, they tend to converge toward similar values, reducing signal variance distinction.

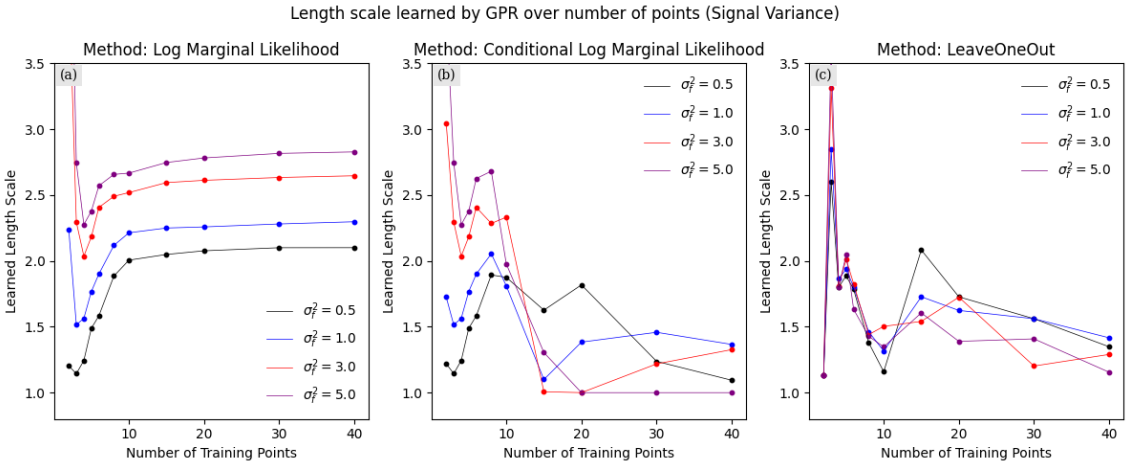

Figure 4.5: The plots illustrate how the learned length scale of a GPR model changes with an increasing number of training points for varying signal variance (σ_f^2). Three different estimation methods are compared: (a) Log Marginal Likelihood, (b) Conditional Log Marginal Likelihood, and (c) LeaveOneOut. Each line represents a different signal variance (σ_f^2) setting ranging from 0.5 to 5.0.

Experiments using different initial kernel length scales

In Figure 4.6, across three methods, the learned length scale converges close to the feature length scale with more training data, but the influence of the initial kernel length scale has less effect on all three methods. As all the methods optimize the learned length scales, the initial kernel length scale has minimal effect on learning the length scales.

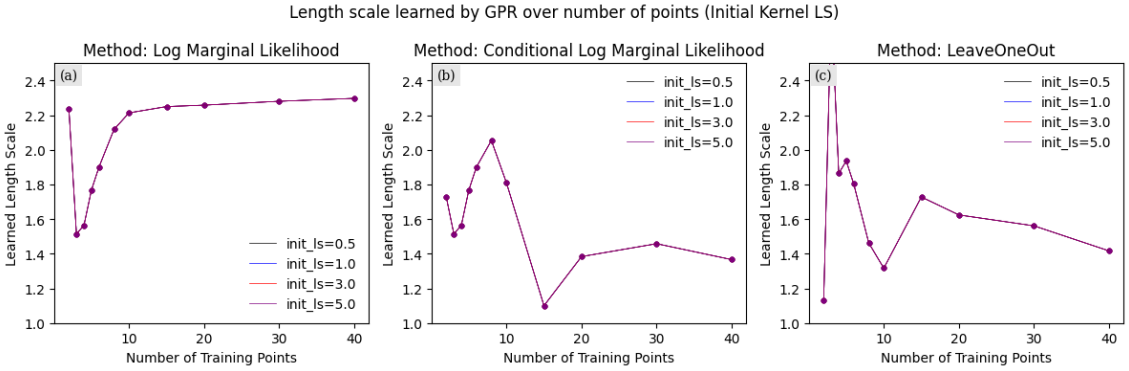

Figure 4.6: Length scale learned by GPR over varying numbers of training points with different initial kernel length scale (ℓ_{init}) settings. Each line represents a different initial length scale value ranging from 0.5 to 5.0.

Learned Length Scale Over Number Of Restarts

The plots in Figure 4.7 show that for LML (a), the learned length scale quickly stabilizes around 2.2 and remains consistent across restarts. CLML (b) also shows rapid stabilization around 2.2 after a few restarts. Similarly, Leave-One-Out (c) converges to approximately 2.3 after an initial increase within the first few restarts. Increasing the number of restarts beyond a small number (around 5-10) does not significantly alter the learned length scale for these methods.

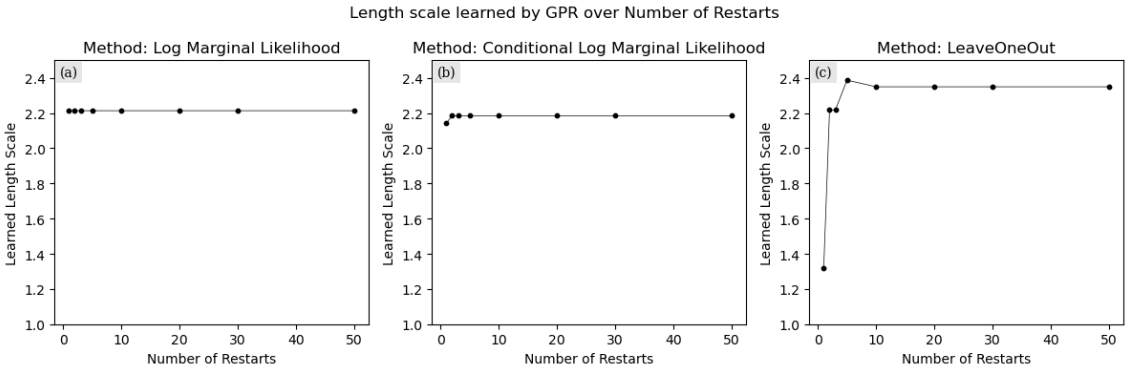

Figure 4.7: Length scale learned by GPR over varying numbers of training points with varying number of restart settings. Each black represents a different number of restarts, ranging from 2 to 50.

4.2 Effect of ARD on feature importance in multi-dimensional Gaussian Process Regression

Learned length scales of two input features

Table 4.3 illustrates how the three GPR optimization approaches learned the length scales of two input features over various TP. All methods show significant overestimation of length scales when the dataset is minimal (e.g., at 2–5 TP), especially with LML, where values exceed 50. As the number of TP increases, the estimates stabilize and approach the actual values. The convergence trend is consistent across all three methods, ranging from 1.8 to 3, though LML generally exhibits higher variability at low sample sizes.

Table 4.3: Learned length scales ($\ell^1_{learned}$,$\ell^2_{learned}$) for two input features over varying numbers of training points (TP) using three inference methods: LML, CLML, and LOO. The true feature length scales are set to (1.0,1.0), with no signal variance and a microscopic noise level (10^{-10}), under an RBF kernel.

ARD of two features (Noise: 1e-10, Signal Variance: none, Kernel: RBF, $L_{feature}$: (1.0, 1.0))						
TP	**LML**		**CLML**		**LOO**	
	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$
2	8.2809	1.1046	0.9695	1.0001	0.9623	0.8793
3	55.0061	31.2800	50.7345	24.4833	38.0151	21.8486
4	23.4654	11.3973	32.3006	11.3954	33.1353	11.8402
5	5.1865	13.7659	4.8776	13.8908	14.7603	14.4414
6	2.9308	3.3722	2.9308	3.3721	3.3028	11.5893
7	2.4847	2.2210	2.2004	2.5085	2.3476	3.5714
8	1.9254	2.0128	1.9254	2.0128	1.9630	3.6305
9	2.1579	1.9340	2.1579	1.9340	2.6054	1.9722
10	2.3012	1.9860	2.3029	1.9869	2.9535	2.2680
20	2.3426	2.3487	2.3534	2.3589	2.6074	2.8265
30	2.5013	2.5126	2.5238	2.5426	2.5070	2.5726
40	2.5784	2.5677	2.6458	2.6139	2.5728	2.4948

Visualizing Input Relevance Learned by ARD

The relative feature relevance, as assessed by ARD, is visually compared in these bar charts 4.8. Substantial differences are observed in low-observation settings, particularly for LML and CLML, where $\ell^1_{learned}$ significantly predominates, signifying uncertainty in feature selection. As data grows, both length scales shrink and level out around the true value. The bar format supports the steep drop in overestimation and the convergence of

learnt hyperparameters for all methods.

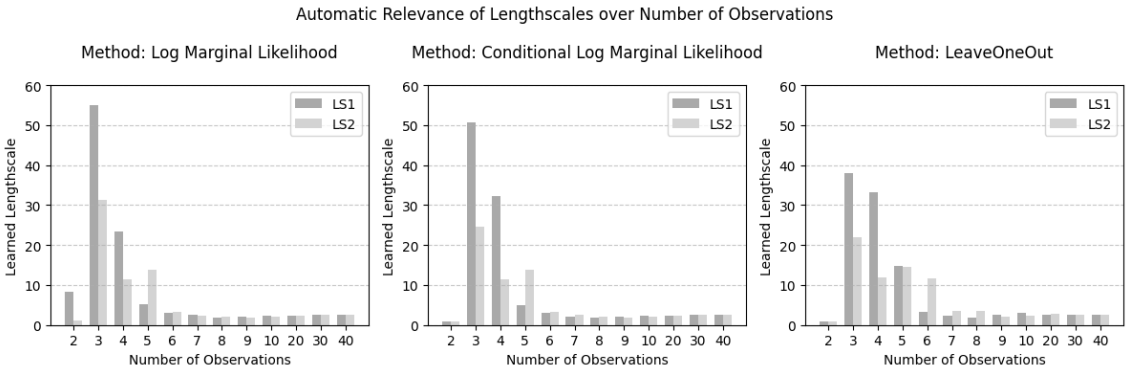

Figure 4.8: Bar plots illustrating the learned length scales ($\ell^1_{learned}$, $\ell^2_{learned}$) across different numbers of training points using three methods. Each subplot compares the magnitudes of the two learned length scales per observation count.

Effects of varying feature length scale two and number of training points on learned length scales

Table 4.4 explores how learned length scales $\ell^1_{learned}$ and $\ell^2_{learned}$ in GPR's ARD framework, which indicate feature relevance, vary with training data and feature length scales. For LML, $\ell^1_{learned}$ ranges from 1.76 to 55.01, demonstrating substantial variation in feature 1's relevance across conditions. Similarly, CLML shows a range of 0.96 to 45.24 for $\ell^1_{learned}$. LOO exhibits the widest range for $\ell^2_{learned}$ (0.88 to 50.49), suggesting greater instability in estimating feature 2's influence with this method, especially with fewer data points.

As the number of training points (TP) increases from 3 to 20, the learned length scales generally tend to stabilize, converging towards more consistent values. This indicates that more data leads to improved and more reliable estimation of feature relevance. However, the rate and extent of this stabilization vary across the three model selection methods.

Furthermore, the feature length scale parameter $\ell_{feature}$ significantly influences the learned length scales. For instance, when $\ell_{feature}$ is small (e.g., 0.5), the learned length scales can exhibit greater variability, implying that the model assigns more distinct relevance to the two features. Conversely, as $\ell_{feature}$ increases (e.g., to 3.0), the learned length scales tend to become more similar, suggesting that the model considers the two features to be more equally relevant.

Table 4.4: Learned length scales ($\ell^1_{learned}$,$\ell^2_{learned}$) for two input features under ARD in GPR, evaluated using three model selection methods: LML, CLML, and LOO. The table shows results for varying numbers of training points (TP) and different feature length scale parameters of one dimension $\ell^2_{feature}$ while keeping the other dimension $\ell^1_{feature}$ fixed to 1.0. The GPR model employs the RBF kernel, with a noise level of 10^{-10} and no signal variance.

ARD of two features (Noise: 1e-10, Signal Variance: none, Kernel: RBF, $L_{feature}$1: 1.0)							
TP	$\ell^2_{feature}$	**LML**		**CLML**		**LOO**	
		$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$
3	0.5	6.0316	35.2228	1.8711	28.7998	11.4776	16.1870
	1.0	55.0061	31.0756	45.2430	25.7052	47.4489	21.8523
	2.0	26.0613	36.2234	21.7561	36.2510	23.2565	41.9731
	3.0	36.1075	23.7987	31.7970	23.8270	33.1728	41.6348
4	0.5	11.1987	52.1904	11.1987	52.2030	11.3425	38.8321
	1.0	23.4654	11.3973	31.0460	11.3954	33.1342	11.8400
	2.0	11.7847	24.1009	11.7847	24.1005	14.0257	22.9416
	3.0	11.8742	29.3687	11.8742	29.3686	23.8165	50.4921
5	0.5	24.6916	1.2433	24.6916	1.2433	20.8833	22.0024
	1.0	5.1865	13.7659	4.8776	13.8908	14.7535	14.4407
	2.0	4.0563	24.9554	4.6273	15.2408	12.4282	32.5020
	3.0	4.5390	27.9107	4.5388	27.9106	2.6806	16.2728
6	0.5	1.7560	21.1730	1.7201	21.2437	1.7396	21.2040
	1.0	2.9308	3.3722	2.9308	3.3722	3.3029	11.5893
	2.0	1.8060	15.5062	1.9492	5.8767	5.6786	14.8991
	3.0	1.8905	9.7843	1.8905	9.7843	1.9492	11.5698
8	0.5	2.3004	2.1469	2.1736	1.1516	1.7469	3.5296
	1.0	1.9254	2.0128	1.9254	2.0128	1.9630	3.6305
	2.0	1.9010	7.5481	1.9010	7.5482	3.2716	10.4824
	3.0	2.0041	7.1168	2.0041	7.1168	2.4410	17.0115
10	0.5	3.4084	0.9116	3.4499	0.9123	4.1003	0.9343
	1.0	2.3012	1.9860	2.3029	1.9869	2.9535	2.2681
	2.0	2.0988	4.1023	2.0971	4.0924	2.5193	4.8339
	3.0	2.1560	6.4037	2.1547	6.4042	2.9158	16.3092
15	0.5	2.0171	1.0067	2.0179	1.0067	2.1211	1.0724
	1.0	2.1861	2.1979	2.1890	2.2005	2.2601	2.1436
	2.0	2.3647	4.5312	2.3668	4.5291	2.7062	4.9166
	3.0	2.4163	6.7355	2.4192	6.7340	2.6372	6.6868
20	0.5	2.1232	1.0905	2.1311	1.0954	2.2287	1.2062
	1.0	2.3426	2.3487	2.3534	2.3589	2.6076	2.8265
	2.0	2.4600	4.7676	2.4791	4.7946	2.1728	4.7178
	3.0	2.4839	7.0607	2.5056	6.9251	2.2706	6.4757

Effects of varying noise level ($\sigma^2 n$) and number of training points on learned length scales

Table 4.5 shows how variations in noise variance (σ_n^2) and the number of training points (TP) influence the learned length scales (ℓ^1_{learned} and ℓ^2_{learned}) in Gaussian Process Regression (GPR) with Automatic Relevance Determination (ARD). Both feature length scales (ℓ^1_{feature} and ℓ^2_{feature}) are fixed at 1.0, allowing the analysis to isolate the effects of noise and data quantity on the inferred feature relevance. The results indicate that the learned length scales are sensitive to noise variance. For example, under the LML criterion, ℓ^1_{learned} ranges from 2.05 to 36.01, reflecting how the model's perception of feature 1's relevance shifts with noise. A similar sensitivity is observed for ℓ^2_{learned}, implying that increased noise can obscure or exaggerate a feature's importance.

An increase in training points generally contributes to more stable estimates of the learned length scales, though the degree of stabilization varies by method. In the case of Leave-One-Out (LOO), substantial fluctuations are observed for small training sets (TP = 4–6), which diminish as more data is introduced. This trend suggests that larger datasets enhance the robustness of relevance estimation, particularly for methods like LOO that are sensitive to training data perturbations. The analysis highlights that noise variance significantly impacts length scale estimation in GPR with ARD. Furthermore, increasing the number of training points can mitigate this effect, leading to more consistent and interpretable relevance assessments, especially when the true feature length scales are known and fixed.

Table 4.5: Learned length scales ($\ell^1_{learned}$,$\ell^2_{learned}$) for two input features under ARD in GPR, evaluated using three model selection methods: LML, CLML, and LOO. The table shows results for varying numbers of training points (TP) and different noise levels (σ_n^2), keeping feature length scale parameters of both dimension $\ell^2_{feature}$ and $\ell^1_{feature}$ fixed to 1.0. The GPR model uses a Radial Basis Function (RBF) kernel, with no signal variance.

ARD of two features (Signal Variance: none, Kernel: RBF, ($\ell^1_{feature}$: 1.0, $\ell^2_{feature}$: 1.0))							
TP	$\sigma^2 n$	**LML**		**CLML**		**LOO**	
		$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$
4	1e-05	32.3006	11.3955	32.3006	11.3955	33.1348	11.8401
	0.01	23.0281	11.4384	27.4236	11.4305	13.4314	11.8410
	0.1	36.0114	30.9881	31.7120	31.0666	21.8995	11.2135
	0.2	21.5146	41.0586	21.5170	41.0208	30.7336	11.3967
5	1e-05	5.2418	13.7670	4.9328	13.8918	14.7588	14.4414
	0.01	3.1850	22.8643	3.1850	22.8642	4.5268	14.4755
	0.1	13.2370	31.7362	13.2370	31.7361	2.8210	13.7162
	0.2	4.9363	50.8201	4.9358	49.9821	2.4362	30.6263
6	1e-05	2.9315	3.3747	2.9315	3.3747	3.3029	11.5893
	0.01	3.4482	12.2990	3.4482	12.2990	2.6255	12.5127
	0.1	4.0505	13.0258	4.0505	13.0258	2.2694	21.4720
	0.2	6.9305	22.4732	6.9292	22.4732	2.2895	21.6449
8	1e-05	1.9242	2.0200	1.9242	2.0200	1.9626	3.6303
	0.01	11.7266	2.0202	1.9052	2.1544	1.6344	2.3769
	0.1	12.2903	3.0008	12.4051	2.4183	11.4536	1.7477
	0.2	22.3408	4.4845	22.3408	4.4839	11.8434	1.7345
10	1e-05	2.3159	1.9843	2.3178	1.9851	2.9530	2.2679
	0.01	2.7821	2.0473	2.8180	2.0402	2.8039	1.8599
	0.1	4.1235	2.0960	3.5659	2.1284	1.9366	1.6250
	0.2	23.1421	11.9042	16.2727	2.5230	1.9117	1.4066
15	1e-05	2.1822	2.1909	2.1851	2.1934	2.2601	2.1435
	0.01	2.0580	2.0772	2.0596	2.0788	1.7285	1.7350
	0.1	2.4401	2.6595	2.4208	2.7248	1.6262	1.4804
	0.2	2.6451	4.0330	2.3562	4.2754	1.6979	1.4601
20	1e-05	2.3174	2.3194	2.3285	2.3298	2.6077	2.8265
	0.01	2.1081	2.1088	2.1104	2.1083	1.8182	1.7976
	0.1	2.2653	2.1075	2.4236	2.0881	1.5721	1.5560
	0.2	3.2650	2.4941	3.2380	2.4635	1.4766	1.4768

Effects of varying signal variance (σ_f^2) and number of training points on learned length scales

Table 4.6 presents learned length-scales (ℓ^1_{learned}, ℓ^2_{learned}) generally decrease with increasing TP (number of training points), suggesting improved identification of feature relevance as more data becomes available. For instance, under LML with $\sigma_f^2 = 0.5$, ℓ^1_{learned} drops from 14.44 (TP=4) to 2.04 (TP=20), indicating a shift from perceived irrelevance to relevance. This pattern holds across all methods, though the rate and extent vary. LML and CLML often yield similar results, but can diverge significantly under certain conditions (e.g., TP=8, $\sigma_f^2 = 2.0$: LML = (9.04, 2.27), CLML = (2.37, 2.42)), showing that CLML's approximation may break down in complex cases (where the GP posterior becomes too sharp, noisy, or underdetermined). LOO tends to overestimate length-scales, especially at low TP, suggesting a more conservative relevance estimate. For example, at TP=4, LOO gives $\ell^1_{\text{learned}} \approx 33$, compared to LML/CLML values around 13–14. As TP increases, LOO values converge closer to those from LML and CLML, but often remain larger.

Across all methods, ℓ^{learned} more than 1.0 implies weak inferred relevance. For TP ≥ 8, values typically fall to 1.5–3.5, indicating improved but imperfect identification of the true relevance. Importantly, optimization criteria can lead to differing conclusions (e.g., TP=8, $\sigma_f^2 = 2.0$: LML suggests feature 2 is more relevant, CLML sees both as equally relevant, LOO favors feature 1). Signal variance (σ_f^2) also interacts with TP and the optimizer, influencing length-scales non-monotonically. For example, under LML at TP=4, increasing σ_f^2 from 0.5 to 3.0 raises ℓ^2_{learned} from 11.11 to 22.89.
Finally, ARD-based relevance inference is sensitive to data availability, signal variance, and the optimization method. While LML and CLML diverge in key settings, LOO offers conservative estimates but converges with more data.

Table 4.6: Learned length scales ($\ell^1_{learned}$,$\ell^2_{learned}$) for two input features under ARD in GPR, evaluated using three model selection methods: LML, CLML, and LOO. The table shows results for varying numbers of training points (TP) and different signal variances (σ_f^2), keeping feature length scale parameters of both dimension $\ell^2_{feature}$ and $\ell^1_{feature}$ fixed to 1.0. The GPR model uses a Radial Basis Function (RBF) kernel, with a microscopic noise of 10^{-10}.

ARD of two features (Noise: 1e-10, Kernel: RBF, ($\ell^1_{feature}$: 1.0, $\ell^2_{feature}$: 1.0))							
TP	$\sigma^2 f$	**LML**		**CLML**		**LOO**	
		$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$
4	0.5	14.4422	11.1134	14.2886	11.1488	33.0983	11.8363
	2.0	13.4436	12.5090	13.4436	12.5090	33.1560	11.8426
	3.0	13.1550	22.8922	14.1320	13.0696	33.1616	11.8430
5	0.5	12.2089	12.6961	12.2089	12.6961	14.0612	14.3762
	2.0	9.3837	5.4177	6.2453	15.3148	15.6402	14.5151
	3.0	11.2971	6.4512	7.4100	16.3369	16.2131	14.5603
6	0.5	1.7087	12.4739	1.7087	12.4739	3.3016	9.6227
	2.0	4.8567	3.2604	4.3072	3.3666	3.3035	11.5891
	3.0	5.2101	3.9146	5.2101	3.9145	3.3038	11.5891
8	0.5	1.5644	1.6454	1.5644	1.6454	1.9402	3.6195
	2.0	9.0355	2.2736	2.3714	2.4193	1.9791	3.6359
	3.0	2.6720	2.6756	2.6720	2.6756	1.9849	3.6373
10	0.5	1.9112	1.6719	1.9213	1.6787	3.0482	2.2682
	2.0	2.7417	2.3191	2.7352	2.3145	2.9473	2.2608
	3.0	3.0137	2.5240	3.0026	2.5166	2.9518	2.2594
15	0.5	1.8875	1.9015	1.8990	1.9124	2.2213	2.1138
	2.0	2.4853	2.4852	2.4812	2.4815	2.2891	2.1670
	3.0	2.6608	2.6498	2.6536	2.6433	2.3022	2.1778
20	0.5	2.0383	2.0514	2.0630	2.0751	2.4292	2.6825
	2.0	2.6053	2.6095	2.6066	2.6099	2.6292	2.6647
	3.0	2.7454	2.7494	2.7426	2.7453	2.6842	2.6695

Effects of varying number of restarts and number of training points on learned length scales

Table 4.7 investigates the influence of optimization restarts (2–10) on ARD-estimated length-scales in a GPR model with fixed signal variance and near-zero noise. Using LML, CLML, and LOO criteria across varying training points (TP), it was observed that CLML produced highly stable length-scales regardless of restart count. In contrast, LML and LOO showed sensitivity, particularly LML at TP=8, where ℓ^1_{learned} increased from 1.93 (2 restarts) to 9.51 (10 restarts). Similarly, LOO at TP=6 saw large swings in both ℓ^1_{learned} and ℓ^2_{learned}, indicating susceptibility to local optima.

Table 4.7: This table analyzes an ARD study for a Gaussian Process Regression model with two features. The model employs an RBF kernel, assumes a very low noise level (10^{-10}), and uses a fixed signal variance. The true feature length-scales (ℓ^1_{feature}, ℓ^2_{feature}) are both set to 1.0. The goal of this investigation is to evaluate how varying the number of optimization restarts (from 2 to 10) influences the learned length-scales (ℓ^1_{learned}, ℓ^2_{learned}). This is examined under three different hyperparameter optimization techniques across multiple levels of TP.

ARD of two features (Noise: 1e-10, Signal Variance: fixed, Kernel: RBF, ($\ell^1_{feature}$: 1.0, $\ell^2_{feature}$: 1.0))							
TP	**Restarts**	**LML**		**CLML**		**LOO**	
		$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$	$\ell^1_{learned}$	$\ell^2_{learned}$
4	2	22.3393	21.3693	31.0460	11.3954	33.1342	11.8400
	3	22.3393	21.3693	31.0460	11.3954	33.1342	11.8400
	4	22.3393	21.3693	31.0460	11.3954	23.9971	11.9557
	5	22.3393	21.3693	31.0460	11.3954	23.9971	11.9557
	10	22.3393	21.3693	31.0460	11.3954	23.9976	11.9558
5	2	5.1865	13.7659	4.8776	13.8908	24.5621	13.6252
	3	5.1865	13.7659	4.8776	13.8908	24.5621	13.6252
	4	5.1865	13.7659	4.8776	13.8908	24.6635	13.8834
	5	5.1865	13.7659	4.8776	13.8908	24.6635	13.8834
	10	5.1865	13.7659	4.8776	13.8908	32.6709	16.8669
6	2	2.9308	3.3722	2.9308	3.3722	3.3029	11.5893
	3	2.9308	3.3722	2.9308	3.3722	3.0673	14.2690
	4	2.9308	3.3722	2.9308	3.3722	3.4563	6.0352
	5	2.9308	3.3722	2.9308	3.3722	3.4563	6.0352
	10	3.2898	3.2985	2.9308	3.3722	10.4969	4.6692
8	2	1.9254	2.0128	2.2156	1.9253	10.7014	6.5236
	3	4.8482	1.8840	2.2156	1.9253	10.7014	6.5236
	4	4.8482	1.8840	2.2156	1.9253	10.7016	6.5538
	5	9.5056	1.7636	2.2156	1.9253	10.7016	6.5538
	10	9.5056	1.7636	2.2156	1.9253	10.7016	6.5538
10	2	2.3012	1.9860	2.3029	1.9869	3.1690	2.2786
	3	2.3012	1.9860	2.3029	1.9869	3.0971	2.6286
	4	2.3012	1.9860	2.3029	1.9869	12.7011	2.9521
	5	2.3012	1.9860	2.3029	1.9869	12.7011	2.9521
	10	2.3012	1.9860	2.3029	1.9869	22.5781	3.2461
15	2	2.1861	2.1979	2.1890	2.2005	2.2601	2.1436
	3	2.1861	2.1979	2.1890	2.2005	2.2601	2.1436
	4	2.1861	2.1979	2.1890	2.2005	2.6628	3.0787
	5	2.1861	2.1979	2.1890	2.2005	2.6628	3.0787
	10	2.1861	2.1979	2.1890	2.2005	2.8441	3.3304

While increasing TP generally reduced learned length-scales (implying increased feature relevance), restart count modulated this trend, especially for LML and LOO. At TP=4, CLML yielded $(31.05, 11.44)$, LML $(22.34, 21.37)$, and LOO $(24\text{–}33, \sim 11.8)$, reflecting differing relevance conclusions. CLML's robustness highlights its potential for being more reliable.

ARD feature importance with 4D inputs

Bar plot 4.9 visualizes ARD for four input features (LS1-LS4) using LML, CLML, and LOO methods as the number of training points (n) increases. Initially, with few training points (n=2, n=5), all methods show high variability in learned length scales, with some features (e.g., LS4, LS3 with LML at n=5) assigned very large length scales, indicating perceived low relevance. As the number of training points grows to n=20, the length scales generally decrease and stabilize, better reflecting feature relevance. For example, LS4 consistently has a larger length scale, suggesting it is the least relevant feature. CLML and LOO appear to offer more consistent convergence patterns across the increasing number of training points compared to LML, particularly in sparse data scenarios (low n). The analysis suggests that sufficient data is crucial for reliable ARD, and method choice impacts stability.

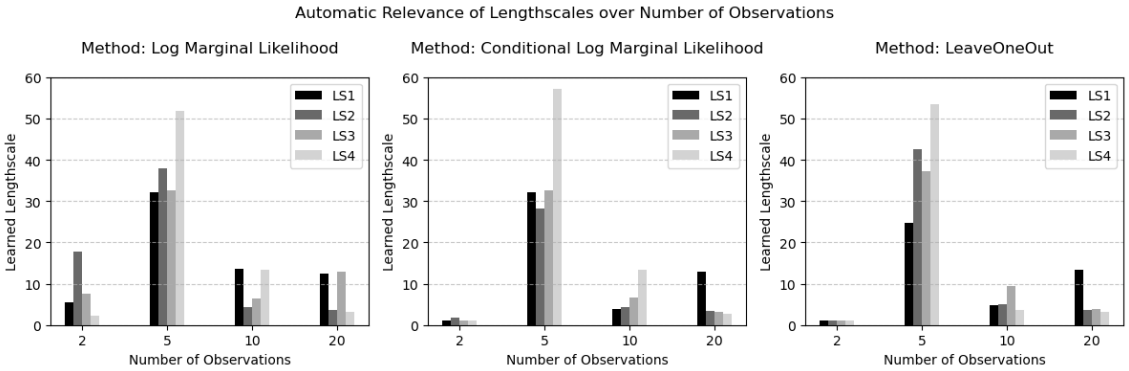

Figure 4.9: Impact of number of training points and inference method on learned lengthscales. Bar plots showing the learned lengthscales for four input features (LS1-LS4) with increasing numbers of training points, evaluated using Log Marginal Likelihood, Conditional Log Marginal Likelihood, and LeaveOneOut.

4.3 Conclusion and discussions

The thesis findings provide a detailed evaluation of GPR with ARD for hyperparameter tweaking and feature importance assessment. The main findings and conclusions are the following:

Performance of three optimization methods

- **LML** is stable but overestimates length scale, resulting in overly smooth function fits. It is less sensitive to dataset size and noise, but it has difficulty reliably recovering genuine feature length scales, especially when dealing with noisy or sparse data.
- **CLML** has substantial variability with small datasets but converges closer to genuine length scales with more data. It is less stable than LML, particularly when the signal variance is considerable, but it has the potential to be robust in certain scenarios.
- **LOO** is the most susceptible to tiny datasets and noise, but it produces more conservative and often more accurate length scale estimations when there is enough data and restarts. It outperforms LML at recovering true length scales in noisy environments.

Effects of varying size of training points

Increasing the number of training points (TP) enhances the consistency and accuracy of length scale estimates in all techniques. Small datasets (TP < 10) exhibit considerable variability and overestimation, but bigger datasets (TP ≥ 20) improve convergence to real values. This emphasizes the need for having enough data for accurate GPR modeling and ARD-based feature relevance estimation.

Effects of variances and restarts

Increased noise levels (σ_n^2) can lead to smoother perceived functions, thereby masking feature importance. Leave-One-Out (LOO) is more resistant to noise, especially on moderate to large datasets. Higher signal variance (σ_f^2) increases the learned length-scale (ℓ), with the LML offering the most stable estimates and the CLML demonstrating instability.

Additional restarts have little effect on LML and CLML, but they improve LOO's stability, particularly with smaller datasets. Beyond 5-10 restarts, the gains are minor, indicating that computational efficiency may be maintained with low restart counts.

Impact of ARD on multidimensional data

ARD accurately identifies feature importance in multidimensional contexts, with longer length scales suggesting lesser relevance. However, its performance is heavily reliant on dataset size and noise level. With sparse data, ARD estimates are unstable, but they stabilize with additional training points, especially for CLML and LOO.

In conclusion, the thesis shows that, by eliminating the need for manual trial-and-error, ARD and hyperparameter tuning minimize training effort. ARD streamlines the model and expedites training by automatically detecting and downweighting unnecessary input features. The model can reuse helpful hyperparameters, such as length scales and noise levels, when tuning is based on similar datasets, which speeds up convergence. By beginning with known initial values rather than arbitrary guesses, this method not only reduces computation costs and time but also increases model reliability. Still, model performance significantly relies on the optimization approach, dataset size, noise, and signal variance. LOO and CLML outperform LML in recovering real hyperparameters, especially in noisy or data-scarce circumstances, although ARD improves feature relevance estimates given sufficient data. These findings help to construct more robust and interpretable surrogate models for use in scientific computing and engineering.

CHAPTER 5

Future Works

While the current work shows the potential of GPR on synthetic data utilizing the RBF kernel, there are various avenues for future research and improvement.

Data dimension:
This thesis work used only up to two dimensions or features to evaluate the three different hyperparameter tuning methods. A broader or more dimensions could be evaluated to understand the behavior of those models. Tables and figures may get complicated with increased dimensions or features, which will induce more challenges for further extension. Evaluating models' performance on high-dimensional, noisy, and sparse datasets is crucial for determining their scalability and practical applicability.

Objective functions:
Three methods (LML, CLML, and LOO) were evaluated during this study. However, there could be other robust methods available or studied through other studies, which could be used to compare with the available methods.

Different kernels:
Only the RBF kernel was used in this study, as synthetic data was used and the pattern was known; however, other kernels were also explored during the literature review. Other kernels like Periodic, Rational Quadratic, etc, could be used concerning the real-world data. The pattern lies in the real-world data that may decide the kernels to be evaluated in future work.

Sparse approximation:
GPR's inherent computational difficulty presents issues when applied to large-scale datasets due to its cubic temporal complexity with the number of training points. Future research could look into sparse approximations of GPR, such as induced point approaches, variational inference, or stochastic variational Gaussian processes. These approaches strive to maintain GPR's interpretability and uncertainty quantification while considerably de-

creasing computational overhead.

Extended multi outputs:
Furthermore, extending the GPR framework to enable multi-output (more than two) or structured prediction tasks remains an open topic of discussion. Multi-task GPR models may capture interdependencies between several target variables, making them particularly useful in complex system modeling where outputs are naturally interrelated.

Finally, future work on Gaussian Process Regression will benefit greatly from broader kernel analysis, improved optimization tactics, real-world dataset applications, scalability solutions, structured output modeling, and integration with deep learning paradigms. These developments would make GPR more applicable and versatile in both theoretical and applied machine learning situations.

Appendices

CHAPTER A

Environments

A.1 Environments and implementation details

This section outlines the software tools, optimization techniques, and computational concerns involved in the real-world use of Gaussian Process Regression (GPR) models. The experimental setup's clarity and reproducibility are ensured by these specifics.

Libraries and Programming Environment

The following libraries were used in all implementations, which were completed in `Python 3.12.4`:

- **NumPy** 1.2.6 and **SciPy** 1.13.1 for numerical operations, random sampling, and optimization.
- For a conventional GPR implementation, **scikit-learn** 1.6 is used with built-in optimizers, kernel functions, and `GaussianProcessRegressor`.
- **Matplotlib** for data visualization.

All the experiments and codes were conducted in both online and offline environments. Python environments like Google Colaboratory, Jupyter Notebook, etc., are used for running experiments and IPython notebooks along with personal computers.

Numerical Stability and Precision

To avoid numerical instability in matrix inversion and Cholesky decomposition, a small jitter term (e.g., $\sigma_n^2 I = 10^{-10}$) was added to the diagonal of the covariance matrix. This practice ensures stable computation during training and prediction, especially for near-singular kernels.

Reproducibility Measures

Due to the positions of the random samples over the distribution (i.e., the first and last random sample being on the true function's mean line), the experimental outcomes varied a lot compared to those samples that are wide from the mean line of the true function. Apart from that, a fixed random seed does not represent all the scenarios that are required to consider.

To overcome this issue, ten random seeds (i.e., 24, 85, 39, 22, 93, 59, 98, 84, 11, 48) are generated once. Using this fixed set of random seeds, each experiment was conducted ten times to guarantee the robustness and dependability of the findings. These seeds are used to independently create training and testing datasets for every iteration, capturing a wide variety of sampling scenarios. Any anomalies or outliers that might arise from the randomness of individual datasets can be mitigated by averaging the results over these ten runs. This method provides a more balanced and statistically representative estimate of model performance, reducing the risk that conclusions are based on one favorable or unfavorable split. Moreover, this multi-run strategy reflects a common best practice in machine learning experiments, where averaging over multiple seeds helps account for uncertainty introduced by data sampling and initialization.

CHAPTER B

Foundations

B.1 Probability

Probability is a term that explains the chance of an outcome occurring among others. One common example of this is the coin toss. Usually, a coin has two sides, mostly called heads and tails (based on which country's coin it is). If throwing a coin in the air is an event (considering that the coin tossing or the coin is a fair one), it should end up with either heads or tails. Probability is a calculation between 0 and 1 to represent how likely an outcome may occur per event. Thus, the probability of the outcome (being heads or tails) for one event is $\frac{1}{2}$ or 0.5 (H or T). However, if the coin is flipped 50 or 100 times (events), the result may vary based on how many times the heads or tails appear.

$$\text{probability} = \frac{\text{Number of favorable outcome occurred}}{\text{Number of possible outcomes}}$$

An example given in Table B.1, a coin was flipped 10 times, where it fell on the heads (H) side 6 times and tails (T) 4 times. The probability of the outcome of being heads (H) over tails (T) was calculated at 60% as it occurred 6 times out of 10 times, similarly for tails it was 40%. Also, according to the theory, the probability calculations of all events must sum up to 1. For the coin toss case given in the table, it was 0.6 (H) and 0.4 (T), which adds up to 1.

Table B.1: Probability of Flipping/Tossing a Coin

Coin tossing events	Outcome	Outcome Frequency	Probability
10	H (Heads)	6	$P(\text{H}) = \frac{6}{10} = 60\%$
	T (Tails)	4	$P(\text{T}) = \frac{4}{10} = 40\%$

Prior Probability

Prior probability refers to the gained knowledge, assumption, or belief about a certain probability to start with. For example, for any previous few events that have already

occurred, there should be some calculated probability. However, when that gained probability or reinforced new information is used for the next few events, it may generate different probabilities with the help of some new pieces of evidence.

In early 2000, approximately 15 out of 10,000 people were affected by Tuberculosis (TB) disease. However, in 2023, that rate went down to 14 out of 10,000 people. Thus, the prior rate of 0.0015 (P(TB) = 0.0015) has been updated with the new information and evidence.

Posterior Probability

Posterior Probability is just the result of the process mentioned in the prior probability. The updated probability is the posterior for that event. When the prior belief is taken into consideration with the observed evidence's likelihood for an event, the updated estimation is called a posterior probability.

B.2 Bayes' Theorem

Back in the 18th century, an English mathematician and Presbyterian minister named Thomas Bayes first brought this theorem to light. This became a crucial base for the concept of probability and its statistics. This theory explains the calculation of getting the posterior or the updated estimation of the probability from the prior belief with the new evidence.

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

where

- $P(A \mid B)$: Posterior probability (the probability of hypothesis A after considering the evidence B)
- $P(B \mid A)$: Likelihood (If A has occurred or is true, then the probability of B happening)
- $P(A)$: Prior probability (The belief or the assumption probability for A).
- $P(B)$: Evidence (the cumulative or marginal probability of B).

Also, it can be written as below,

$$\text{Posterior Probability} = \frac{\text{Likelihood Function} \times \text{Prior Probability}}{\text{The Evidence}}$$

B.3 Expectations

Expectation and Average are quite similar yet different. The average or arithmetic mean measures the central line of tendency. It represents the sum of a given set of numbers divided by the set length. While Expectation is a different kind of concept in statistics, it is related to probability theories. This is a broader average value of the possible outcome weighted by the probability of the probable outcome of a random variable over many events. It is the average of a series of events and the possible outcomes from these events, considering the probabilities of these outcomes.

While average applies to a set of fixed values, expectations related to a random variable and their probability of different possible outcomes. The mathematical terms of Expectation are as follows,

For **discrete random variables** -

$$\mathbb{E}[f] = \mathbb{E}_{x\sim p(X)}[f(x)] = \sum_X P(x) \cdot f(x)$$

(In the case of Conditional Probability)

$$\mathbb{E}[f] = \mathbb{E}_{x\sim p(X|Y=y)}[f(x)] = \sum_X P(X|Y=y) \cdot f(x)$$

For **continuous random variables** -

$$\mathbb{E}[f] = \mathbb{E}_{x\sim p(X)}[f(x)] = \int_{\infty}^{-\infty} p(x)f(x)dx$$

(In the case of Conditional Probability)

$$\mathbb{E}[f] = \mathbb{E}_{x\sim p(X|Y=y)}[f(x)] = \int_{\infty}^{-\infty} P(X|Y=y)f(x)dx$$

where

- $\mathbb{E}[f]$: Expectation or expected value
- $P(x)$: Probability of each possible outcome

- $f(x)$: is a function of possible outcome from the given X

If there are some observations in a set of $\{x_1, x_2, \ldots, x_n\}$, then expectation of the function approximately is,

$$\mathbb{E}[f] \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n)$$

B.4 Variance

Variance refers to the expected quadratic distance between f and its mean $\mathbb{E}[f]$. It measures the scatteredness or spread of the data around its mean. The farther a data point is from its overall mean, the larger the variance it has compared to all other data points.

Suppose there are 5 people in a room, and their weights (x) are 70 kg, 65 kg, 100 kg, 55 kg, and 110 kg. Thus, the calculation of variance would be like below in Table B.2,

Average or arithmetic mean (μ), $\frac{\text{Sum of weights}}{\text{Number of people}} = \frac{70+65+100+55+110}{5}$ = 80 kg.

Variance Equation:

$$\begin{aligned} var[f] &= \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \\ &= \mathbb{E}[f(x)^2 - 2f(x)\mathbb{E}[f(x)] + \mathbb{E}[f(x)^2]] \\ &= \mathbb{E}[(f(x)^2] - \mathbb{E}[(f(x)]^2 \end{aligned} \tag{B.1}$$

Table B.2: Calculation of Variance and Standard Deviation of Weights

Person	Weight (kg)	Deviation from Mean $(x-\mu)$	Squared Deviation $((x-\mu)^2)$
1	70	$70-80=-10$	$(-10)^2=100$
2	65	$65-80=-15$	$(-15)^2=225$
3	100	$100-80=20$	$(20)^2=400$
4	55	$55-80=-25$	$(-25)^2=625$
5	110	$110-80=30$	$(30)^2=900$
		Total	**2250**

$$\text{Variance}(\sigma^2) = \frac{\text{Total Squared Deviation}}{\text{Number of People}} = \frac{2250}{5} = 450$$

$$\text{Standard Deviation}(\sigma) = \sqrt{\text{Variance}} = \sqrt{450} \approx 21.21$$

B.5 Co-Variance

Variance calculates the changes over different values of a random variable, such as weights, in the previous example in Table B.2. However, covariance refers to the amount of change two variables make together. It can be a positive or negative relation of change, meaning it has a direction in the relationship of the change. For example, if both variables have similar directions of change or increase/decrease together, then it has a positive relation. On the other hand, if one is increasing while the other one decreases, then the relationship is negative. It is also possible to have no relationship at all, it is called zero covariance.

B.6 Gaussian Distribution

In statistics, the Gaussian distribution is an essential probability distribution. Another name for the Gaussian distribution is the normal distribution. Gaussian distribution is a symmetric bell shape in its distribution, meaning most of the data cluster around its average or mean value. The average value or mean is the peak location; it is the center of the distribution. This is one of the most widely used topics in statistics.

The probability density function of a normal distribution has two parameters, mean (μ) or the mean where the center tendency lies, and standard deviation (σ) or the spread of the data around the mean.

The probability density function of a Gaussian Distribution:

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where
μ = Mean
σ^2 = Variance
$\frac{1}{\sqrt{2\pi\sigma^2}}$ = Correction Factor, scaled by σ^2
$(x-\mu)^2$ = Squared Distance

In a bell curve shape distribution or normal distribution, changing the value for the mean shifts the center of the bell curve to the left or right. Also, changing the value of variance stretches or shrinks the bell shape.

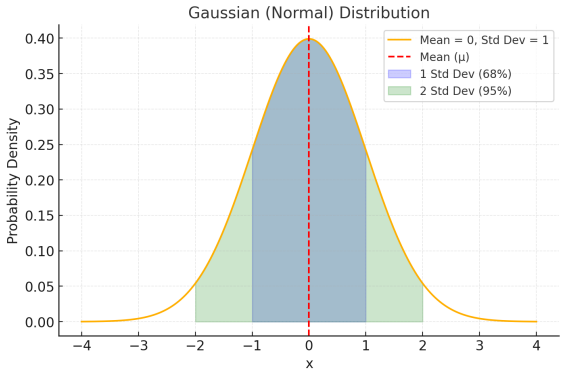

Figure B.1: Normal or Gaussian Distribution

B.7 Multivariate Gaussian Distribution

Similar to the Gaussian Distribution but with another parameter dimensionality D, instead of a single random variable multivariate Gaussian distribution works with D-dimensional vectors. Also, instead of a mean and a standard deviation, it has two vector parameters called Mean Vector (μ) and Covariance Matrix (Σ).

If x is a D-dimensional vector, whose values are $x = (x_1, x_2, \ldots, x_D)^T$, then the formula of Multivariate Gaussian Distribution is as follows,

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}$$

where
$\Sigma = Cov[x,x]$
and $\mathbb{E}[x] = \mu$

B.8 Maximum Likelihood

When working with data and machine learning, the main goal is to construct a model that can explain most data or observations. These models may have no or multiple parameters. Different values of these parameters may result in varied outputs for the model. Maximum likelihood is related to finding the parameter value that maximizes the likelihood of explaining the observations.

Likelihood is the probability of observing the given data using the parameters of a model. Likelihood distribution with parameters, $P(D|w)$, where D = $(x_1, x_2, \ldots, x_n)$. Maximum likelihood can be found by drawing this likelihood over each observation and getting the maximum point among all measurements.

Maximum Likelihood is as follows,

$$W_ML = \arg\max_w P(D|w)$$

thus,

$$\begin{aligned} P(D|w) &= P(x_1, x_2, \ldots, x_{N|w}) \\ &= \prod_{i=1}^{n} P(X_i|w) \end{aligned} \tag{B.2}$$

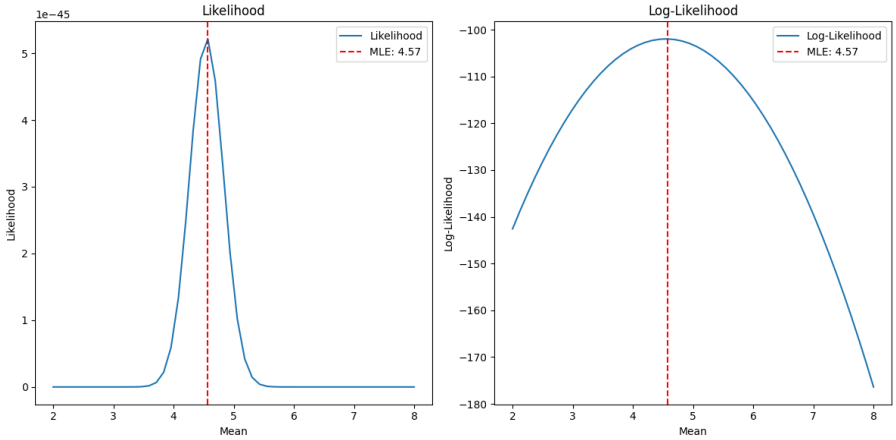

Figure B.2: Likelihood, Maximum and Log Likelihood Estimation

B.9 Conditional Marginal Likelihood

Conditional Marginal Likelihood (CML) is a concept used in probability and statistics, particularly in Bayesian inference and machine learning. CML is useful in Bayesian models to evaluate how well a model explains observed data while conditioning certain variables. It helps in parameter estimation, model selection, and improving predictions by filtering out irrelevant factors.

$$p(Y \mid X) = \int p(Y \mid X, \theta) p(\theta \mid X) d\theta$$

where

- Y = observed data (what will be predicted)
- X = given conditions (extra information)
- θ = unknown parameters (hidden factors)
- p(Y|X,θ) = likelihood of data given parameters
- p(θ|X) = prior probability of parameters given conditions

There are various issues with using marginal likelihood rather than conditional marginal likelihood. The propensity to either overfit or underfit is a significant problem since marginal probability may favor models that are too complicated or too simple, which results in poor generalization.
Furthermore, a high marginal likelihood is useless for evaluating generalization because it does not always imply improved predictive performance on unknown data. Additionally, it has trouble with hyperparameter adjustment in intricate models like deep learning,

where performance is greatly impacted by the selection of parameters and priors.
Calculating marginal likelihood is computationally costly, particularly for large-scale models, because it frequently requires high-dimensional integrations. By conditioning on pertinent subsets of data, conditional marginal likelihood improves model evaluation and overcomes these issues.

CHAPTER C

Additional figures and results

C.1 Experiment with fewer training points and short feature length scale

Figure C.1 suggests that with a very short feature length scale ($\ell_{feature}$=0.001), and a few training points (4), the model is fitting the training data very closely at the observation points, as shown by the prominent spikes there. The true function exhibits high-frequency oscillations, posing a challenge for GPR with coarse length-scales. The kernel length-scale ($\ell_{init} = 0.1$) is larger than the feature length-scale ($\ell_{feature} = 0.001$), causing oversmoothing.

As a result, the model captures the broad structure but fails to reconstruct sharp local variations. Confidence intervals widen in sparse regions, indicating increased predictive uncertainty.

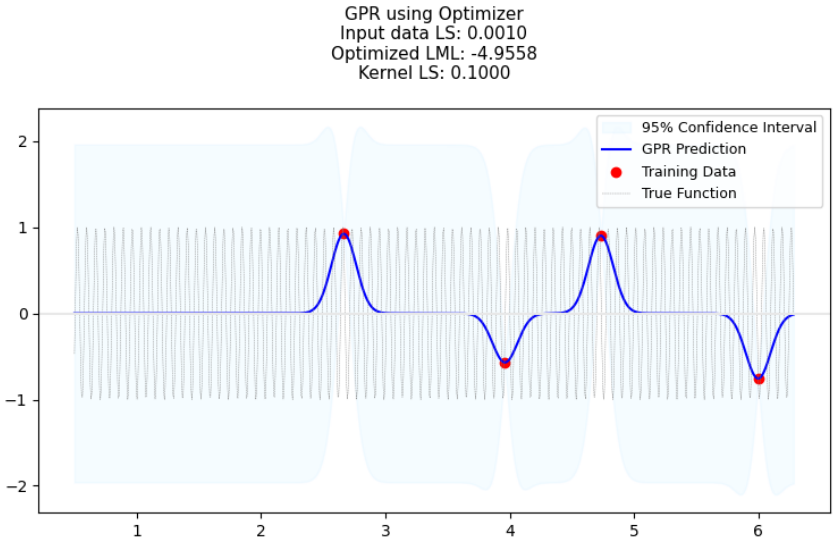

Figure C.1: Gaussian Process Regression (GPR) fit using optimizer-based hyperparameter tuning. The plot displays the GPR prediction (blue line), true function (grey line), training data (red dots), and 95% confidence interval (shaded area). The optimized Log Marginal Likelihood (LML) is -4.9558 with an initial kernel lengthscale (ℓ_{init}) of 0.1.

C.2 Hyperparameter tuning using log marginal likelihood

Table C.1 presents the outcomes of tuning the hyperparameters (ℓ and σ_f^2) of a GPR model employing a RBF kernel. Two distinct scenarios were investigated: one without added noise in the training data and another with a noise level of 0.1. A fixed signal variance means a signal variance of 1.0 added with bounds (1.0, 1.0), which does not affect the kernel.

In the noise-free case, optimal learned lengthscales and signal variances are sensitive to the initial values ($\ell_{feature}$, σ^2), especially when $\ell_{feature}$ is 3.0, where high log marginal likelihood (LML) values and stable predictions are achieved. For low or high $\ell_{feature}$ (0.5 or 5.0), optimization often results in parameter estimations that are linked to lower LML values and less accurate fits to the actual underlying function, which suggests locally optimum or inferior solutions. The optimization becomes more resilient when a noise (σ_n^2=0.1) is included. Regardless of initialization, it consistently converges to similar learnt parameters. Nevertheless, this resilience results in lower LML values, which indicate a worsened model fit as a result of noise. Therefore, while modest noise can lessen sensitivity to early settings, thorough initialization is crucial in noise-free scenarios.

Table C.1: Hyperparameter optimization for GPR with RBF kernel under varying noise levels.

Hyperparameter Tuning (Training points: 10, Kernel: RBF, $\ell_{feature}$: 1.0)					
Hyperparameters		**Learned Hyperparameters**		**CI**	**LML**
ℓ_{init}	σ_f^2	$\ell_{learned}$	σ_f^2	$\mu_{mean}(\bar{y}) \pm 2\sigma_{mean}$	LML Value
Case: No noise in the input data					
0.5	1.0	0.5	1.0	-0.3597 – 0.4354	-1.7935
	2.0	0.5	2.0	-0.5243 – 0.6001	-4.5881
	5.0	0.5	5.0000	-0.8511 – 0.9268	-8.7668
1.0	1.0	1.0	1.0	0.0204 – 0.0588	9.6592
	2.0	1.0	2.0	0.0124 – 0.0668	6.8216
	5.0	1.0	5.0000	-0.0033 – 0.0826	2.6170
3.0	1.0	2.6458	4.6757	0.0384 – 0.0385	28.9233
	2.0	2.6459	4.6769	0.0384 – 0.0385	28.9233
	5.0	2.6459	4.6767	0.0384 – 0.0385	28.9233
5.0	1.0	0.0000	0.6019	-1.3278 – 1.3278	-11.5189
	2.0	0.0000	0.6019	-1.3278 – 1.3278	-11.5189
	5.0	0.0000	0.6019	-1.3278 – 1.3278	-11.5189
Case: Noise (Level = 0.1) added to the input data					
0.5	1.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	2.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	5.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
1.0	1.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	2.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	5.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
3.0	1.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	2.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	5.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
5.0	1.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	2.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228
	5.0	1.4052	0.4887	-0.4405 – 0.4776	-4.5228

References

[1] A. Forrester, A. Sobester, and A. Keane, *Engineering Design via Surrogate Modelling*. Chichester, UK: Wiley, 2008, Used for fundamental concepts in surrogate modeling.

[2] B. Bischl *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 2, e1484, 2023.

[3] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006, ISBN: 026218253X. [Online]. Available: `www.GaussianProcess.org/gpml`.

[4] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker, "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences*, vol. 41, no. 1, pp. 1–28, Jan. 2005, Available online 8 April 2005, ISSN: 0376-0421. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0376042105000102`.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[6] S. Lotfi, P. Izmailov, G. Benton, M. Goldblum, and A. G. Wilson, "Bayesian model selection, the marginal likelihood, and generalization," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 14 345–14 368. [Online]. Available: `https://proceedings.mlr.press/v162/lotfi22a.html`.

[7] D. J. C. MacKay, "Bayesian methods for adaptive models," PhD thesis, California Institute of Technology, Pasadena, CA, USA, 1994.

[8] K. Liu, Y. Li, X. Hu, M. Lucu, and W. D. Widanage, "Gaussian process regression with automatic relevance determination kernel for calendar aging prediction of lithium-ion batteries," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 3767–3777, 2020.

[9] D. J. C. MacKay, "Introduction to gaussian processes," in *Neural Networks and Machine Learning*, ser. NATO ASI Series F: Computer and Systems Sciences, C. M. Bishop, Ed., vol. 168, Springer, 1998, pp. 133–166.

[10] D. Duvenaud, *The kernel cookbook: Advice on covariance functions*, Accessed: 10 March 2025, 2014. [Online]. Available: https://www.cs.toronto.edu/~duvenaud/cookbook/.

[11] D. K. Duvenaud, "Automatic model construction with gaussian processes," PhD thesis, University of Cambridge, Pembroke College, 2014.

[12] C. Broyden, "Quasi-newton methods and their application to function minimization," *Mathematics of Computation*, vol. 21, no. 99, pp. 368–381, 1967.

[13] R. M. Neal, *Bayesian Learning for Neural Networks* (Lecture Notes in Statistics). New York: Springer, 1996, vol. 118.

[14] J. Jordon *et al.*, "Synthetic data – what, why and how?" *arXiv preprint arXiv:2205.03257*, 2022. [Online]. Available: https://arxiv.org/abs/2205.03257.

[15] R. L. Harrison, "Introduction to monte carlo simulation," *AIP Conference Proceedings*, vol. 1204, pp. 17–21, 2010. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC2924739/pdf/nihms219206.pdf.

[16] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed. Springer, 2004.

[17] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, 5th. Academic Press, 2009, ch. 5, pp. 160–168.

[18] P. Adamson and J. Nicholas, *Introduction to Trigonometric Functions*. Sydney, Australia: Mathematics Learning Centre, University of Sydney, 1998, Accessed: 2025-03-20. [Online]. Available: https://pdfdirectory.com/pdf/0688-introduction-to-trigonometric-functions.pdf.

[19] E. Kreyszig, *Introductory Functional Analysis with Applications*. Wiley, 1978.

[20] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[21] B. Wang and T. Chen, "Gaussian process regression with multiple response variables," *Chemometrics and Intelligent Laboratory Systems*, vol. 142, pp. 159–165, 2015.

[22] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018. [Online]. Available: https://doi.org/10.1016/j.jmp.2018.03.001.

[23] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008. [Online]. Available: https://projecteuclid.org/euclid.aos/1211819561.

[24] D. Azzimonti, *Gaussian processes and sequential design of experiments*, https://ai-dd.eu/sites/default/files/school-2/Azzimonti_noAppendix.pdf, Accessed: 2024–11-05, 2022.

[25] S. learn Developers, *Gaussian process regression (gpr)*, https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy_targets.html, Accessed: 2025-03-06, 2021.

[26] G. Gundersen, *Practical gaussian process regression*, Accessed: 2025-03-10, 2019. [Online]. Available: https://gregorygundersen.com/blog/2019/09/12/practical-gp-regression/.

[27] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," in *Journal of Machine Learning Research*, vol. 13, 2012, pp. 281–305.

[28] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.