UCLouvain

Institut de Statistique, Biostatistique et Actuariat

An Overview of Gaussian Process Regression In The Context of Multi Task/Multi Output Regression

Membres du jury:

Prof. Robin Van Oirbeek *Promoteur*
Prof. XXXXXXX

Master thesis prepared for
the fulfillment of the master in
Data Science
(Statistics Focus)
by:

Agenor Uel Dje Palmer Kouame

Louvain-La-Neuve
Juin 2025

Preface.

This dissertation has been prepared in partial fulfillment of the requirements for the master degree in actuarial sciences delivered by the Université Catholique de Louvain.

Acknowledgment.

Many people have contributed to the completion of this dissertation.

Contents

Chapter 1

Introduction

Background and Context: The energy industry is an highly lucrative and essential market, in the context of the climate crisis the energy transition market is expected to reach 5.3 trillion US dollars in 2031. Beyond its financial impact, it plays a critical role in improving human lives as every aspect of modern human activity has a crucial need for reliable and abundant energy. The integration of technology in the production of energy has been transformative, particularly in reducing the time required for designing, manufacturing and testing the equipment but also enhancing system efficiency through predictive maintenance. Advances in sensor technology, data storage, and computational power have created a fertile ground for the application of Machine Learning (ML) and Artificial Intelligence (AI) in energy sector for energy generation, but also storage technology via batteries.

The increasing availability of data from diverse energy equipment sources has spurred interest in using shared knowledge to develop more accurate models. Traditional single-task regression models handle outputs separately, often missing the potential advantages of multi-task regression. By modeling multiple tasks simultaneously, multi-task regression methods can leverage shared information across tasks, leading to more accurate predictions. For example, when constructing regression models for multiple batteries exhibiting different behaviors, instead of building individual models for each battery, one can develop a global model that simultaneously encompasses all batteries by leveraging the correlations among them. This is especially useful in predictive maintenance, where data from a newly acquired machine may not be readily available, such model can help avoiding downtime by using information from other machines to train a model for the new machine. Gaussian Process Models (GPMs), a non-parametric and probabilistic framework, are particularly suited to these tasks as they can flexibly model nonlinear relationships and provide uncertainty estimates. Given the importance of energy generation and distribution for the economy, the ability to quantify uncertainty is of paramount importance. By extending GPMs to handle multiple tasks, one can create a more comprehensive model capable of improving predictive performance and supporting critical decision-making.

Research Problem:

In the energy sector, the need to accurately predict outcomes across multiple, interrelated variables is crucial for informed decision-making. Traditional single-tasks regression models, however, fail to fully exploit the shared information between tasks, leading to less optimal performance. Multi-tasks Gaussian Process Regression (GPR) models address this problem by incorporating correlations between tasks, thereby enhancing predictive accuracy. However,

there remain challenges in optimizing these models for high-dimensional, complex datasets, particularly when applying them to real-world contexts such as federated learning. This study aims to address these challenges by developing more effective methods for multi-task regression for the energy sector application.

Significance of the Study: This study holds significance because of its potential to improve energy storage capability by developing more accurate predictive models for batteries but also energy generation by predicting the active power from wind turbines. By focusing on Multi-Task Gaussian Process Regression (MTGPR), it seeks to advance predictive maintenance by better accounting for machine-specific conditions. Additionally, with the growing prominence of the Internet of Things (IoT) and federated learning, extending these models to operate in a federated framework aligns with the future of energy technology. Processing large-scale, decentralized data efficiently while maintaining prediction accuracy and providing uncertainty estimates is vital to advancing AI's role in the energy sector.

Research Questions and Objectives:

This study is guided by several key questions. First, it seeks to determine how the correlation between tasks can be optimally harnessed in Multi-Task Gaussian Process Regression. It also aims to evaluate the theoretical and practical benefits of different methods for quantifying task correlations. Finally, the study investigates how MTGPR models can be extended to the federated learning framework without compromising performance or interpretability. The study's objectives are to compare methods for integrating task correlations in MTGPR, to assess the theoretical and empirical performance of these methods in terms of complexity, convergence, bias, and variance, and to extend MTGPR to a federated learning environment while evaluating its application to real energy data.

Thesis Statement:

This thesis investigates methods for Multi-Task Gaussian Process Regression in the energy sector, focusing on task correlation to improve predictive performance. It explores various theoretical approaches and their practical applications, culminating in a comprehensive multi-task regression framework that extends to the federated learning context.

Thesis Structure:

The thesis begins with a literature review that explores the current understanding of Gaussian Process Regression, its relationship to Bayesian statistics, and the state of the art in Multi-Task Gaussian Process Regression. The review highlights gaps in the literature and the challenges associated with modeling task correlations. The second section presents the theoretical framework, comparing different approaches to quantifying task correlations in MTGPR. It focuses on theoretical advantages such as convergence properties, complexity, bias, and variance, and how these factors influence the choice of method. The third section conducts a simulation study where known outcomes are used to verify theoretical predictions. This study will evaluate the method's convergence, stability, and consistency with theoretical expectations. The fourth section applies these methods to real-world energy data to assess their practical utility. This will determine whether the results from simulated data align with those obtained from real datasets. The fifth section explores the extension of MTGPR to a federated learning framework, providing a theoretical definition of federated learning before diving into federated Gaussian Process Regression. This section will also feature simulations and real-data analyses to assess the extension's feasibility. The thesis concludes with a discussion of the findings from both multi-task GPR and its extension into federated learning, summarizing the key points and providing directions for future research.

Chapter 2

Literature

2.1 Gaussian Process Models

Gaussian Process Models (GPs) are key supervised learning models that can be applied to regression and classification tasks. Even though it is widely used, understanding the underlying principles that make Gaussian Process Models work can be quite challenging. It is therefore important to explain the mathematical concepts behind the model including multivariate Gaussian distribution, kernels, non parametric models, joint probabalities, conditional probabilities and distribution over functions.

2.1.1 Gaussian Distribution (Univariate and Multivariate)

We start by introducing the concept of Gaussian distribution see for example HÃ€rdle and Simar [8]. We define a random variable X that follows a Gaussian distribution with mean μ and variance σ^2. The probability density function (PDF) of this Gaussian random variable is then given by:

$$P_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.1.1}$$

If we sample n points from the Gaussian density function such that $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^\top$. The generated points can then be connected as shown in the Figure 2.1.1

Sampling points in that fashion is equivalent from sampling a vector of random variables from a multivariate Gaussian distribution such that:

$$\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$$

because the points sampled are independent from each other:

$$\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$$

We can see from the Figure 2.1.1 that when connecting the points sampled from a univariate it is possible to see a function however the function lack smoothness for regression tasks, one can correlate the points forming a multivariate Gaussian distribution. The multivariate Gaussian distribution is the extension of the univariate Gaussian distribution to allow correlation (linear dependence) between the variables being modeled. The probability density function of the multivariate Gaussian distribution is given as follows:

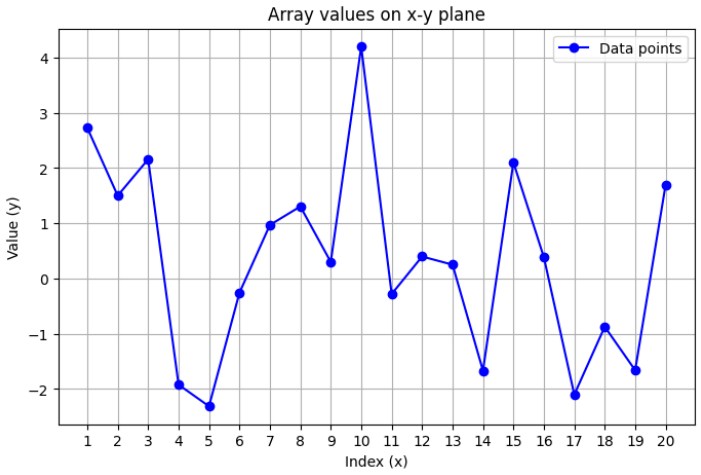

Figure 2.1.1: Points sampled from a univariate Gaussian density function

$$P_X(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.1.2}$$

where:

- $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.
- $\boldsymbol{x}$ is an D-dimensional vector such that $\boldsymbol{x} = [x^1, x^2, \ldots, x^D]^\top$.
- $\boldsymbol{\mu}$ is the mean vector and also D-dimensional such that $\boldsymbol{\mu} = [\mu^1, \mu^2, \ldots, \mu^D]^\top$.
- $\boldsymbol{\Sigma}$ is the covariance matrix which is positive semi-definite and symmetric with a dimension $d \times d$ where d is the number of dimensions.

The covariance matrix is crucial as it provides an understanding of the dependence structure between jointly modeled random variables as explained in [16]. A multivariate Gaussian distribution allows to have distribution over a random vector of real-valued variables such that:

$$[x^1, x^2, x^3 \ldots, x^D] \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Here, the D-dimensional vector is drawn from the distribution that is also D-dimensional as a random realization of this distribution. If we connect the random variables from each dimension of a correlated multivariate Gaussian density function the same way we did with multiple points sampled from a univariate Gaussian density function we obtain Figure 2.1.2.

In fact, as the dimensions are correlated if we connect the points sampled from the dimensions the graph tend to look like a smooth function even though not smooth enough in this case for regression tasks. A Gaussian process is an extension of the multivariate Gaussian density to an infinitely sized vector (as continuous functions have infinite number of points). This allows to see Gaussian processes not as distribution over a finite sized vectors but as distributions over functions which will be discussed in more details in the following section.

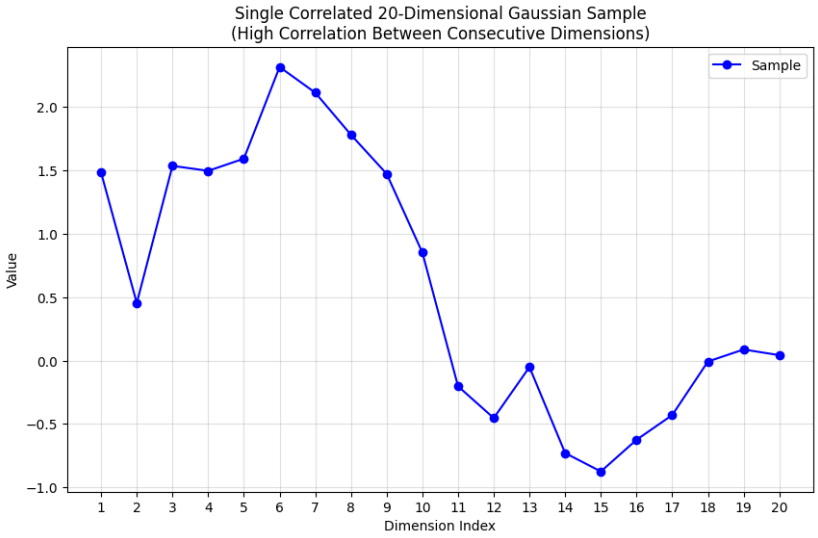

Figure 2.1.2: Points from a 20-variate multivariate Gaussian density function

2.1.2 Gaussian Processes

Now that the Gaussian distribution has been introduced, it is essential to extend this understanding to Gaussian Processes (GPs) to build towards Gaussian Process Models and, eventually, Gaussian Process Regression (GPR). In this section, we present background ideas on Gaussian Process and later in Section 2.2 mainly based on the previous work from Rasmussen and Williams [2] and Beckers [13]. A Gaussian process is a stochastic (random) process, defined as a collection of random variables over time or space. To formalize this, we define the probability space $(\Omega_{ss}, F_\sigma, P)$:

- Ω_{ss} is the sample space containing all possible outcomes or values that our collection of random variables can take.
- F_σ, or σ-algebra, is a collection of subsets of Ω_{ss} that we are interested in measuring probabilities for. It is crucial as it provides a framework to assign probabilities consistently. F_σ must satisfy three conditions:
  1. It must contain the sample space Ω_{ss} itself.
  2. If a set is in F_σ, then its complement must also be in F_σ.
  3. If a countable union of sets is in F_σ, then each of these sets must also belong to F_σ.
- P is the probability measure, which assigns probabilities to the events in F_σ, describing the likelihood of various events.

We also define $Z \subseteq \mathbb{R}^{n_z}$ as the index set, $n_z \in \mathbb{N}$ is the number of dimensions. Each element $z \in Z$ specifies a particular point or time at which we observe or evaluate the stochastic process. The function $f_{GP}(z, \omega_{ss})$ represents the stochastic process where z indicates the point of evaluation, and $\omega_{ss} \in \Omega_{ss}$ represents a realization of the function. Simplifying the notation

to $f_{GP}(z)$ allows us to focus on the random variable itself, without explicitly referring to the specific realization. This notation makes it easier to work with the process in formulas and discussions, as it now depends only on the index.

A Gaussian process is then defined by a mean function $m : Z \mapsto \mathbb{R}$ and a covariance function $k : Z \times Z \mapsto \mathbb{R}$. This leads to the following formalization:

$$f_{GP}(z) \sim \mathcal{GP}(m(z), k(z, z')) \tag{2.1.3}$$

where:

$$\begin{aligned} m(z) &= E[f_{GP}(z)] \\ k(z, z') &= E[(f_{GP}(z) - m(z))(f_{GP}(z') - m(z'))] \end{aligned}$$

Before discussing what the mean function and the covariance function represent in the context of Gaussian Processes it is important to understand the concept of distribution over functions. We start by defining a function $f_0 \in \mathcal{F}$ and $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ being a finite set of points. We consider $\mathcal{F}$ to be the set of all possible functions mapping $\mathcal{X}$ to $\mathbb{R}$. In our example with f_0 we then have $f_0(\cdot) \in \mathcal{F}$ that gives:

$$f_0(x_1) = 0, f_0(x_2) = 15, \ldots, f_0(x_{m-1}) = 12, f_0(x_m) = 3$$

The set of points introduced in the function result in real-valued variables that can be represented as a vector such that:

$$\boldsymbol{f_0} = [f_0(x_1), f_0(x_2) \ldots f_0(x_m)]^\top$$

The concept of distribution over functions rises from associating a probability measure to the function $f_0(\cdot)$ and ultimately to all functions in $\mathcal{F}$. Functions, however, are a collection of infinitely many points therefore we extend the concept explained above to an infinite number of points. In that case, $f_0(\cdot)$ and all other functions in $\mathcal{F}$ are vectors belonging to an extremely high dimensional multivariate Gaussian distribution making it a distribution over functions of infinite domain.

Next, let's have a closer look at the mean function, which is defined as:

$$m(z) = E[f_{GP}(z)]$$

The mean function is the expected value of the function $f_{GP}(\cdot)$ when evaluated at the points in the domain of interest. To understand the covariance function and how it relates to the principle of distribution over function it is important to go back to the concept of multivariate Gaussian distribution. Let's consider the function $f_0(\cdot)$ again as we evaluated the vector of points $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^\top$ the vector of realizations $\boldsymbol{f_0} = [f_0(x_1), f_0(x_2) \ldots f_0(x_m)]^\top$ is obtained. Because we are in the context of Gaussian Processes, we consider this vector to have a multivariate Gaussian distribution with no correlation between the dimensions and a mean as 0 vector for an easier comprehension such that

$$\boldsymbol{f_0} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$$

Because there is no correlation between the dimensions the covariance matrix is given by

$$\mathbf{\Sigma} = \sigma^2 \mathbf{I}$$

meaning that each realization of the function when a particular points x_i is evaluated in it is a dimension of the multivariate Gaussian distribution and the realization are independent from each other such that the probability over the function $f_0(\cdot)$ is given by

$$p(f_0) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(f_0(x_i) - \mu_i)^2}{2\sigma^2}\right)$$

If we want to plot a graph of points it is equivalent to draw m univariate Gaussian distribution and connecting the dots with a line it will give us a noisy line like the graph like 2.1.1 but with more points.

Functions, however, are smooth and similar inputs should produce similar outputs. This is where the concept of covariance function (kernel function) is important for understand distribution over functions. For the time being, only the squared exponential kernel also known as the Gaussian kernel, which is defined as:

$$k(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right)$$

will be considered. We also will not focus on its properties as they will be explained in more details as well as other common kernels in following sections. Here, σ^2 controls the variance (overall amplitude) and ℓ the length scale control for how the dependence decreases with distance between the points [19]. For inputs x_i and x_j that are close together the kernel yields a covariance close to 1 whereas for distant points it is close to 0. With the new covariance matrix, when we sample from the multivariate Gaussian distribution the resulting function values $f_0(x_i)$ are no longer independent. The induced correlations ensure that function values of inputs that are close to one another are similar, producing a smooth curve when plotted. If we extend this idea from a finite domain to an infinite domain we obtain a smooth function for $f_0(\cdot)$ which has a probability density function given by a multivariate Gaussian distribution and this will be the same for all functions $\{f_1(\cdot), \ldots, f_m(\cdot)\}$ in the set of possibles functions $\mathcal{F}$.

2.2 Gaussian Process Regression

The next concept in line is Gaussian Process Regression (GPR). Gaussian Process Regression is closely linked to Bayesian Linear Regression as both take advantages of observed data in order to update the model according. Introducing Bayesian Linear Regression is thus important in order to move forward to Gaussian Process Regression.

2.2.1 Bayesian Aspect

Bayesian Linear Regression is an alternative to the frequentist Ordinary Least Squares (OLS) estimate linear regression models. It takes advantage of the Bayesian methodology in an effort to update the model as new observations become available [12]. In this section, Bayesian Linear Regression will be introduced as a concept. It is a crucial step toward understanding Gaussian Process Regression.

2.2.1.1 Linear Regression

Linear regression is a statistical tool that allows to analyze the relationship between a set of explanatory variables $x \in \mathbb{R}^d$ and a dependent variable y. The work presented on Linear Regression here is built upon HÃ€rdle and Simar [8]. One can specify the model of linear regression model as:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2.2.1}$$

where:

- $\boldsymbol{y}$ is the vector of dependent variable of dimension $n \times 1$.
- $\boldsymbol{X}$ is the matrix of independent variables for each individual of dimension $n \times d$.
- $\boldsymbol{\beta}$ is the vector of coefficient for the independent variables of dimension $d \times 1$.
- $\boldsymbol{\epsilon}$ is the vector of error term representing sampling noise (or the effect of variables not included in the model) of dimension $n \times 1$. The error terms are assumed to follow Gaussian distribution with a zero mean and unknown variance.

The estimation of $\boldsymbol{\beta}$ via the method of Ordinary Least Squares (OLS) gives (check appendix for proof):

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{2.2.2}$$

The variance of the estimator is given by:

$$Var[\hat{\boldsymbol{\beta}}] = \sigma^2 (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \tag{2.2.3}$$

In the context of Bayesian Linear Regression, the estimation of the vector of coefficient as well as the variance is different. It follows Bayesian methodology.

2.2.1.2 Bayesian Linear Regression

We introduce the Bayesian paradigm of thinking by having a closer look at the Bayes Rule first which corresponds to:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)} \tag{2.2.4}$$

where:

- $P(A)$ is the probability of event A, also known as the prior probability.
- $P(A \mid B)$ is the conditional probability of event A knowing that event B is true, it is also known as the posterior probability.

- $P(B \mid A)$ is the conditional probability of event B knowing that event A is true, it is also known as the likelihood.

- $P(B)$ is the probability of event B.

Bayes' theorem can be used to calculate the posterior probability (the updated probability of after new evidence is observed) of an event [12]. The posterior probability is calculated by updating the prior probability (prior belief about the occurence of an event), which nicely summarizes what the Bayesian paradigm is all about: prior belief is updated using data, resulting in a posterior or updated understanding of the random process at hand.

After introducing Bayes' Theorem, it can now be used to explain Bayesian Linear Regression. We introduced earlier in the section the standard linear regression model with its equation and how the dependent and independent variables where related. Here, we introduce the Bayesian Linear Regression model in the context of a simple regression model. Let $\boldsymbol{y}$ be a vector of dependent variables such that $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\top$. We define a simple regression model such that:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where:

- y_i is the dependent variable associated with the i-th individual.

- x_i is the independent variable associated with the i-th individual.

- β_0 is the intercept parameter of the model.

- β_1 is the coefficient associated with the independent variable x_i.

- ϵ_i are the error terms which are independent and identically distributed as Gaussian random variables with mean zero and constant variance σ^2.

The goal of the model will be to update the distribution of the parameters β_0, β_1 and σ^2 the variance based on the observed data. Under our assumption for ϵ_i we have for the dependent variables y_i conditionally on x_i, β_0, β_1 and σ^2 the following distribution:

$$y_i | x_i, \beta_0, \beta_1, \sigma^2 \sim \mathcal{N}(\beta_0 + \beta_1 x_i, \sigma^2)$$

Under the assumption that the dependent variables are independent, the likehood for y_i is then given by:

$$p(y_i | x_i, \beta_0, \beta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right)$$

Since we assumed the independence between the dependent variables the likelihood for the whole dataset is then the product of the individual likelihood such that:

$$p(\boldsymbol{y} | x_i, \beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right)$$

The objective is to update the distribution of the unknown parameters β_0, β_1 and σ^2 the variance based on the observed data. The Bayesian paradigm is all about having prior knowledge or assumptions about data which are updated as new information becomes available. The update of the posterior distribution is done by multiplying the prior with the likelihood. The likelihood can be seen as the mechanism by which new observations are continuously incorporated into the model, refining the parameter estimates. For simplicity, let consider nothing is known about the model at hand meaning that non-informative priors are considered for the model parameters such that:

$$p(\beta_0, \beta_1|\sigma^2) = 1$$

and

$$p(\sigma^2) = \frac{1}{\sigma^2}$$

The prior for the parameters β_0 and β_1 which $p(\beta_0, \beta_1 \mid \sigma^2)$ is considered non-informative because when multiplied to the likelihood it does not change the likelihood adding no prior belief to the likelihood. The prior for the variance σ^2 is also considered non-informative. This is because when this prior is combined (multiplied) with the likelihood, it does not favor any particular scale of variance [5]. Specifically, the prior assigns equal weights to all scales on a log-scale of σ^2. the joint prior for the model parameters is then given as follows:

$$p(\beta_0, \beta_1|\sigma^2) \times p(\sigma^2) = p(\beta_0, \beta_1, \sigma^2) = \frac{1}{\sigma^2}$$

Applying the Bayes rule to obtain the joint posterior distribution for the parameter, we obtain:

$$\begin{aligned}
p(\beta_0, \beta_1, \sigma^2|\boldsymbol{y}) &\propto \left[\prod_{i=1}^{n} p(y_i|x_i, \beta_0, \beta_1, \sigma^2)\right] \times p(\beta_0, \beta_1, \sigma^2) \\
p(\beta_0, \beta_1, \sigma^2|\boldsymbol{y}) &\propto \left[\prod_{i=1}^{n} \frac{1}{\sqrt{\sigma^2}} \exp\left(-\frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right)\right] \times \frac{1}{\sigma^2} \\
p(\beta_0, \beta_1, \sigma^2|\boldsymbol{y}) &\propto \frac{1}{(\sigma^2)^{\frac{n}{2}}} \times \exp\left(-\frac{\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right) \times \frac{1}{\sigma^2} \\
p(\beta_0, \beta_1, \sigma^2|\boldsymbol{y}) &\propto \frac{1}{(\sigma^2)^{\frac{n}{2}+1}} \times \exp\left(-\frac{\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right) \\
p(\beta_0, \beta_1, \sigma^2|\boldsymbol{y}) &\propto \frac{1}{(\sigma^2)^{\frac{n+2}{2}}} \times \exp\left(-\frac{\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}\right)
\end{aligned}$$

To obtain the marginal distribution for any of the parameter, one needs to integrate out the other parameters from joint posterior distribution. The posterior distribution for the parameter β_0 is given by:

$$p(\beta_0 \mid \boldsymbol{y}) = \int_0^{\infty} \left(\int_{-\infty}^{\infty} p(\beta_0, \beta_1, \sigma^2 \mid \boldsymbol{y})\, d\beta_1\right) d\sigma^2$$

The detailed derivation for the integration will be provided in the appendix. The posterior marginal distribution for the parameter β_0 is then given by:

$$\beta_0 \mid \boldsymbol{y} \sim t\left(n-2, \hat{\beta}_0, \hat{\sigma}^2\left(\frac{1}{n}+\frac{\bar{x}^2}{S_{xx}}\right)\right) = t\left(n-2, \hat{\beta}_0, (\mathrm{se}_{\beta_0})^2\right)$$

To obtain the marginal posterior distribution for the parameter β_1 the same procedure is needed:

$$p(\beta_1 \mid \boldsymbol{y}) = \int_0^\infty \left(\int_{-\infty}^\infty p(\beta_0, \beta_1, \sigma^2 \mid \boldsymbol{y})\, d\beta_0\right) d\sigma^2$$

The posterior distribution is also a student-t distribution given by:

$$\beta_1 \mid \boldsymbol{y} \sim t\left(n-2, \hat{\beta}_1, \frac{\hat{\sigma}^2}{S_{xx}}\right) = t\left(n-2, \hat{\beta}_1, (\mathrm{se}_{\beta_1})^2\right)$$

Finally, to obtain the marginal distribution for σ^2 the same procedure is followed such that:

$$p(\sigma^2 \mid \boldsymbol{y}) = \int_{-\infty}^\infty \left(\int_{-\infty}^\infty p(\beta_0, \beta_1, \sigma^2 \mid \boldsymbol{y})\, d\beta_0\right) d\beta_1$$

it can be shown that the distribution of σ^2 is the inverse Gamma distribution therefore we define τ the precision parameter which is the reciprocal of σ^2 such that $\tau = \frac{1}{\sigma^2}$. The marginal posterior distribution for τ is then given by:

$$\tau \mid \boldsymbol{y} \sim \mathrm{Gamma}\left(\frac{n-2}{2}, \frac{\mathrm{SSE}}{2}\right)$$

The detailed derivation for all of the parameters' marginal distributions will be provided in the appendix. From the marginal distributions of the parameters β_0, β_1 it is possible to see the link between the Bayesian Linear Regression model and the frequentist Linear Regression model as the scale parameter for each of those parameters is their respective standard error under the frequentist OLS estimation. Similarly, the center of the Student-t distribution for each those parameter represent their respective coefficient estimate under the frequentist OLS estimation. This connection highlights how Bayesian Linear Regression extends and generalizes the frequentist OLS framework by incorporating prior information and expressing uncertainty in the parameter estimates. The advantages of Bayesian Linear Regression can be observed here as prior knowledge of the problem can be introduced in the model and the resulting posterior distribution of the parameters is based on the prior knowledge and the observed data. As new data becomes available the posterior becomes the prior of the new model which has a new updated posterior and so on. This makes the Bayesian paradigm especially suitable for time series dataset as well as online dataset. Introducing the concepts of Bayesian Linear Regression was important as Gaussian Process Regression heavily relies on concept of Bayesian Linear Regression but extends the idea of parameters vector following a prior distribution and once updated with the data exhibits a posterior distribution to the realm of functions. It will be explained in more details in the following section.

2.2.2 Theoretical Aspects of Gaussian Process Regression

2.2.2.1 Model predictive posterior distribution

As discussed above, Gaussian processes are a way to model probability distributions over functions. This can be used in the context of Bayesian Regression in order to create a Gaussian Process Regression model. We start by defining the data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$. The training set is assumed to be independently and identically distributed following an unknown distribution. The Gaussian Process Regression model is defined as follows:

$$y_i = h(x_i) + \epsilon_i \tag{2.2.5}$$

where:

- y_i is the dependent variable associated with the i-th individual.
- x_i is the independent variable associated with the i-th individual.
- $h(\cdot)$ is a random function drawn from a Gaussian process prior, representing the underlying relationship between (x_i) and (y_i).
- ϵ_i are the error terms which are identically and independently distributed following $\mathcal{N}(0, \sigma^2)$.

Using the concepts detailed about Bayesian Linear Regression, we assume a prior distribution for the function $h(\cdot)$. The prior distribution for the function will be a Gaussian process with some mean and covariance function given by:

$$h(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)) \tag{2.2.6}$$

As Bayesian statistics suggests the prior belief about a dataset is updated by new observations. Therefore, let $\mathcal{D}^* = \{(x_i^*, y_i^*)\}_{i=1}^{n_*}$ be the test set which also follows an unknown distribution as $\mathcal{D}$. Using the concepts learned from Bayesian Linear Regression, one can compute the posterior predictive distribution from which $h(\cdot)$ will be drawn. We consider that the prior distribution of $h(\cdot)$ is a zero-mean Gaussian process prior with a covariance function given by $k(\cdot, \cdot)$. This means that for any finite set of input points, the function values follow a multivariate Gaussian distribution [2], [17]. The joint distribution for the whole set of points (the training and test sets) is given by:

$$\begin{bmatrix} h \\ h_* \end{bmatrix} \Bigg| \boldsymbol{X}, \boldsymbol{X}_* \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) \end{bmatrix}\right)$$

- $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$: The covariance matrix for the matrix training inputs $\boldsymbol{X}$. It represents the covariance between each pair of training data points.

- $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$: The covariance matrix between the matrix of training inputs $\boldsymbol{X}$ and the matrix test inputs $\boldsymbol{X}_*$. It captures the similarity between the training data and test data .

- $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})$: The covariance matrix between the matrix of test inputs $\boldsymbol{X}_*$ and the matrix of training inputs $\boldsymbol{X}$. This is the transpose of $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$ because the covariance matrix $\boldsymbol{K}$ is symmetric.

- $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*)$: The covariance matrix of the matrix test inputs $\boldsymbol{X}_*$. It represents the covariance between each pair of test data points.

It was also assumed that the noise were identically and independently distributed such that:

$$\begin{bmatrix} \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon}_* \end{bmatrix} \Bigg| \boldsymbol{X}, \boldsymbol{X}_* \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \sigma^2 I & \boldsymbol{0} \\ \boldsymbol{0}^\top & \sigma^2 I \end{bmatrix}\right)$$

Since the sum of independent Gaussian random variables also follows a Gaussian distribution we therefore obtain:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_* \end{bmatrix} \Bigg| \boldsymbol{X}, \boldsymbol{X}_* = \begin{bmatrix} h \\ h_* \end{bmatrix} + \begin{bmatrix} \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I} & \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) + \sigma^2 \boldsymbol{I} \end{bmatrix}\right)$$

Using the concept highlighted from Bayesian Linear Regression, the posterior distribution of $\vec{y}_*$ is given by conditioning it on the already observed outcomes which constitute the prior belief we have about our dataset just as it is done in the case of conditional multivariate Gaussian distribution [8]. We obtain for the posterior distribution of $\vec{y}_*$ the following:

$$\boldsymbol{y}_* \mid \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

where

$$\begin{aligned} \boldsymbol{\mu}_* &= (\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \end{aligned}$$

The detailed derivation for the mean and covariance is based on the concept of conditional distribution and will be provided for in the appendix.

2.2.2.2 Covariance Function

The formula for the mean and covariance matrix show that $\boldsymbol{K}(\cdot, \cdot)$ (also called the Gram matrix) plays an important aspect in the use of Gaussian Process Regression for prediction of unseen data [1], [19]. It is then crucial to explain covariance functions in more details and how they influence how Gaussian Process Regression models work. We define x and x' to be points in Euclidean space. Gaussian Process Regression use the kernel function in order to define a prior covariance over two functions value such that:

$$Cov[f(x), f(x')] = k(x, x')$$

Because the Gaussian Process prior is a distribution over function it is important to think of the covariance used in the prior as the covariance between the value of the function evaluated at two different inputs. If we consider two inputs points, x_1 and x_2 which are considered to be similar therefore the covariance between the value of the function evaluated at two different inputs should be high. Under this consideration, some functions, going smoothly from x_1 to x_2 are more likely. However, because the covariance function enforces correlation between the output values for two similar inputs, functions that have large jumps between similar inputs points go against that correlation assumption and hence have low prior probability under the Gaussian Process Prior. Because the Gaussian Process prior is used in determining the posterior as new input points become available, the covariance function then ultimately influences the prediction properties of the model [2], [7], [19]. There are many different kernel functions and custom ones can be created to fit a particular problem. Covariance functions can be categorized based on their properties [19]:

- A covariance function is *stationary* if it depends only on the distance between inputs, i.e., $k(x, x') = k(x - x')$, and is invariant to translations in the input space.
- A covariance function is *isotropic* if it depends only on the magnitude of the distance, $k(x, x') = k(|x - x'|)$, and not the direction.

A variety of kernel functions are used in GPR, each with its own characteristics and suitability for different problems. Below, we provide an overview of some commonly used kernels mainly based on the work of Duvenaud [19]:

1. **Radial Basis Function (RBF) Kernel**:
   The RBF kernel, also known as the squared exponential kernel, is given by:

$$k_{\text{RBF}}(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right),$$

   where ℓ is the length-scale parameter. The length-scale value determines how quickly the correlation between points decays with distance. Because it is infinitely differentiable, this kernel assumes smooth functions and is widely used due to its flexibility. It is stationary and isotropic.

2. **Matérn Kernel**:
   Because when modelling physical systems the assumption of smoothness can lead to wrong prediction, the Matérn kernel generalizes the RBF kernel and is controlled by a smoothness parameter ν:

$$k_{\text{Matérn}}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\|x - x'\|}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}\|x - x'\|}{\ell}\right),$$

where K_ν is the modified Bessel function of the second kind. For $\nu = 1/2$, it reduces to the absolute exponential kernel due to the mathematical properties of the Gamma and Bessel functions; for $\nu \to \infty$, it approaches the RBF kernel. The Matérn Kernel is stationary and isotropic.

3. **Absolute Exponential Kernel**:
   The absolute exponential kernel is a special case of the Matérn kernel when $\nu = 1/2$. It is given by:
   $$k_{\text{AbsExp}}(x, x') = \exp\left(-\frac{\|x - x'\|}{\ell}\right),$$
   where ℓ is the length-scale parameter. This kernel models rougher, less smooth functions compared to the RBF kernel. it is stationary and isotropic.

4. **Rational Quadratic Kernel**:
   The rational quadratic kernel:
   $$k_{\text{RQ}}(x, x') = \left(1 + \frac{\|x - x'\|^2}{2\alpha\ell^2}\right)^{-\alpha},$$
   This kernel can be seen as a scale mixture of RBF kernels. In fact, the Rational Quadratic Kernel works as an infinite sum of RBF Kernel with different length scale (scale mixture). It has two parameters, the scale mixture parameter α and the length scale parameter ℓ. The length scale parameter ℓ behaves just as in a RBF Kernel. The scale mixture parameter modulates the relative weighting of the different length scales. As it increases when everything else stays constant it makes the value of the kernel decay faster between distant points. As $\alpha \to \infty$ the Rational Quadratic Kernel converges to a RBF Kernel. However, when it decreases when everything else stays constant it makes the value of the kernel decay slower even for distant points capturing short-range and long-range dependencies. It is usually used to model functions that exhibits long term trends but short term fluctuations. Also, the value of α and ℓ are not chosen arbitrarly and are often subject to hyperparameters optimization via maximization of marginal likelihood. The Rational Quadratic Kernel it is stationary and isotropic.

5. **Exponential Sine-Squared Kernel**:
   This kernel is useful for modeling periodic functions:
   $$k_{\text{ExpSinSq}}(x, x') = \exp\left(-\frac{2\sin^2\left(\frac{\pi|x-x'|}{p}\right)}{\ell^2}\right),$$
   It is useful for modeling periodic data as it capture repeating patterns such as seasonal or cyclical trends. The function can encode the periodicity explicitly with p the periodicity parameter which ensures that the kernel value (and therefore the correlation between points) repeats with a fixed period and ℓ is the length-scale parameter that governs both the smoothness of the periodic pattern within each cycle and the rate at which the correlation between points decays as they become increasingly misaligned in their periodic phase. A larger ℓ results in smoother transitions and slower amplitude decay, while a smaller ℓ leads to sharper transitions and faster decay of correlation. Since it depends on the difference between the points, it is stationary.

6. **Dot Product Kernel**:
   The dot product kernel is defined as:

$$k_{\text{DotProduct}}(x, x') = x \cdot x',$$

   It is a non-stationary kernel meaning the value of the function depends on the absolute location of the inputs rather than their relative distance. It is particularly useful when the data exhibits linear relationships. It calculates the similarity between two data points as the dot product of their feature vectors. This kernel is widely used for linear models. This kernel is neither isotropic nor stationary.

7. **Polynomial Kernel**:
   An extension of the dot product kernel, the polynomial kernel is given by:

$$k_{\text{Poly}}(x, x') = (x \cdot x' + c)^d,$$

   where c is a constant and d is the degree of the polynomial. The polynomial kernel allows to model non-linear relationship in the data by introducing higher order terms. It captures interactions between features. The constant term is the bias which prevents the value of the kernel from being totally dominated by small inputs or negative dot product. The kernel is non-stationary just like the dot product kernel but allows for more flexibility when it comes to capturing non-linearity in the data as one can just try different value for the degree of the polynomial. This kernel is neither stationary nor isotropic.

2.2.2.3 Model Predictive Variance and Confidence Intervals

In Gaussian Process Regression, not only do we obtain a predictive mean for new data points but also a predictive variance, it is important to quantify the uncertainty associated with the model's predictions. This predictive variance is also crucial for constructing credible intervals and understanding the reliability of the model's predictions [2], [11], [13]. We call $\boldsymbol{X_*}$ the matrix of test inputs with m observations. The test inputs posterior distribution is characterized by parameters $\boldsymbol{\mu_*}$ the posterior mean vector and $\boldsymbol{\Sigma_*}$ the posterior covariance matrix which are given by:

$$\boldsymbol{\mu_*} = (\boldsymbol{K}(\boldsymbol{X_*}, \boldsymbol{X})\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{y}$$
$$\boldsymbol{\Sigma_*} = \boldsymbol{K}(\boldsymbol{X_*}, \boldsymbol{X_*}) - \boldsymbol{K}(\boldsymbol{X_*}, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X_*})$$

where:

- $\boldsymbol{K(X_*, X)}$: The covariance vector between all test inputs represented by the matrix $\boldsymbol{X_*}$ and the matrix of training inputs $\boldsymbol{X}$. It has a $m \times n$ dimension.
- $\boldsymbol{K(X, X)}$: The $n \times n$ covariance matrix of the training inputs.
- $\boldsymbol{K(X_*, X_*)}$: The covariance matrix between all test inputs.
- σ^2: The variance of the observation noise.

- $\boldsymbol{I}$: The $n \times n$ identity matrix.
- $\boldsymbol{y}$: The vector of observed outputs corresponding to the training inputs, i.e., $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\top$.

For a single new observation x_* its predictive mean and variance is given as follows:

$$\mu(x_*) = \boldsymbol{k_*}^\top (\boldsymbol{K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}$$
$$\sigma(x_*) = k(x_*, x_*) - \boldsymbol{k_*}^\top (\boldsymbol{K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k_*}$$

where:

- $\boldsymbol{k_*}$: The covariance vector between the test input x_* and each of the n training inputs $\{x_i\}_{i=1}^n$ of dimension $n \times 1$, defined as: $\boldsymbol{k_*} = [k(x_*, x_1), k(x_*, x_2), \ldots k(x_*, x_n)]^\top$
- $\boldsymbol{K(X, X)}$: The $n \times n$ covariance matrix of the training inputs.
- $k(x_*, x_*)$: The covariance of the test input x_* with itself.
- σ^2: The variance of the observation noise.
- $\boldsymbol{I}$: The $n \times n$ identity matrix.
- $\boldsymbol{y}$: The vector of observed outputs corresponding to the training inputs, i.e., $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\top$.

The predictive mean for the new observation depends on the similarity between the new observation and the training points normalized by the combined covariance of the training data and observation noise [2]. Consequently, as the test point becomes more similar to the training points, the predictive mean becomes more accurate. The predictive variance for the new observation x_* is dependent on the similarity between this new observation and the training data points calculated via the kernel function and is of particular important. The similarity measures how closely related the new observation is to the training data points.

We introduce the information gain term which is given as follows for $\boldsymbol{X_*}$ the matrix of test inputs:

$$\boldsymbol{K(X_*, X)}(\boldsymbol{K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{K(X, X_*)}$$

and as follows for a single test input:

$$\boldsymbol{k_*}^\top (\boldsymbol{K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k_*}$$

The information gain term is dependent on new observations, therefore as the new observations lie close to the training data points the value of the information gain term increases and the whole predictive variance value for the new observations decreases. Alternatively, as the new observations fall far from the training data points the value for the information gain term

decreases increasing the predictive variance value for the new data points. The overall confidence in the model's prediction for a new observation is then highly dependent on how closely related new data points are to the training data points. This behavior shows the importance of the specification of the covariance function in the predicting but also measuring the model's credible interval. Additionally, the predictive variance which quantifies the uncertainty in the model's prediction is defined for $\boldsymbol{X_*}$ the matrix of new input points as:

$$\boldsymbol{K(X_*, X)(K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{K(X, X_*)}$$

and it is defined for x_* a single new input point as:

$$\boldsymbol{k_*}^\top (\boldsymbol{K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k_*}$$

The predictive variance is dependent on two terms the first term which is the prior belief about how much the function might vary at the new input points and the information gain term introduced earlier. The predictive variance therefore account for the model's uncertainty as when there a low amount of training data in certain region of the input space, $\boldsymbol{K(X_*, X(K(X, X)} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{K(X, X_*)}$ is small due to $\boldsymbol{K(X_*, X)}$ and $\boldsymbol{K(X, X_*)}$ being small. The predictive variance remains high as the information gain term substracted from the prior variance is small. Moreover, the predictive variance accounts for the noise in the training inputs. In fact with everything else staying constant, if $\sigma^2 \boldsymbol{I}$ which represents the noise in the training inputs increase, the whole information gain term decreases. The predictive variance remains high as the information gain term substracted from the prior variance is small. The predictive variance thus provides a comprehensive measure of the model's uncertainty that combines both model uncertainty due to limited training data coverage in the input space and inherent training data noise.

The posterior predictive distribution of the new observation is a Gaussian distribution whose parameters are the predictive mean and variance are $\mu(x_*)$ and $\sigma(x_*)$. One can utilize the properties of a Gaussian distribution to create a credible interval around the value predicted by the model. A common approach is to construct a $(1 - \alpha) \times 100\%$ credible interval around the new observation which is given by:

$$\mu(x_*) \pm z_{\alpha/2} \sigma(x_*)$$

with $z_{\alpha/2}$ being the critical value from a standard Gaussian distribution. Because the confidenece region is directly proportional to the characteristic variance as the new observation lies close to the training points the characteristic variance will be low resulting in a narrow credible region and low uncertainty about the prediction given by the model. Alternatively, as the new observation falls far from the training points the predictive variance will be high resulting in a wider credible region around the predicted value reflecting the model uncertainty due to the lack of information from the training data.

2.2.2.4 Model Error Bound

Gaussian Process Regression is a powerful as it gives a credible interval for predictions. The credible interval, it was seen earlier, depends on the characteristic variance of the new predicted point. The variance, however, can be misleading. In fact, the characteristic variance of a new predicted points depends on the chosen covariance function. However, the choice of the covariance function is not trivial and a misspecified covariance function can thus lead to a

variance and consequently to a credible interval for the prediction that is wrong [11]. In certain scenarios, the knowledge of highest possible error, an upper bound for the model error is necessary to decide whether or not one would choose Gaussian Process Regression to model a system. In this section, we present the important work of Beckers, Umlauft, and Hirche [11] in single task Gaussian Process Regression framework but it can be applied to the multi-task Gaussian Process Regression framework as done in the original text. We define two Gaussian Process Regression models $\mathcal{GP}^1$ and $\mathcal{GP}^2$, for simplicity, we specify the mean of the two models to be 0. The model $\mathcal{GP}^1$ is an unknown Gaussian Process Regression model with an unknown covariance function with unknown parameters for the covariance function. The model $\mathcal{GP}^1$ is the true underlying process one is trying to model. The model $\mathcal{GP}^2$ uses an estimated covariance with estimated hyperparameters for the covariance function. We define the two models as follows:

$$\begin{aligned} \boldsymbol{y} &= f(\boldsymbol{x}) \\ f(\boldsymbol{x}) &\sim \mathcal{GP}^1\big(0, k(\varphi, x, x')\big) \end{aligned}$$

- $\boldsymbol{y}$ is the vector of true output.
- $\boldsymbol{x}$ is the vector of independent variable associated with the output.
- $\mathcal{GP}^1$ is the Gaussian Process from which $f(\cdot)$ is drawn and represent the true underlying process.
- k is the true covariance function.
- $\boldsymbol{\varphi}$ is the vector of hyperparameters of the covariance function k. The specific number and nature of these hyperparameters depend on the particular form of the covariance function k. Φ the set (or domain) of all feasible values for $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi} \in \Phi$.

$$\begin{aligned} \hat{\boldsymbol{y}} &= f(\boldsymbol{x}) + \boldsymbol{\epsilon} \\ f(\boldsymbol{x}) &\sim \mathcal{GP}^2\big(0, \hat{k}(\varphi, x, x')\big) \end{aligned}$$

- $\hat{\boldsymbol{y}}$ is the vector of estimated output.
- $\boldsymbol{x}$ is the vector of independent variable introduced in the estimated model.
- $\mathcal{GP}^2$ is the estimated Gaussian Process from which $f(\boldsymbol{x})$ is drawn and we use to estimate $\hat{\boldsymbol{y}}$.
- $\hat{k}$ is the estimated covariance function.
- $\hat{\boldsymbol{\varphi}}$ is the vector of hyperparameters of the covariance function $\hat{k}$. The specific number and nature of these hyperparameters depend on the particular form of the covariance function $\hat{k}$.

- $\boldsymbol{\epsilon}$ is the vector of residuals for the estimated model

The goal is to compute the mean squared prediction error between $\boldsymbol{y}$ which is the true output coming from the true but unknown model $\mathcal{GP}^1$ and $\mu\left(\hat{\boldsymbol{y}}\right)$ which is the mean prediction made by the estimated model $\mathcal{GP}^2$ such that:

$$E\left[|| \boldsymbol{y} \mid (x, \mathcal{D}) - \mu(\hat{\boldsymbol{y}} \mid (x, \mathcal{D}))\, ||^2\right]$$

It was shown that under some assumptions an upper bound can be derived for the mean squared prediction error.

- **Assumption 1** we have knowledge about a possible set of covariance functions $\tilde{\mathcal{K}} = \{\tilde{k}_1, \tilde{k}_2, \dots \tilde{k}_j\}$. Each covariance function in the possible set of covariance functions has a specific vector of hyperparameters depending on specific covariance function such that $\{\tilde{\boldsymbol{\varphi}}_{\mathbf{1}}, \dots, \tilde{\boldsymbol{\varphi}}_{\boldsymbol{j}}\}$ respectively correspond to $\tilde{k}_1 \dots \tilde{k}_j$. We also have a possible set of all feasible values (or domain) for their respective vectors of hyperparameters $\tilde{\Phi} = \{\tilde{\Phi}_1, \tilde{\Phi}_2, \dots \tilde{\Phi}_j\}$. Similarly each vector of hyperparameters for the different covariances has a set of feasible values such that $\tilde{\boldsymbol{\varphi}}_{\mathbf{1}} \in \tilde{\Phi}_1, \tilde{\boldsymbol{\varphi}}_{\mathbf{2}} \in \tilde{\Phi}_2, \dots, \tilde{\boldsymbol{\varphi}}_{\boldsymbol{j}} \in \tilde{\Phi}_j$

Following the first assumption, it is not necessary to know the exact covariance function of $\mathcal{GP}^1$ it just needs to be part of the set of possible covariance functions. The exact hypermarameters can also be unknown but needs to be part of the possible set of all feasible values. One's main focus is on quantifying the mean squared prediction error (MSPE) which is the expectation of the squared difference between $\boldsymbol{y}$ and the mean prediction of the estimated model $\mu(\hat{\boldsymbol{y}})$. Here, $\mu(\hat{\boldsymbol{y}})$ denotes the mean of the predictive distribution for $\boldsymbol{y}$ under $\mathcal{GP}^2$. The mean squared prediction error between $\boldsymbol{y}$ following $\mathcal{GP}^1$ and $\mu(\hat{\boldsymbol{y}})$ given by $\mathcal{GP}^2$ is given by:

$$E\left[|| \boldsymbol{y} \mid (x, \mathcal{D}) - \mu(\hat{\boldsymbol{y}} \mid (x, \mathcal{D}))\, ||^2\right] = E\left[\boldsymbol{y}^2\right] - 2E\left[\boldsymbol{y}\mu(\hat{\boldsymbol{y}})\right] + E[\mu(\hat{\boldsymbol{y}})^2]$$

Developing the expression further, we have for each term in the above equation the following:

$$E\big[\boldsymbol{y}^2\big] = k(x_*, x_*; \boldsymbol{\varphi}),$$

$$E\big[\boldsymbol{y}\, \mu(\hat{\boldsymbol{y}})\big] = \hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})^\top\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \boldsymbol{k}_{\boldsymbol{*}}(\boldsymbol{\varphi}),$$

$$E\big[\mu(\hat{\boldsymbol{y}})^2\big] = \hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})^\top\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \boldsymbol{K}(\boldsymbol{\varphi})\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}}),$$

$$\text{MSPE} = k(x_* x_*; \boldsymbol{\varphi}) \;-\; 2\left[\hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})^\top\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \boldsymbol{k}_{\boldsymbol{*}}(\boldsymbol{\varphi})\right] \;+\; \hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})^\top\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \boldsymbol{K}(\boldsymbol{\varphi})\, \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1}\, \hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})$$

where:

- $k(x_*, x_*; \boldsymbol{\varphi})$: The variance of the true GP at x_*. $\boldsymbol{\varphi}$ is the true vector of hyperparameters corresponding to the true covariance function k.

- $\hat{\boldsymbol{k}}_{\boldsymbol{*}}(\hat{\boldsymbol{\varphi}})$: The vector covariance between the test point x_* and all the training inputs under the estimated GP. $\hat{\boldsymbol{\varphi}}$ is the vector of hyperparameters corresponding to the covariance function $\hat{k}$ chosen under the estimated GP.

- $\hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})$: The covariance matrix of all the training inputs under the estimated GP. $\hat{\boldsymbol{\varphi}}$ is the vector of hyperparameters corresponding to the covariance function $\hat{k}$ chosen under the estimated GP.

- $\boldsymbol{k}_*(\boldsymbol{\varphi})$: The vector covariance between the test point x and all the training inputs under the true GP. $\boldsymbol{\varphi}$ is the vector of hyperparameters corresponding to the covariance function k.

- $\boldsymbol{K}(\boldsymbol{\varphi})$: The covariance matrix of all training inputs under the true GP. $\boldsymbol{\varphi}$ is the vector of hyperparameters corresponding to the covariance function k.

To derive the upper bound for the MSPE it is then needed to maximize each term. We define $\hat{k}$ the covariance function of the estimated model $\mathcal{GP}^2$ and $\hat{\boldsymbol{\varphi}}$ its vector of hyperparameters. One then needs to find the covariance function and the vector of hyperparameters for true underlying model $\mathcal{GP}^1$ that maximizes the MSPE between to find the upper bound for the error between $\mathcal{GP}^1$ and $\mathcal{GP}^2$. We define k the covariance function for the true underlying model $\mathcal{GP}^1$ and $\boldsymbol{\varphi}$ its vector of hyperparameters. The covariance function k and its vector of hyperparameters $\boldsymbol{\varphi}$ need not to be known. As said previously, there is a set of possible covariance functions and set of ranges for the vector of hyperparameters of each covariance function, k needs to belong to the set of possible covariance functions and $\boldsymbol{\varphi}$ its vector of hyperparameters needs to be in the set of possible values for it. If the chosen covariance function $\hat{k}$ for the estimated model $\mathcal{GP}^2$ and its vector of hyperparameters $\hat{\boldsymbol{\varphi}}$ are correct then $k = \hat{k}$ and $\boldsymbol{\varphi} = \hat{\boldsymbol{\varphi}}$. The MSPE is then given by:

$$\text{MSPE} = k(x_*, x_*; \boldsymbol{\varphi}) \; - \; 2\left[\hat{\boldsymbol{k}}_*(\hat{\boldsymbol{\varphi}})^\top \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})\right] \; + \; \hat{\boldsymbol{k}}_*(\hat{\boldsymbol{\varphi}})^\top \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1} \boldsymbol{K}(\boldsymbol{\varphi}) \hat{\boldsymbol{K}}(\hat{\boldsymbol{\varphi}})^{-1} \hat{\boldsymbol{k}}_*(\hat{\boldsymbol{\varphi}})$$

$$\text{MSPE} = k(x_*, x_*; \boldsymbol{\varphi}) \; - \; 2\left[\boldsymbol{k}_*(\boldsymbol{\varphi})^\top \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})\right] \; + \; \boldsymbol{k}_*(\boldsymbol{\varphi})^\top \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{K}(\boldsymbol{\varphi}) \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})$$

$$\text{MSPE} = k(x_*, x_*; \boldsymbol{\varphi}) \; - \; 2\left[\boldsymbol{k}_*(\boldsymbol{\varphi})^\top \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})\right] \; + \; \boldsymbol{k}_*(\boldsymbol{\varphi})^\top \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})$$

$$\text{MSPE} = k(x_*, x_*; \boldsymbol{\varphi}) \; - \; \left[\boldsymbol{k}_*(\boldsymbol{\varphi})^\top \boldsymbol{K}(\boldsymbol{\varphi})^{-1} \boldsymbol{k}_*(\boldsymbol{\varphi})\right]$$

The above derivation shows that, for a well-specified model, the MSPE at a single observation is equal to that observation's posterior variance. Extending this result to the entire set of points, the total MSPE is given by the sum of the posterior variances across all points, which is equivalent to the trace of the posterior covariance matrix. However, for not well specified models we seek to find the maximum of the mean squared prediction error (MSPE). Because this error expression is a sum of terms (some appearing with a positive sign, others with a negative sign), we strive to *maximize* the positively signed terms and *minimize* the negatively signed ones. **Assumption 1** guarantees the positivity and feasibility of the kernel values, while two additional assumptions ensure that this maximization can be carried out in closed form:

- **Assumption 2** places the model hyperparameters in a convex (often rectangular) set.

- **Assumption 3** each covariance funtion in the $\tilde{k}$ in the set of possible covariance functions $\tilde{K}$ is strictly monotonically increasing.

Those last two assumptions tie with the first assumption and justify working with a set of all feasible values (or domain) for the vector of hyperparameters for the covariance functions in the possible set of covariance functions. In fact, because for each covariance function its vector of hyperparameters is a set of all feasible values (or domain) and that for each covariance function there is a strict monotonicity with respect to its vector of hyperparameters one only needs to evaluate the covariance function at the upper boundary of the set of all feasible values (or domain) of its vector of hyperparameters. This procedure is performed for each covariance function, and the overall upper bound on the MSPE is determined by whichever covariance function and hyperparameters combination maximizes it. Consequently, one obtains a *closed-form* solution for the MSPE.:

$$\max_{\hat{k}\in\tilde{K},\hat{\boldsymbol{\varphi}}\in\tilde{\Phi}} \mathrm{MSPE}(k\left(\varphi\right),\hat{k}\left(\hat{\boldsymbol{\varphi}}\right)) \;=\; \max\Big\{\mathrm{MSPE}\big(k\left(\boldsymbol{\varphi}\right),\hat{k}_1\left(\hat{\boldsymbol{\varphi}}_{\mathbf{1}}\right)\big),\ldots,\mathrm{MSPE}\big(k\left(\boldsymbol{\varphi}\right),\hat{k}_j\left(\hat{\boldsymbol{\varphi}}_{\boldsymbol{j}}\right)\big)\Big\},$$

which can be computed analytically rather than requiring iterative optimization. This worst-case scenario yields a maximum possible predictive variance, you limit how large the predictive variance can grow. This approach gives a more robust indication about the stability of the model than only pointwise credible interval which is crucial in certain applications. While credible intervals quantify uncertainty for individual predictions, the MSPE-based upper bound ensures that, on average, the discrepancy between the true values and the model's predictions does not exceed a known limit.

2.2.2.5 Hyperparameter Optimization

Gaussian Process Regression prior highly depends on the specified covariance function. The covariance functions can have multiple hyperparameters. It is then crucial to find hyperparameters that yield the best model. There are many different ways to optimize a function given a set of hyperparameters, however, one approach that is commonly used with Gaussian Process Regression is to optimize the log marginal likelihood function [2], [13] conditionally on the data matrix X and the set of hyperparameters ϕ. The marginal log likelihood function is given by:

$$\log p(\boldsymbol{y}\mid\boldsymbol{X},\boldsymbol{\varphi}) = -\tfrac{1}{2}\,\boldsymbol{Y}^{\top}\big(\boldsymbol{K}+\sigma^2\boldsymbol{I}\big)^{-1}\boldsymbol{Y} \;-\; \tfrac{1}{2}\,\log\big|\boldsymbol{K}+\sigma^2\boldsymbol{I}\big| \;-\; \tfrac{n}{2}\,\log(2\pi)$$

The terms in the log marginal likelihood, the first is depending on the data it rewards a good fit of the data, the second term penalizes for model complexity and the final is a normalization constant depending on the number of observations. Because a closed for solution for the problem is not possible, it is solved through gradient based optimization algorithms that try to minimize the function. However, because non-convex in general there is no guarantee to find a global optimum for the optimization problem at hand.

2.3 Multi Task Gaussian Process Regression

A natural next step after exploring single-task Gaussian Process Regression (GPR) is to consider scenarios where multiple, potentially correlated tasks are of interest. Multi-task learning is an area of research that has gained popularity over the past few years. In many real-world problems, ranging from multi-sensor data analysis to multi-output regression in engineering or biomedicine, we often have several response variables that we would like to

model simultaneously. By leveraging the shared structure among these related tasks, we can improve predictive performance and gain deeper insights into the underlying processes [4]. However, an important aspect to consider when building a multi-task learning model is the relatedness between different tasks. In fact, knowing which tasks are related is not easy and it can be problematic to assume relatedness between all tasks and simply learning them together can be problematic [3], [21]. Therefore, it will be important to build a model capable of leveraging the relatedness between tasks when they are related while not will not hurting performance when these tasks are unrelated meaning the performance of the model for unrelated tasks should be at least as good as the "no transfer" case.

This motivates the study of Multi-Task Gaussian Process Regression (also referred to as Multi-Output Gaussian Process Regression), which generalizes the single-task framework to handle multiple correlated tasks. In what follows, we will outline the key concepts, theoretical underpinnings, and practical considerations of multi-task Gaussian processes.

2.3.1 Theoretical Framework

2.3.1.1 The model

We remember from earlier sections that GPR is used to model an independent variable using a model defined as follows:

$$y_i = h(x_i) + \epsilon_i \tag{2.3.1}$$

where:

- y_i is the dependent variable associated with the i-th individual.
- x_i is the independent variable associated with the i-th individual.
- $h(\cdot)$ is a random function drawn from a Gaussian process prior, representing the underlying relationship between (x_i) and (y_i).
- ϵ_i are the error terms which are identically and independently distributed following $\mathcal{N}(0, \sigma^2)$.

with:

$$h(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)) \tag{2.3.2}$$

One can extend this model to a multi-task learning framework. We define M the total number of tasks one is interested in modelling, N the total number of observations (we consider for simplicity that each task has the same number of observations). Let x_{ij} be the observation i-th observation out of N observations for the j-th out of M tasks [20]. Let y_{ij} be the response variable for the i-th observation for the j-th task. We define $\boldsymbol{y}$ in this framework as the vector of response variable for all observations and all tasks such that $\boldsymbol{y} = [y_{11}, y_{21}, \ldots, y_{n1}, \ldots, y_{1m}, \ldots, y_{nm}]^\top$. We define the vector of response variables for each j-th out of M tasks as $\boldsymbol{y_j} = [y_{1j}, y_{2j}, \ldots, y_{nj}]^\top$. The output vector $\boldsymbol{y}$ can also be defined as a matrix $\boldsymbol{Y}$ with dimensions $n \times m$ where each column represented the vector of outputs for a

particular task. For the inputs variables, let x_{ij} be the input variable for the i-th observation for the j-th task. We define $\boldsymbol{x}$ in this framework as the vector of independent variable for all observations and all tasks such that $\boldsymbol{x} = [x_{11}, x_{21}, \ldots, x_{n1}, \ldots, x_{1m}, \ldots, x_{nm}]^\top$. We define $\boldsymbol{x_j} = [x_{1j}, x_{2j}, \ldots, x_{nj}]^\top$ which is the vector of N inputs for the j-th out of M tasks. We can also define a matrix $\boldsymbol{X}$ with dimension $n \times m$ where each column represented the vector of inputs for a particular task. However, this is the case where the inputs are univariate. In the case where the inputs are multivariate with P dimensions we have a matrix $\boldsymbol{X}$ with dimension $n \times (p \times m)$. Let's note in the Multi-Task Gaussian Process Regression framework, if tasks are indeed in the same input space meaning that the independent variables for every task are the same, the size of the matrix reduces and we obtain $\boldsymbol{X}$ with dimension $n \times p$. Each row of $\boldsymbol{X}$ corresponds to an observation, and each column is a feature, regardless of the task. Then, the multi-task aspect is handled entirely on the output side in $\boldsymbol{Y}$ or through an additional task covariance structure in your Gaussian Process model. While this isotopic framework is mathematically elegant and has powered many applications, it is fundamentally limited whenever tasks draw on different feature sets or modalities. Current advances in Multi-Task Gaussian Process Regression present the disadvantages of focusing heavily on the isotopic case meaning Multi-Task Gaussian Process Regression where all tasks share the same input domain. In practice, however, it is common for different tasks to have different input domains [6], [15]. While current approaches are great are modeling Multi-Task Gaussian Process Regression in the isotopic case, they lack the flexibility to model Multi-Task Gaussian Process Regression in the heterotopic case meaning when tasks have different input domains. In Section 2.3.2.2 a more flexible model is proposed which can handle both the isotopic and the heterotopic case. Following our discussion on the general setup for Multi-Task Gaussian Process Regression , we present the model as follows:

$$y_{ij} = h_j(x_{ij}) + \epsilon_{ij}$$

where:

- y_{ij} is the dependent variable associated with the i-th individual for j-th task.
- x_{ij} is the independent variable associated with the i-th individual for j-th task.
- $h_j(\cdot)$ is a random function drawn from j-th Gaussian process prior associated with the j-th task, representing the underlying relationship between x_{ij} and y_{ij}.
- ϵ_{ij} are the error terms associated with the i-th individual for j-th task which are identically and independently distributed following $\mathcal{N}(0, \sigma^2)$. We define $\boldsymbol{\epsilon_j}$ to be the vector of error terms associated with the j-th out of M tasks.

extending the concepts introduced earlier about Gaussian Process Regression we therefore have:

$$h_j(\cdot) \sim \mathcal{GP}(m^j(\cdot), k^j(\cdot, \cdot))$$

where:

- $h_j(\cdot)$ is a random function drawn from j-th Gaussian process prior associated with the j-th task, representing the underlying relationship between x_{ij} and y_{ij}.

- $\mathcal{GP}(m^j(\cdot), k^j(\cdot,\cdot))$ is the Gaussian Process prior associated from which $h_j(\cdot)$. $m^j(\cdot)$ is the mean function associated with the Gaussian Process Prior and $k^j(\cdot,\cdot)$ is the covariance function associated with the Gaussian Process prior.

Since we are in Multi-Task Learning framework, $k^j(\cdot,\cdot)$ is of particular interest as it will represent not only covariance between inputs but also between tasks. We will give more details about the characteristics of $k(\cdot,\cdot)$ in following sections. After defining our framework we then obtain for our complete model the following:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y_1} = h_1(\boldsymbol{x_1}) + \boldsymbol{\epsilon_1} \\ \vdots \\ \boldsymbol{y_m} = h_m(\boldsymbol{x_m}) + \boldsymbol{\epsilon_m}, \end{bmatrix} = \begin{cases} h_1(\cdot) \sim \mathcal{GP}\big(m^1(\cdot), k^1(\cdot,\cdot)\big), \\ \vdots \\ h_m(\cdot) \sim \mathcal{GP}\big(m^m(\cdot), k^m(\cdot,\cdot)\big)\big), \end{cases}$$

We consider that the prior distribution of $h_j(\cdot)$ is a zero-mean Gaussian process prior with a covariance given by $k^j(\cdot,\cdot)$, therefore we obtain:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y_1} = h_1(\boldsymbol{x_1}) + \boldsymbol{\epsilon_1} \\ \vdots \\ \boldsymbol{y_m} = h_j(\boldsymbol{x_m}) + \boldsymbol{\epsilon_m}, \end{bmatrix} = \begin{cases} h_1(\cdot) \sim \mathcal{GP}\big(0, k^1(\cdot,\cdot)\big), \\ \vdots \\ h_m(\cdot) \sim \mathcal{GP}\big(0, k^m(\cdot,\cdot)\big)\big), \end{cases}$$

After defining the framework for the Multi-Task Gaussian Process Regression (MTGPR) model, it should be noted that the above equations correspond to a case where all task are modeled separately with M different GPR models. The objective, however, is to find one common model capable of predicting outputs for each task using the relatedness between tasks. Such model would therefore be given as follows:

$$\boldsymbol{y} = h(\boldsymbol{x}) + \boldsymbol{\epsilon}$$

with

$$h(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$$

where:

- $\boldsymbol{y} = [y_{11}, y_{21}, \ldots, y_{n1}, \ldots, y_{1m}, \ldots, y_{nm}]^\top$ is the vector of response variable for all observations and all tasks

- $\boldsymbol{x} = [x_{11}, x_{21}, \ldots, x_{n1}, \ldots, x_{1m}, \ldots, x_{nm}]^\top$ is the vector of independent variable for all observations and all tasks

- $h(\cdot)$ is a random function drawn from the global Gaussian process prior associated with all tasks, representing the underlying relationship between (x_{ij}) and (y_{ij}).

- $\mathcal{GP}(m(\cdot), k(\cdot,\cdot))$ is the Gaussian Process prior associated from which $h(\cdot)$. $m(\cdot)$ is the mean function associated with the Gaussian Process Prior and $k(\cdot,\cdot)$ is the covariance function associated with the Gaussian Process prior representing both the covariance between inputs and between tasks.

- $\boldsymbol{\epsilon}$ are the error terms associated with the observations which are identically and independently distributed following $\mathcal{N}(0, \sigma^2)$.

As discussed in earlier sections, the covariance function is central to specifying a Gaussian Process Regression (GPR) model. In the context of multi-task GPR (MTGPR), it becomes even more critical, as it must capture not only the correlations across input points but also the relationships between different tasks. In the following section, we will examine the structure of the covariance function for MTGPR, explore how it influences the overall model, review various approaches proposed in the literature, and finally introduce a novel method for learning the covariance function in a multi-task setting.

2.3.2 Task Correlation Approaches

The covariance structure is very important in the specification of the model when it comes to GPR. It is even more true in the case of MTGPR where one needs to model the correlation not only between inputs but also between tasks. The goal is that the correlation between enhance the prediction by incorporating relatedness of inputs points not part of the same task. For simplicity, we consider our N inputs for the different task to be part of the same input space therefore dimension for the inputs matrix $\boldsymbol{X}$ is $n \times p$ as already pointed out in previous sections. We define a covariance matrix for the MTGPR model as follows:

$$\boldsymbol{K_{t,x}} = \boldsymbol{K_t} \otimes \boldsymbol{K_x}$$

where:

- $\boldsymbol{K_{t,x}}$ is the covariance matrix for all points regardless of tasks. It has dimension $(m \times n) \times (m \times n)$.

- $\boldsymbol{K_t}$ is the covariance matrix for different M tasks. It has dimension $m \times m$.

- $\boldsymbol{K_x}$ is the covariance matrix for different N tasks. It has dimension $n \times n$.

The important property of the structure of the covariance matrix is that the Kronecker product allows to avoid a block diagonal matrix with respect to the tasks. This means that inputs that are not part of the same task are still able to affect the prediction for other task new observations. In the case tasks are not related, we have block diagonal matrix where inputs correlation are only affected by the inputs part of the same task [4]. The use of the Kronecker product allows for separability between the kernels which simplifies the model and the computations. It also introduces an assumption that the dependencies across tasks is independent

of the input space which allows for the decoupling of tasks and inputs multiplicatively [4], [7]. This is a strong assumption that works well when the inter-task correlations are homogeneous with respect to the input space. The separation allows you to choose different kernel functions for the task and input dimensions [18]. However, if the actual dependence structure is more complex, this formulation might be too restrictive for example in cases where inter-tasks dependency is also dependent on the inputs [7]. We will try to address that issue in following sections.

Previously, we introduced how covariance between input points can be determined using covariance functions, and we noted that misspecifying these functions can introduce errors into the model. However, choosing a suitable covariance function for multiple tasks is less straightforward, as numerous approaches have been proposed in the literature. In the following sections, we will review these existing methods and then propose our own custom approach for modeling inter-task correlations.

2.3.2.1 Covariance Functions in Multi-Task Gaussian Process Regression

Covariance functions in the context of Gaussian Process Regression are important to model the prior over functions. In the context of Multi-Task Gaussian Process Regression, multiple approaches to model the joint tasks and inputs covariance function have been proposed in the literature. It will be important to enumerate them in this section and explain their intricacies before proposing a different approach.

1. **The Linear Model of Coregionalization**:
   One is interested in modeling D different tasks each following respectively an output function $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_D(\boldsymbol{x})$ and each of these outputs are defined over the same input space $\boldsymbol{x} \in \mathbb{R}^p$. Instead of modelling each task independently, the model assumes that there is a set of Q latent functions $u_1(\boldsymbol{x}), u_2(\boldsymbol{x}), \ldots, u_q(\boldsymbol{x})$ that can be used to construct the output for the different tasks. Each output is then modeled as follows:

$$f_d(\boldsymbol{x}) = \sum_{q=1}^{Q} a_{d,q} u_q(\boldsymbol{x})$$

   The parameter $a_{d,q}$ is the weight that specifies how much of the latent function $u_q(\boldsymbol{x})$ contributes a particular function $f_d(\boldsymbol{x})$. In this approach, the way the output are correlated is via the shared underlying latent functions. Each output function depends on the same set of latent functions but to a varying degree which is specified via the weight. It is useful to note that the model is still a Gaussian Process because the set of latent fucntion $\{u_q(\boldsymbol{x})\}_{q=1}^{Q}$ are drawn from a Gaussian Process with wero mean and covariance function such that:

$$u_q(\cdot) \sim \mathcal{GP}(m^q(\cdot), k^q(\cdot,\cdot))$$
$$u_q(\cdot) \sim \mathcal{N}(0, k^q(\cdot,\cdot))$$

   A linear combination of a Gaussian Process still being a Gaussian Process the different tasks are still Gaussian Processes. The latent function are assumed to be drawn indepedently, therefore $cov[u_q(\boldsymbol{x}), u_{q'}(\boldsymbol{x'})] = k^q(\boldsymbol{x}, \boldsymbol{x'})$ only if $q = q'$ and zero otherwise. Despite the fact that the latent functions are modeled independently, the outputs are

still correlated as they share those latent functions. Sometimes, despite being drawn from independent Gaussian Processes, the latent functions can share the same covariance function. We consider Q different covariance functions that are shared by the latent functions and each covariance function is shared among M latent functions. One can then rewrite the output functions using the following sum instead:

$$f_d(\boldsymbol{x}) = \sum_{q=1}^{Q} \sum_{i=1}^{M} a_{d,q}^i u_q^i(\boldsymbol{x})$$

the set of functions $\left\{u_q^i(\boldsymbol{x})\right\}_{i=1}^M$ is the set of latent functions sharing the same covariance function $k^q(\cdot,\cdot)$. The cross covariance between two different output functions for different tasks d and d' is given as follows:

$$cov\left[f_d(\boldsymbol{x}), f_d'(\boldsymbol{x'})\right] = \sum_{q=1}^{Q}\sum_{q'=1}^{Q}\sum_{i=1}^{M}\sum_{i'=1}^{M} a_{d,q}^i a_{d',q'}^{i'} cov\left[u_q^i(\boldsymbol{x}), u_{q'}^{i'}(\boldsymbol{x'})\right]$$

Because the latent functions are independent from each other if $q \neq q'$ then the covariance between latent functions is 0. It leads to the cross covariance between ouput functions for different tasks d and d' collapse to the following sum:

$$\begin{aligned}
cov\left[f_d(\boldsymbol{x}), f_d'(\boldsymbol{x'})\right] &= \sum_{q=1}^{Q}\sum_{i=1}^{M} a_{d,q}^i a_{d',q}^i cov\left[u_q^i(\boldsymbol{x}), u_q^i(\boldsymbol{x'})\right] \\
cov\left[f_d(\boldsymbol{x}), f_d'(\boldsymbol{x'})\right] &= \sum_{q=1}^{Q}\sum_{i=1}^{M} a_{d,q}^i a_{d',q}^i k^q\left(\boldsymbol{x}, \boldsymbol{x'}\right) \\
cov\left[f_d(\boldsymbol{x}), f_d'(\boldsymbol{x'})\right] &= \sum_{q=1}^{Q} b_{d,d'}^q k^q\left(\boldsymbol{x}, \boldsymbol{x'}\right)
\end{aligned}$$

with $b_{d,d'}^q = \sum_{i=1}^M a_{d,q}^i a_{d',q}^i$. The weights represent the contribution of the latent function u_q^i to both outputs functions. If the weights are high then they have a strong relationship via that latent functions and enhances the correlation of the inputs via the multiplication of the weights and the kernel. Extended to the matrix notation where we have M tasks, Q groups for latent functions sharing the same kernel and N input points we get the following expression for the covariance matrix over all processes:

$$\boldsymbol{K} = \sum_{q=1}^{Q} \boldsymbol{A}_q \boldsymbol{A}_q^\top \otimes \boldsymbol{K}_q$$

$$\boldsymbol{K} = \sum_{q=1}^{Q} \boldsymbol{B_q} \otimes \boldsymbol{K}_q$$

where:

- $\boldsymbol{K}$ is the covariance matrix for all the processes combining the coregionalization matrix and the kernel matrix via their kronecker product. It has dimension $(m \times n) \times (m \times n)$.

- $\boldsymbol{B_q} = \boldsymbol{A}_q \boldsymbol{A}_q^\top$ is the coregionalization matrix of dimension $m \times m$
- $\boldsymbol{K}_q$ is the kernel matrix of inputs for latent processes sharing the same covariance functions $k^q(\cdot, \cdot)$. The kernel matrix has dimension $n \times n$

The covariance matrix for the complete model can be interpreted via the coregionalization matrix and the kernel matrix over inputs. In fact, the coregionalization matrix via the weights models how the output functions are correlated independent of the inputs and the kernel matrix models how the inputs are correlated independent of the outputs functions. It then leads to a covariance structure for our Gaussian Process Regression taking into account both the correlation between inputs and output functions. For more details on the derivation above see [6].

2. **Intrinsic Coregionalization Model**:
The Intrinsic Coregionalization Model (ICM) is a simplification of the Linear Model of Coregionalization (LMC) [6]. In fact the element of the coregionalization matrix can be rewritten to give the following expression:

$$cov\left[f_d(\boldsymbol{x}), f'_d(\boldsymbol{x'})\right] = \sum_{q=1}^{Q} b^q_{d,d'} k^q\left(\boldsymbol{x}, \boldsymbol{x'}\right)$$

$$cov\left[f_d(\boldsymbol{x}), f'_d(\boldsymbol{x'})\right] = \sum_{q=1}^{Q} \upsilon_{d,d'} b^q k^q\left(\boldsymbol{x}, \boldsymbol{x'}\right)$$

$$cov\left[f_d(\boldsymbol{x}), f'_d(\boldsymbol{x'})\right] = \upsilon_{d,d'} \sum_{q=1}^{Q} b^q k^q\left(\boldsymbol{x}, \boldsymbol{x'}\right)$$

where $\upsilon_{d,d'}$ represent the weight of coregionalization b^q between two outputs functions. For M different tasks if extended to a matrix notation, the complete covariance matrix for the model becomes:

$$\boldsymbol{K} = \boldsymbol{\Upsilon} \otimes \boldsymbol{K_x}$$

where:

- $\boldsymbol{K}$ is the covariance matrix for all the processes combining the coregionalization weights and the kernel matrix via their kronecker product. It has dimension $(m \times n) \times (m \times n)$.
- $\boldsymbol{\Upsilon}$ is the coregionalization weights associated with each tasks pairs. It is of dimension $m \times m$
- $\boldsymbol{K}_x = \sum_{q=1}^{Q} b^q \boldsymbol{K_q}$ is the kernel matrix for all inputs. b^q represent the coregionalization associated with that particular kernel group and it is weighted by $\upsilon_{d,d'}$ for each output function pair. It is of dimension $n \times n$.

Intuitively, each kernel group has some degree of coregionalization determined by b^q and which is weighted via $\upsilon_{d,d'}$ for each output function pair making $\boldsymbol{\Upsilon}$ the correlation

across output functions and $\boldsymbol{K_x}$ the correlation across inputs where both are determined independently of each other.

3. **Deep Kernel Learning**:
The covariance matrix in Gaussian Process Regression is highly important and the relationship between two inputs vector $\boldsymbol{x}$ and $\boldsymbol{x'}$ is important in modeling it. Often times though there exists some complex relationship between input points that can not directly be modeled with common covariance functions. Deep Kernel Learning (DKL) was introduced in Wilson, Hu, Salakhutdinov, *et al.* [9] to improve the expressive power of covariance function by learning a non-linear mapping of inputs using deep neural network architecture. The model is formulated as follows:

$$k(x, x') = k(g(x), g(x'))$$

where $g(\cdot)$ is the mapping of the inputs learned from a deep neural network and $k(\cdot, \cdot)$ is the chosen covariance function. One interpretation of DKL is that we perform a Gaussian Process Regression with the covariance function $k(\cdot, \cdot)$ where the inputs to the covariance are the output of the last hidden layer of the neural network. It needs to be noted, however, that the hyperparameters of the chosen covariance function and the deep neural network weights need to be learned together. In the case of multi task Gaussian Process Regression, the learned covariance function can then be used with a coregionalization matrix that model the correlation between tasks as seen above.

4. **Task Embedding**:
In the context of multi-task Gaussian Process Regression, determining the correlation between task can difficult. The reason for it being that it is difficult to clearly model the similarity between different task. One approach that has been proposed in Hutter, Stosch, Cruz Bournazou, *et al.* [14], however, is the use of embedding vectors for different tasks borrowed from the concept of (word) embedding vectors used natural language understanding. Traditional approaches to model multi-task to handle the product identity for each data point as a categorical variable and for analysis purposes transformed into a one hot (dummy) variable. In the context of Multi-Task Gaussian Process Regression this approach would be problem. Let consider two inputs points $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ both coming from the same process, $\boldsymbol{x_3}$ and $\boldsymbol{x_4}$ coming from similar processes while not exactly the same and $\boldsymbol{x_5}$ and $\boldsymbol{x_6}$ coming from two totally different processes. Let also for simplicity consider and RBF covariance function with $\ell = 1$ for the continuous features and and RBF covariance function with $\ell = 1$ for the categorical features. We then have the following:

$$\begin{cases} k(\boldsymbol{x_1}, \boldsymbol{x_2}) = k_{rbf}(\boldsymbol{x_1}, \boldsymbol{x_2}) \cdot \exp\left(-\frac{\|t-t'\|^2}{2\ell^2}\right), \\ k(\boldsymbol{x_3}, \boldsymbol{x_4}) = k_{rbf}(\boldsymbol{x_3}, \boldsymbol{x_4}) \cdot \exp\left(-\frac{\|t-t'\|^2}{2\ell^2}\right), \\ k(\boldsymbol{x_5}, \boldsymbol{x_6}) = k_{rbf}(\boldsymbol{x_5}, \boldsymbol{x_6}) \cdot \exp\left(-\frac{\|t-t'\|^2}{2\ell^2}\right) \end{cases}$$

The implication of such approach is that for the same task the value of inputs covariance function will be multiplied by 1, but for task that are different even when similar it will

be multiplied by something around 0.36. This approach is then unable to make use of intricate similarities between task. To improve this, it was proposed to use embedding vectors where two tasks are considered similar (high kernel value) if the two associated points in embedding space are close to each other as measured by euclidean distance. The embedding vectors are learned using the following procedures. We define a covarince function such that:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = k_{rbf}(\boldsymbol{x}, \boldsymbol{x'}) \cdot \exp\left(-\frac{1}{2}\left\|We - We'\right\|^2\right)$$

where e and e are one hot representations of the tasks which are appended to the training data. The embedding are then obtained via the optimization the covariance function hyperparameters done through a Gaussian Process Regression:

$$\theta, W = \arg\max_{\theta', W'} \log P\big(Y \mid X, \theta', W'\big)$$

Once learned the embeddings can be used together with the inputs kernel matrix to build the joint covariance function for the multi-task Gaussian Process Regression model.

2.3.2.2 Proposed Model: Deep Kernel Learning for Features and Task Embedding

In this section, we define the model of interest. The model is built upon the above work. The work cited above provided various approaches to model covariance functions for multi-task Gaussian Process Regression. We recall the model one is interested in as follows:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y_1} = h_1(\boldsymbol{x_1}) + \boldsymbol{\epsilon_1} \\ \vdots \\ \boldsymbol{y_m} = h_j(\boldsymbol{x_m}) + \boldsymbol{\epsilon_m}, \end{bmatrix} = \begin{cases} h_1(\cdot) \sim \mathcal{GP}\big(0, k^1(\cdot,\cdot)\big), \\ \quad \vdots \\ h_m(\cdot) \sim \mathcal{GP}\big(0, k^m(\cdot,\cdot))\big), \end{cases}$$

The complete model fro the Multi-Task Gaussian Process Regression is then defined as:

$$\boldsymbol{y} = h(\boldsymbol{x}) + \boldsymbol{\epsilon}$$

with

$$h(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$$

The goal is to model a covariance structure that take into account the correlation between input and task. We propose an approach where we use the concepts from DKL [9], Task embedding [14] and Kronecker algebra [7] seen above. It is important to give the reader a good understanding of the model, to do so we start by defining an autoencoder neural network. An autoencoder is a type of neural network that aims to learn the most informative representation of a dataset in an unsupervised manner, it was first introduced in **hinton2006reducing**. We define $\boldsymbol{x} \in \mathbb{R}^D$ where $D \in \mathbb{N}$. We define a function $h(\cdot) : \mathbb{R}^D \to \mathbb{R}^P$, the function $h(\cdot)$ takes an input of D dimensions to a latent space of P dimensions where $P \in \mathbb{N}$. Finally, we

define a function $f(\cdot) : \mathbb{R}^P \to \mathbb{R}^D$ which takes an input of P dimensions and maps it back to the original D dimensional space. An autoencoder is then a neural network with "two" components, an encoder component such that:

$$\boldsymbol{l}_i = h(\boldsymbol{x}_i)$$

where $\boldsymbol{l}_i$ is the latent representation of $\boldsymbol{x}_i$ after being passed through the encoder. The autoencoder has a decoder component such that:

$$\hat{\boldsymbol{x}}_i = f(\boldsymbol{l}_i) = f(h(\boldsymbol{x}_i))$$

where $\hat{\boldsymbol{x}}_i$ is the reconstructed version of $\boldsymbol{x}_i$ after being passed through the decoder. The autoencoder is then trained trying to minimize the difference between the original point and its reconstruction as follows:

$$\arg\min_{h,f} E\left[\Delta\left(\boldsymbol{x}_i, f(h(\boldsymbol{x}_i))\right)\right]$$

In our model because we are only insterested in the latent space representation only the encoder will really be of insterest. Instead of using the raw input features $\boldsymbol{x}$, we first apply an auto-encoder **hinton2006reducing** neural network to learn a nonlinear transformation of the input data. It will be important, however, to train both the encoder and the decoder together to learn the best representation possible. The resulting features from the encoder component $\boldsymbol{l}$ replace $\boldsymbol{x}$ in the subsequent modeling steps. The adavantages of using an autoencoder is the added flexibility it gives to the model in handling the isotopic as well as the heterotopic case in Multi-Task Gaussian Process Regression as pointed in Section 2.3.1.1. In fact, when the tasks share the same input domain, the autoencoder would just encode the input in a latent space whose dimension P is arbitrarly chosen. Now, consider a model with $T \geq 2$ tasks. Each task have a different input domain with varying dimensionality D such that $\left\{D^t\right\}_{t=1}^{T}$. This is an heterotopic case. In such case where the tasks come from different input domains the autoencoder will encode the data in a latent dimensional space P where $P \leq min\left\{D^t\right\}$ meaning the latent dimensional space where all the inputs are encoded will be lower or equal to the input domain having the smallest dimension. Consequently, this unified latent representation resolves domain-misalignment issues and places every tasks data within the same effective input domain. Simultaneously, the proposed model introduces a separate neural network to learn tasks embeddings. Let consider $\boldsymbol{x}_i \in \mathbb{R}^{N+1}$ where $N \in \mathbb{N}$ meaning that the point has N continuous dimensions and 1 categorical dimension. One is trying to find embeddings for the categorical variable value has some machine learning model tend to learn better when inputs are continuous. We define a function $g(\cdot) : \mathbb{R}^1 \to \mathbb{R}^Q$ where $Q \in \mathbb{R}$, meaning the function $g(\cdot)$ takes a input in a categorical space one dimensional then maps it to Q dimensional space where Q is a hyperparameter that is pre-determined by the modeler, the model is defined as follows:

$$\boldsymbol{e}_i = g(x_i)$$

It is showed in Guo and Berkhahn [10] showed that learning the embeddings was equivalent to learning the weights of embedding layer, the model proposed here is different has one wants to utilize the continuous variables to better learn the embeddings. Let consider $\boldsymbol{x}_i \in \mathbb{R}^{N+1}$ where $N \in \mathbb{N}$ meaning that the point has N continuous dimensions and 1 categorical

dimension, categories are defined as number $1, 1, \ldots, m$ where m is the number of categories. First a trainable lookup table is created by using an embedding layer defined as $g(\cdot)$ such that:

$$g(\cdot) : x_i \rightarrow \boldsymbol{e}_i$$

where:

- $g(\cdot)$ is the function in the embedding layer such that $g(\cdot) : \mathbb{R}^1 \rightarrow \mathbb{R}^Q$ where $Q \in \mathbb{R}$
- x_i is the value of the categorical variable with m category in the categroical dimension for the i-th observation.
- $\boldsymbol{e}_i$ is the vector of embedding vector for encoded category for the i-th observation.

Guo and Berkhahn [10] show how to create a trainable lookup table using an embedding layer. Let define α an index having the same possible values as the possibilities of x_i such that:

$$\alpha \in \{1, 2, \ldots, m\}$$

We define a function $u(\cdot)$ taking an input, it outputs the Kronecker delta of that input and α the index value defined above such that:

$$u(\cdot) : x_i \rightarrow \delta_{x_i,\alpha}$$

where $\delta_{x_i,\alpha}$ is a vector of dimension m with entry 1 where the index α has the same value as the input x_i and 0 otherwise, meaning if for example the value of $x_i = 1$ then the value of the vector at index 1 will be 1. The embeddings are then initialized using an embedding layer, a lookup table is then created for the different categories to enable the embedding. We define $\boldsymbol{W} \in \mathbb{R}^{m \times Q}$ where $Q \in \mathbb{R}$ is the number of dimensions chosen for the embeddings. The matrix $\boldsymbol{W}$ is then defined as follows:

$$\boldsymbol{W} = [w_{\alpha\beta}]_{\substack{\alpha=1,\ldots,m \\ \beta=1,\ldots,Q}}$$

The entries of $\boldsymbol{W}$ the lookup table (and also the weight matrix of the embedding layer) are calculated as follows:

$$x_i \equiv \sum_{\alpha=1}^{m} w_{\alpha\beta}\delta_{x_i\alpha} = w_{x_i\beta}$$

where $w_{x_i\beta}$ is the entry in the lookup table where the row is equal to encoding number for x_i and the column is the value of β. It needs to be noted that the weights are the weights connecting the one-hot-encoding layer and the embedding layer. The results of the embedding is then given as follows:

$$\boldsymbol{e}_i \equiv [w_{x_i1}, w_{x_i2}, \ldots, w_{x_iQ}]^\top$$

meaning that $\forall x_i$ having the same value of the categorical one selects the x_i-th row from the lookup table which also the weight matrix between the one-hot-encoded layer and the

embedding layer. To find the best embeddings, one then needs to train the model so that the embeddings can capture how well categories relates to each other. It is where the approach proposed here fundamentally departs from the one proposed in Guo and Berkhahn [10] while they initialize the embeddings, concatenate them to the continuous variablestreat them like a normal input layer in neural networks and other layers can be build on top of it and continuously learn them via backpropagation, we proposed a different approach. After the embeddings are ~~then~~ initialized using an embedding layer because, we want to utilize the continuous variables to learn the embeddings an autoencoder is also used here. The lookup table values replace the categorical variable encoding such that $\boldsymbol{x}_i \in \mathbb{R}^{N+Q}$ where $N \in \mathbb{N}$ is the number of initial continuous variables and $Q \in \mathbb{R}$ is the number of dimensions chosen for the embeddings. We build an autoencoder as previously defined above such that we have a function $h(\cdot)$ takes an input of $N+Q$ dimensions to a latent space of P dimensions where $P \in \mathbb{N}$. Finally, we define a function $f(\cdot) : \mathbb{R}^P \to \mathbb{R}^{N+Q}$ which takes an input of P dimensions and maps it back to the original $N+Q$ dimensional space. Here again, the autoencoder has two "components", an encoder component such that:

$$\boldsymbol{l}_i = h(\boldsymbol{x}_i)$$

where $\boldsymbol{l}_i$ is the latent representation of $\boldsymbol{x}_i$ after being passed through the encoder. The autoencoder has a decoder component such that:

$$\hat{\boldsymbol{x}}_i = f(\boldsymbol{l}_i) = f(h(\boldsymbol{x}_i))$$

The key principle here is that the autoencoder is trained to minimize the following loss function:

$$\mathcal{L} = \underbrace{\text{MSE}\big(\hat{x}_{\text{cont}},\, x_{\text{cont}}\big)}_{\text{continuous reconstruction}} + \underbrace{\text{CE}\big(\text{logits},\, x_{\text{cat}}\big)}_{\text{category reconstruction}}$$

The MSE loss pushes network to reconstruct the continuous features accurately by measuring how close the reconstructed continuous outputs are to the true continuous inputs while the CE pushes the network to reconstruct the encoded category by measuring how well the reconstructed logits recover the true category index. The training is performed so that at each iteration the embeddings is shaped so that when concatenated to the continuous features they help the network minimize its joint reconstruction loss. After training, the reconstructed outputs are discarded. The sole output is the embedding matrix which captures how the model "positioned" each category in $\mathbb{R}^Q$ to best explain both its continuous data distribution and its identity. Our approach fundamentally departs from theirs in two ways: in our appraoch the embeddings are learned in a self-supervised manner as not external target variable are required to be predicted. The second difference is that the joint reconstruction loss forces the category embeddings to capture the structure inherent in the joint distribution between the continuous variables and the categorical variables. Using this model, one can represent the tasks as categories in a data matrix $\boldsymbol{X}$ and learn the embedding, these learned embeddings can be used cosine similarity-based covariance function, capturing the relationships among tasks. It needs to be noted that because the model makes use of an autoencoder again both the isotopic and heterotopic case can be solved. The learned non-linear transformation covariance matrix and the learned embeddings covariance matrix will be combined via their Kronecker Product to obtain a joint covariance matrix as follows:

$$\boldsymbol{K} = \boldsymbol{K}_{\text{Cosine Similarity}} \otimes \boldsymbol{K}_{\boldsymbol{x}}$$

Training deep autoencoder can be difficult due to the problem of vanishing gradient that usually arises in deep neural networks. This phenomenon occurs when gradients become increasingly small as they are backpropagated through the layers, leading to negligible weight updates in the earlier layers of the network. As a result, the network learns very slowly or may even stop learning altogether. A related issue is known as the sleepy neuron problem (often referred to as the "dying ReLU" problem). In networks that use the standard ReLU activation function, neurons output zero for any negative input:

$$ReLU(x) = max(0, x)$$

Because the gradient of ReLU is zero for $x < 0$, these neurons might receive no update during backpropagation if they fall into the negative regime, effectively becoming inactive. A way to solve this issue when training the model would be to use LeakyReLU an improved version of the ReLU (Rectified Linear Unit) function given by:

$$LeakyReLU(x) = max(\alpha x, x)$$

This modification helps mitigate both the vanishing gradient and sleepy neuron problems by allowing a small gradient even for negative input unlike the standard ReLU activation, promoting healthier gradient flow throughout the deep network and ultimately facilitating more effective training. Let consider the data matrix $\boldsymbol{X}$ which has dimension $n \times p$. The data matrix contains the observations for each task. The approach proposed above take advantages of ideas presented in [9], [14] but differs from them by using autoencoder where learning the representation is done in an unsupervised manner. The model differs from the work proposed in [14] where one had to have a prior belief in terms of kernel chosen for learning the embeddings. This approach is similar to the task embedding using Gaussian Process Regression but differs by not imposing a prior structure in the dataset through the covariance function. The resulting embeddings will be fed into a cosine similarity covariance function to create a covariance matrix to be combined with the covariance matrix created from the autoencoder's learned features from the autoencoder network. The complete covariance matrix would then encapsulates both the task correlation via the embeddings, the input correlation via the learned features and will be used as the covariance matrix in Gaussian Process Regression. ~~After outlining the proposed model, in the next section we will perform a simulation study using Monte Carlo simulations to compare the proposed model empirically with previously proposed approaches~~.

Chapter 3

Multi Task Gaussian Process Regression Simulation Study

The previous section highlighted the Gaussian Process Regression literature. It reviewed previous approach to model Multi-Task Gaussian Process Regression and our new proposed model. The goal of this chapter is to perform simulation studies using Monte Carlo simulations to compare empirically the behavior of the new proposed model compared to previous approaches. To reflect both accuracy and uncertainty quantification, hallmarks of Gaussian Process Regression, we evaluate each model on the mean squared error (MSE) of point predictions and the width of the predictive confidence intervals. Evaluating these metrics will assess which model makes the most accurate predictions but also provides the most reliable measure of uncertainty.

3.1 Simulation Data Setup

Let us define the experimental setting of the simulation studies. We define three experimental tasks as follows

$$\begin{aligned} Y_t^{(1)} = &-0.4\, Y_{t-1}^{(1)} + 0.5\, X_{1,t-1} + 0.1\, X_{2,t-1} - 0.2\, X_{3,t-1} + 0.2\, X_{4,t-1} + 1.2\, X_{5,t-1} + 0.43\, X_{6,t-1} \\ &+ 0.4\, X_{7,t-1} + 1.2\, X_{8,t-1} - 0.6\, X_{9,t-1} - 0.4\, X_{10,t-1} + \varepsilon_{1,t} \end{aligned}$$

$$\begin{aligned} Y_t^{(2)} = &\ 0.1\, Y_{t-1}^{(2)} + 3.7\, X_{1,t-1} + 5.1\, X_{2,t-1} - 7.2\, X_{3,t-1} + 0.8\, X_{4,t-1} + 1.2\, X_{5,t-1} + 3.3\, X_{6,t-1} \\ &+ 0.5\, X_{7,t-1} + 1.2\, X_{8,t-1} - 0.6\, X_{9,t-1} - 3.9\, X_{10,t-1} + \varepsilon_{2,t} \end{aligned}$$

$$\begin{aligned} Y_t^{(3)} = &\ 0.7\, Y_{t-1}^{(3)} + 1.7\, X_{1,t-1} + 0.1\, X_{2,t-1} - 6.2\, X_{3,t-1} \\ &+ 3.4\, X_{4,t-1} + 2.2\, X_{5,t-1} + 5.3\, X_{6,t-1} \\ &+ 1.5\, X_{7,t-1} + 3.2\, X_{8,t-1} - 0.6\, X_{9,t-1} - 1.9\, X_{10,t-1} + \varepsilon_{3,t} \end{aligned}$$

The three experimental tasks are linear combinations of regressors and coefficients such that:

$$\text{For each series we have } j \text{ variables such that } j = 1, \ldots, 10$$

$$X_{j,t} = \begin{cases} X_{j,t-1} + \gamma_j + \eta_{j,t}, & \text{when } j \text{ is odd } (1,3,5,7,9) \\ \mu_j + \eta_{j,t}, & \text{when } j \text{ is even } (2,4,6,8,10) \end{cases}$$

where:

- γ_j is small drift (or trend) with a distribution $\gamma_j \sim \mathcal{U}(0.001, 0.1)$.
- μ_j is the random baseline drawn once at $t = 0$
- $\eta_{j,t}$ is the random noise added to the term with a distribution $\eta_{j,t} \sim \mathcal{N}(0, \sigma_j^2)$.

It is important to note that the odd and even variables have a baseline, however the odd variables drift from their baseline while the even variables are "stable" and only have random noise added to them. The three experimental tasks are AR-X responses whose equations are given as follows:

$$Y_t^{(k)} = \rho_k Y_{t-1}^{(k)} + \sum_{j=1}^{10} \beta_{k,j} X_{j,t-1} + \varepsilon_{k,t},$$
$$\varepsilon_{k,t} \sim \mathcal{N}(0, \sigma_k^2), k = 1, 2, 3$$

The different variables, in the experimental setup, have a baseline drawn from a particular probability density function and from this baseline depending if the variable is odd or even either a drift is added or only noise is added as described above. The different distributions for each variable baseline are as follows:

- $X_{1,0} \sim \mathcal{U}(6, 7)$
- $X_{2,0} \sim \text{Poisson}(\lambda = 9)$
- $X_{3,0} \sim \Gamma\big(k = 9,\ \theta = 3\big)$
- $X_{4,0} \sim \mathcal{U}(30, 31)$
- $X_{5,0} \sim \mathcal{U}(3, 4)$
- $X_{6,0} \sim \mathcal{N}(1,\ 1^2)$
- $X_{7,0} \sim \mathcal{U}(6, 7)$
- $X_{8,0} \sim \text{Beta}(\alpha = 6,\ \beta = 7)$
- $X_{9,0} \sim \mathcal{N}(6,\ 1^2)$
- $X_{10,0} \sim \mathcal{U}(20, 24)$

The experimental procedure is a Monte Carlo simulation. To compare different models, the procedureis M times and the sample size is T. The parameters of the simulation study is then as follows:

- Number of simulations: $M = 50$

- Sample size: $T = 50, 100, 500$

The samples are generated after 200 burn-in realisations to ensure that the process is stationary. This is because when simulating a dynamic process you have to start it somewhere, usually with arbitrary or highly simplified initial values. Those early values are not drawn from the model's long-run (stationary) distribution, so the first few iterations contain transient behaviour that differs from the behaviour one might care about. Below, the graphs for each experimental task $Y^(k)$ for $T = 100$ to show the general trend of our dependent variables over time.

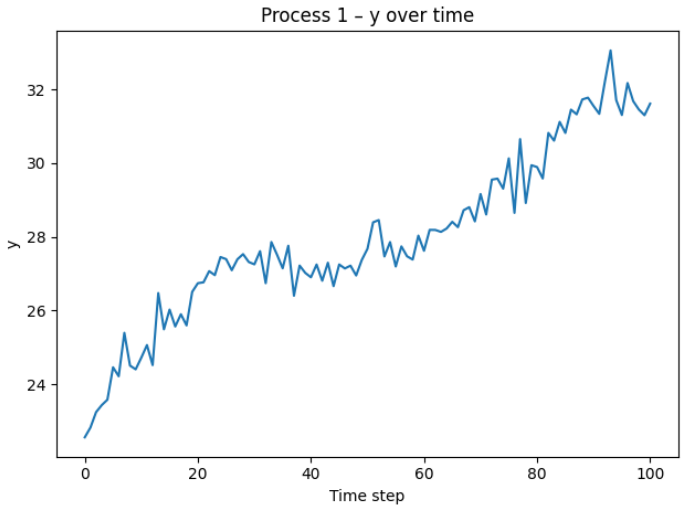

Figure 3.1.1: Graph of Y variable over time for Experimental Task 1

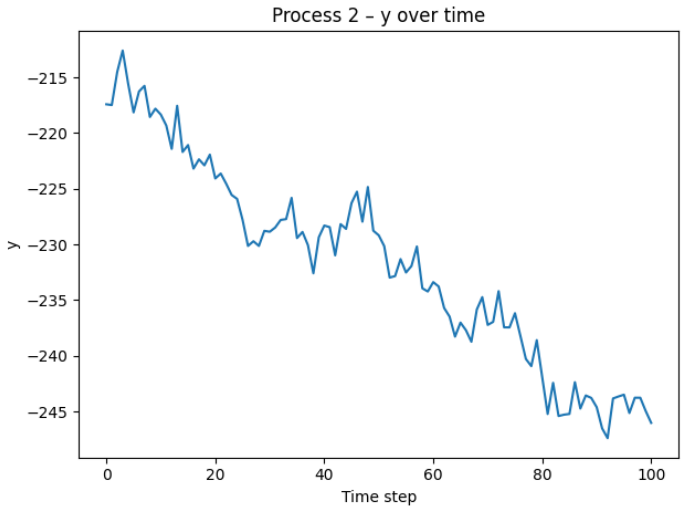

Figure 3.1.2: Graph of Y variable over time for Experimental Task 2

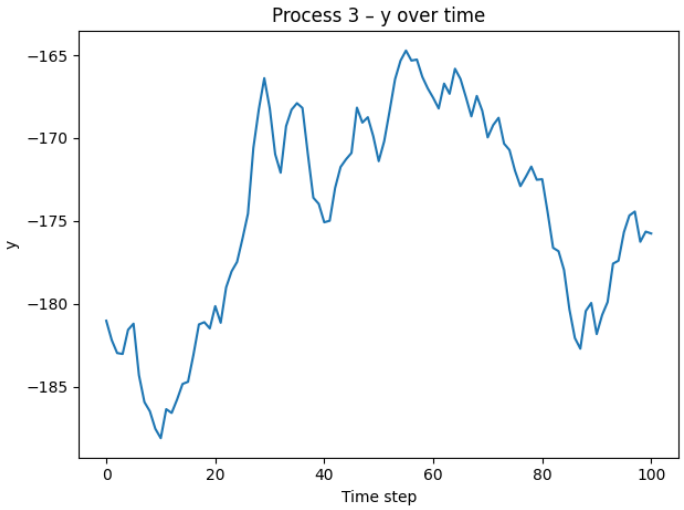

Figure 3.1.3: Graph of Y variable over time for Experimental Task 3

When comparing the performances of different models on a particular process, we run the procedure M times where for each iteration, we simulate a sample size. It needs to be noted as already pointed previously that for simplicity the dataset for our experimental studies is isotopic in that the inputs are the same for every output and they differ only in the coefficient or the transformation applied to the inputs. We start by generating N sample points for each X variable where we increase the sample size to see the impact of a large dataset on the model. Each model is fitted on the first 80% of the sample and tested on the remaining 20% of the sample. The predictions made on the test set are then used to quantify the perfomance metrics specified above for each model. Let T^* be the size of the test set, the performance metrics for the various models are given by:

$$MSE = \frac{1}{M}\left(\frac{1}{T^*}\sum_{t^*=1}^{T^*}\left(Y_t - \hat{Y}_t\right)^2\right)$$

where $\hat{Y}_t$ is the prediction made by the model of interest for the value of Y at time t. The second metric that will be used is the width of the predictive confidence intervals given by:

$$\text{Width} \;=\; \frac{1}{M}\left(\frac{1}{T^*}\sum_{t^*=1}^{T^*}\left(U_{t^*} - L_{t^*}\right)\right)$$

where:

- $U_{t^*} = \mu_{t^*} + z_{1-\alpha/2}\,\sigma_{t^*}$ is the upper bound of the predictive confidence interval for a particular test point
- $L_{n^*} = \mu_{t^*} - z_{1-\alpha/2}\,\sigma_{t^*}$ is the lower bound of the predictive confidence interval for a particular test point

It needs to be noted however that despite the fact all experimental tasks have variables coming from the same distributions since the input variables for each task are drawn independently

make the current experimental setting a heterotopic design where the model needs to be estimated over an heteregenous input domain. The proposed model has the ability to learn the correlation between task that have different inputs domain through the embedding of the task in a latent space. The correlation between the task is then learned through their cosine similarity in that space. Regarding the input domain the mapping of independent variables to a latent space via an Autoencoder neural network gives some flexibility as outlined in Section 2.3.2.2.

3.2 Empirical Results

3.2.0.1 title

3.3 Comparision of Methods

3.4 Convergence and Stability Analysis

3.5 Limitations

Chapter 4

Multi Output/Multi Task Gaussian Process Regression Data Analysis

Chapter 5

Federated Learning Extension

5.1 Federated Learning Framework

5.2 Federated Multi Output Gaussian Process Regression

5.2.1 Task Correlation in Federated Learning Framework

5.2.2 Model Averaging Strategies

5.2.3 Theoretical Derivations

5.3 Simulation Study

5.3.1 Simulation Data Setup

5.3.2 Theoretical Results

5.3.3 Comparision of Methods

5.3.4 Convergence and Stability Analysis

5.3.5 Limitations

5.4 Federated Learning Data Analysis

5.4.1 Data source and explanation

5.4.2 Pre-processing

5.4.3 Implementation

5.4.4 Comparison to simulation study

Chapter 6

Discussion

Chapter 7

Conclusion

Bibliography

[1] C. Williams and C. Rasmussen, "Gaussian processes for regression", in *Advances in Neural Information Processing Systems*, vol. 8, MIT Press, 1995. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1995/hash/7cce53cf90577442771720a370c3c723-Abstract.html (visited on 10/17/2024).

[2] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning* (Adaptive computation and machine learning). Cambridge, Mass: MIT Press, 2006, 248 pp., OCLC: ocm61285753, ISBN: 978-0-262-18253-9.

[3] E. V. Bonilla, F. V. Agakov, and C. K. I. Williams, "Kernel multi-task learning using task-specific features", in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, ISSN: 1938-7228, PMLR, Mar. 11, 2007, pp. 43–50. [Online]. Available: https://proceedings.mlr.press/v2/bonilla07a.html (visited on 03/19/2025).

[4] E. V. Bonilla, K. Chai, and C. Williams, "Multi-task gaussian process prediction", in *Advances in Neural Information Processing Systems*, vol. 20, Curran Associates, Inc., 2007. [Online]. Available: https://papers.nips.cc/paper_files/paper/2007/hash/66368270ffd51418ec58bd793f2d9b1b-Abstract.html (visited on 10/17/2024).

[5] C. P. Robert, Ed., *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, Springer Texts in Statistics, New York, NY: Springer New York, 2007, ISBN: 978-0-387-71598-8 978-0-387-71599-5. DOI: 10.1007/0-387-71599-1.

[6] M. A. Alvarez and N. D. Lawrence, "Computationally efficient convolved multiple output gaussian processes", *The Journal of Machine Learning Research*, vol. 12, pp. 1459–1500, 2011.

[7] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, *Kernels for vector-valued functions: A review*, Apr. 16, 2012. DOI: 10.48550/arXiv.1106.6251. arXiv: 1106.6251[stat]. [Online]. Available: http://arxiv.org/abs/1106.6251 (visited on 05/03/2025).

[8] W. K. HÃ€rdle and L. Simar, *Applied Multivariate Statistical Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, ISBN: 978-3-662-45170-0 978-3-662-45171-7. DOI: 10.1007/978-3-662-45171-7. [Online]. Available: https://link.springer.com/10.1007/978-3-662-45171-7 (visited on 05/03/2025).

[9] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, *Deep kernel learning*, Nov. 6, 2015. DOI: 10.48550/arXiv.1511.02222. arXiv: 1511.02222[cs]. [Online]. Available: http://arxiv.org/abs/1511.02222 (visited on 02/19/2025).

[10] C. Guo and F. Berkhahn, *Entity embeddings of categorical variables*, Apr. 22, 2016. DOI: 10.48550/arXiv.1604.06737. arXiv: 1604.06737[cs]. [Online]. Available: http://arxiv.org/abs/1604.06737 (visited on 05/02/2025).

[11] T. Beckers, J. Umlauft, and S. Hirche, *Mean square prediction error of misspecified gaussian process models*, Nov. 16, 2018. arXiv: 1811.06642. [Online]. Available: http://arxiv.org/abs/1811.06642 (visited on 11/26/2024).

[12] B. Wundervald, *Bayesian Linear Regression*. Jun. 21, 2019. DOI: 10.13140/RG.2.2.28385.97121.

[13] T. Beckers, *An introduction to gaussian process models*, Feb. 10, 2021. arXiv: 2102.05497[cs,eess]. [Online]. Available: http://arxiv.org/abs/2102.05497 (visited on 10/08/2024).

[14] C. Hutter, M. von Stosch, M. N. Cruz Bournazou, and A. ButtÃ©, "Knowledge transfer across cell lines using hybrid gaussian process models with entity embedding vectors", *Biotechnology and Bioengineering*, vol. 118, no. 11, pp. 4389–4401, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bit.27907, ISSN: 1097-0290. DOI: 10.1002/bit.27907. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bit.27907 (visited on 04/15/2025).

[15] H. Liu, K. Wu, Y.-S. Ong, C. Bian, X. Jiang, and X. Wang, *Learning multi-task gaussian process over heterogeneous input domains*, Jun. 18, 2022. DOI: 10.48550/arXiv.2202.12636. arXiv: 2202.12636[stat]. [Online]. Available: http://arxiv.org/abs/2202.12636 (visited on 05/02/2025).

[16] J. Wang, "An intuitive tutorial to gaussian process regression", *Computing in Science & Engineering*, vol. 25, no. 4, pp. 4–11, Jul. 2023, ISSN: 1521-9615, 1558-366X. DOI: 10.1109/MCSE.2023.3342149. arXiv: 2009.10862[cs,stat]. [Online]. Available: http://arxiv.org/abs/2009.10862 (visited on 10/08/2024).

[17] D. Ye and M. Guo, *Bayesian approach to gaussian process regression with uncertain inputs*, May 28, 2023. arXiv: 2305.11586. [Online]. Available: http://arxiv.org/abs/2305.11586 (visited on 10/17/2024).

[18] N Bu, "School of informatics, university of edinburgh",

[19] D. K. Duvenaud, "Automatic model construction with gaussian processes",

[20] T. Evgeniou, T. Evgeniou, C. A. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods",

[21] Y. W. Teh, M. Seeger, and M. I. Jordan, "Semiparametric latent factor models",