Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Study of Multi-Output Gaussian Processes On Heterogeneous Medical Data

Bachelor's Thesis

Solmaz Mohammadi

Supervisor:

Dr. Mohammadreza Faraji

February 2021

Abstract

Gaussian Processes (GPs) are a popular non-parametric covariance approximation method that provides a Bayesian approach to smoothing and interpolation. The extension of this method, Multi-Output Gaussian Processes (MOGPs), has shown significant progress in expressiveness and interpretation of the relatedness between different models and exposing underlying models in heterogeneous data. We review the fundamentals of Gaussian processes and mixture kernels. Furthermore, we represent a method for applying MOGPs to a series of heterogenous medical records that have achieved noticeable results. This method uses the combination of the squared exponential and the periodic kernels to build a multi-output time-to-event model for each patient. Moreover, it constructs a population-level model to estimate new patients.

Table of Contents

1. Background

"Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed."

- Arthur Samuel

1.1. Introduction

Machine learning is a branch of the Artificial Intelligence field that focuses on building applications that learn from data and improve their accuracy without being programmed. In machine learning, algorithms are trained to find patterns and features in a large amount of data and make predictions for new data [1].

Over the decades of revolutionary changes in data processing and machine learning, the process of model selection has become more critical. The exact definition of a good training model has become very dependent on different data elements that need to be trained. However, there are some methods and approaches for reaching the maximum accuracy using the training model. Machine learning methods are mainly divided into three major categories: supervised, unsupervised, and reinforcement learning.

The features of data may be continuous, categorical, or binary. If features are given with known labels (the corresponding correct outputs), then the learning is called supervised, in contrast to unsupervised learning, where instances are unlabeled. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown but practical classes of items. Another kind of machine learning is reinforcement learning. The training information provided to the learning system by the environment is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates [2].

1.2. Supervised Learning

Supervised machine learning is the search for algorithms that reason externally supplied instances to produce general hypotheses, making predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown [3]. (Figure 1.1)

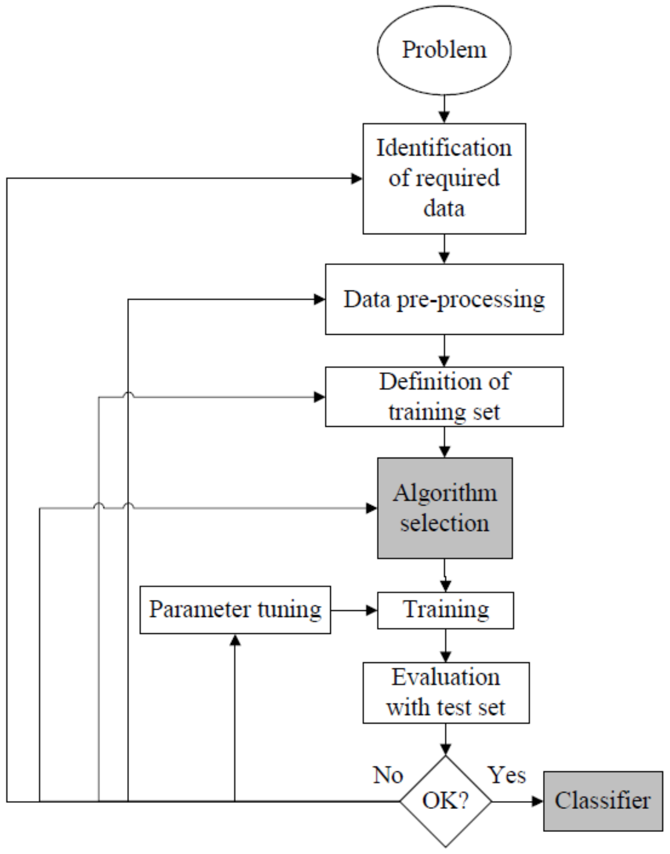


***Figure 1.1**. The process of supervised ML [3]*

The primary purpose of supervised learning techniques is to learn how to predict a random variable Y∈Y based on a set of explicative random variables denoted by X ∈ X, where Y and X depend on the problem at hand [2]. We will often call X the input and Y the output. The main problem is to find a predictor of output y based on the given input x:

$$f: \chi \rightarrow \gamma, \qquad (1.1)$$

To build a good predictor, we have to define a performance criterion that we call the loss function denoted L, which depends on f, X, and Y. The most common loss function is the mean squared error (MSE) [4]:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{1.2}$$

The mean square error is measured as the average squared difference between predictions and actual observations. It is only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions that are far away from actual values are penalized heavily compared to less deviated predictions [4]. We thus say that predictor *f* is better than predictor *g* if $L(f, X, Y) < L(g, X, Y)$. Suppose that there exists a unique predictor $f^* \in F$ which minimizes the loss function *L*, called the optimal predictor,

$$f^* = argmin\ L(f, X, Y). \tag{1.3}$$

In general, it is not possible to minimize L over the whole set of all possible functions F but only over a given class of predictors C that corresponds to a set of practically computable predictors. In such a case, one obtains a predictor *f*** satisfying

$$f^{**} = argmin\ L(f, X, Y). \tag{1.4}$$

The predictor f** is not available neither in practice since we cannot minimize the loss function on C but only an empirical version of it based on a sample of size n [2]. The loss of performance due to the difference between f* and f** mainly depends on how relevant is our choice of model.

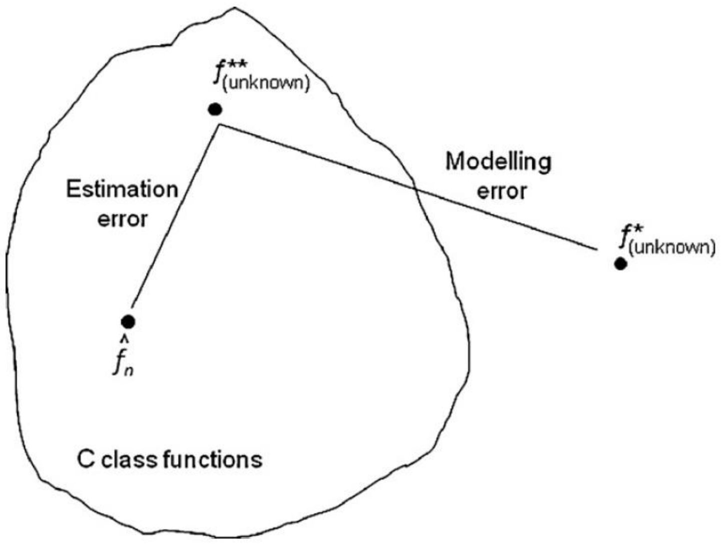


Figure 1.2*. Representation of best of all possible predictors (f*), best of all possible predictors inside the chosen class C (f**), and the empirical predictor ($\hat{f}_n$). [2]*

However, the gaussian process models are a statistical way of maximizing the model's accuracy without choosing just one model. In other words, we use all the possible models to the extent of each model's relevancy to a specific data point. This method is considered a non-parametric approach to modeling data.

1.3. Model Selection in Supervised Learning

In statistics and machine learning, the bias-variance tradeoff is the property of a set of predictive models whereby models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples and vice versa. The bias-variance dilemma or problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set:

- The bias is an error from erroneous assumptions in the learning algorithm. High bias (overly simple) can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).

- The variance is an error from sensitivity to small fluctuations in the training set. High variance (overly complex) can cause an algorithm to model the random noise in the training data rather than the intended outputs (overfitting). [5]

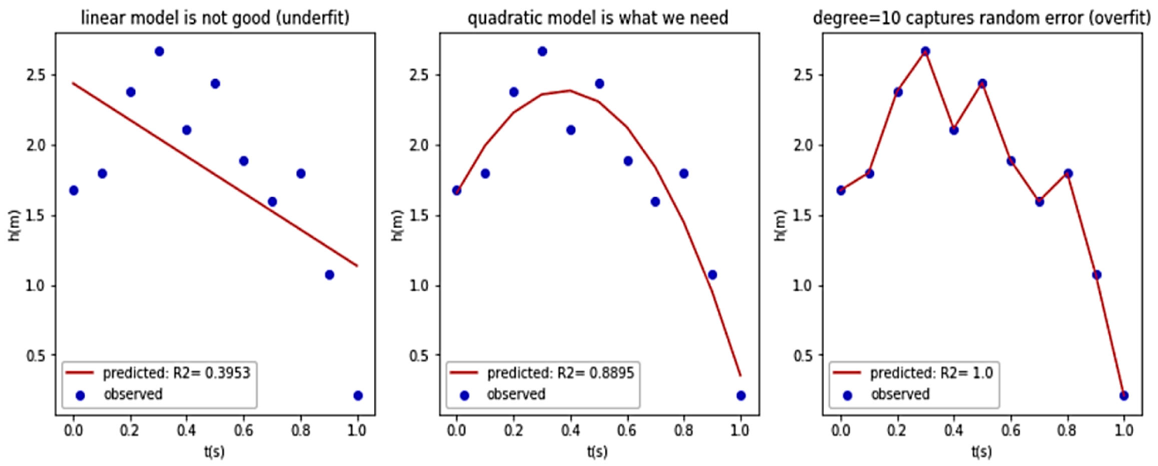


Figure 1.3. *Illustration of bias error (underfit) and variance error (overfit) [6].*

To illustrate the bias-variance problem, let us say we have a model with low errors on train data but does not perform as well on test data. In this case, the gap between train and test errors is high. It can be a result of overfitting, and it can be improved by methods like increasing the amount of training, bagging, regularization, or reducing complexity. On the other hand, when both train and test data errors are high, the model is under-fitted, and we need to increase the level of complexity.

One of the most influential concepts, yet challenging ones, is finding the proper relations between data features and representing the model based on these relations. This approach can capture nonlinear aspects of data that may be unavailable linearly. However, in most cases, there are many features in the data that make this approach almost computably impossible since we do not usually know what features we need to combine to make the prominent relations. The result would be extremely high dimensional data that cannot be computed.

2. Gaussian Processes

In linear regression, the data is trained, and a line is fitted to predict new variables. Despite its simplicity, when the data is too complicated, it often results in an underfit regression. Some ways can prevent underfitting such as adding features or increasing complexity in the model. Using kernel regression in these cases can be extremely helpful, but we can further take it by fitting a probability distribution instead of a line. We can now think of each data point's predicted value as a Gaussian distribution.

Gaussian Processes (GPs) are a non-parametric function and covariance approximation method. By definition, a gaussian process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions.[4] Alternatively, a GP is an arbitrary function defined as:

$$f\ (x) \sim\ GP\big(m(x), \kappa(x, x')\big), \tag{2.1}$$

Where *m(x)* is the mean function:

$$m(x) =\ E[\ f\ (x)] \tag{2.2}$$

And $\kappa(x, x')$ is the covariance function or the kernel:

$$\kappa(x, x')\ =\ E\big[\ \big(f\ (x) -\ m(x)\big)\big(f\ (x') -\ m(x')\big)\big]. \tag{2.3}$$

It is common practice to assume that the mean function is zero everywhere since uncertainty about the mean function can be considered by adding an extra term to the kernel. After accounting for the mean, a GP model's structure is entirely determined by its kernel. The kernel determines how the model generalizes or extrapolates to new data.

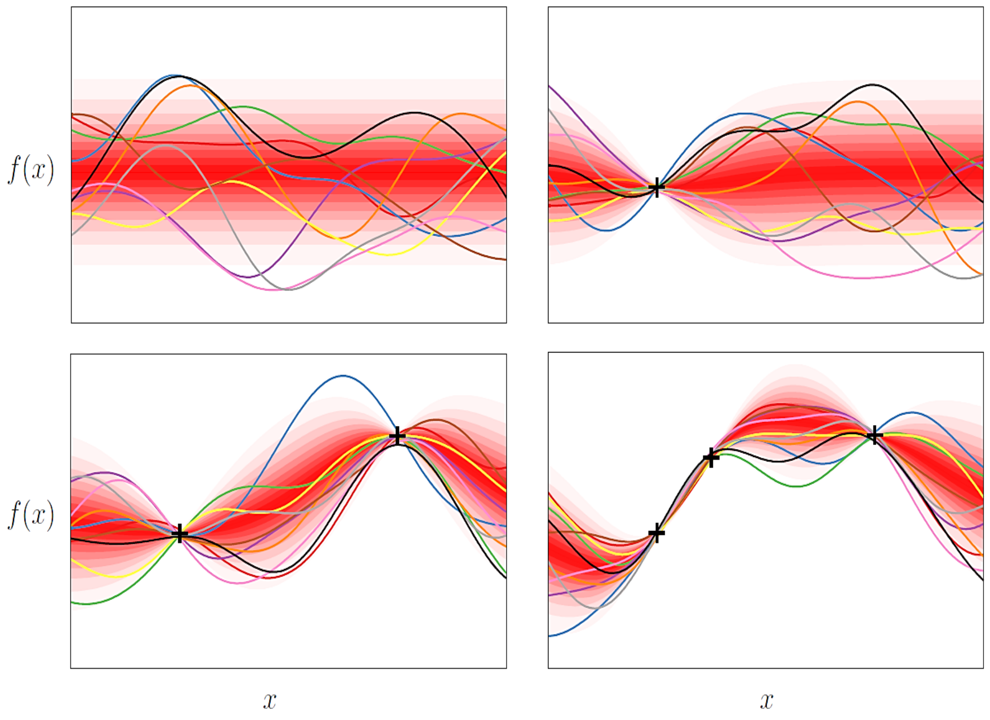


Figure 2.1. *A visual representation of a Gaussian process modeling a one-dimensional function. Different shades of red correspond to deciles of the predictive density at each input location. Colored lines show samples from the process – examples of some of the hypotheses included in the model. [7]*

There are many possible choices of covariance function, and we can specify a wide range of models just by specifying the kernel of a GP. For example, linear regression, splines, and Kalman filters are all GPs with particular kernels. However, these are just a few familiar examples out of a wide range of possibilities. One of the main difficulties in using GPs is constructing a kernel representing the particular structure present in the modeled data.

2.1. Model selection

The crucial property of GPs that allows us to construct models automatically is that we can compute the marginal likelihood of a dataset given a particular model, also known as the evidence. [7] The marginal likelihood allows one to compare models, balancing between the capacity of a model and how it is fitted to the data. [8] The marginal likelihood under a GP prior to a set of function values

$$[f(x_1), f(x_2), \dots, f(x_N)] := f(X)$$

at locations, X is given by:

$$p\big(f(X) \big| X, \mu(0), k(0,0)\big) = \mathcal{N}\big(f(X) \big| \mu(X), k(X,X)\big)$$

$$= (2\pi)^{-\frac{N}{2}} \times |k(X,X)|^{-\frac{1}{2}} \times e^{-\frac{1}{2}(f(X) - \mu(X))^T k(X,X)^{-1}(f(x) - \mu(X))} \quad (2.4)$$

This multivariate Gaussian density is referred to as the marginal likelihood because it implicitly marginalizes over all possible function values $f(\overline{X})$, where $\overline{X}$ is the set of all locations where we have not observed the function.

2.2. Prediction

The prior GP does not depend on the training data but specifies some properties of the functions. One of the primary goals of computing the posterior is that it can be used to make predictions for unseen test cases [7]. Let *f* be the known function values of the training cases, and let *f** be a set of function values corresponding to the test set inputs, *X**. Again, we write out the joint distribution of everything we are interested in:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim N\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^T & \Sigma_{**} \end{bmatrix}\right) \quad (2.5)$$

The training and test means are μ and μ_* respectively. For the covariance, we use Σ for the training set covariances, Σ_* for training-test set covariances and Σ_{**} for test set covariances. Since we know the values for the training set *f*, we are interested in the conditional distribution of f_* given *f*, which is defined as:

$$f_* | f \sim N(\mu_* + \Sigma_*^T \Sigma^{-1}(f - \mu), \Sigma_{**} - \Sigma_*^T \Sigma^{-1} \Sigma_*) \quad (2.6)$$

The last issue to consider is the training output noise. There are several ways to take noise into account. In the gaussian process model, every f(x) has a different covariance with itself only (since the noise is assumed independent), with a magnitude equal to the noise variance:

$$y(x) = f(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,\, \sigma_n^2) \tag{2.7}$$

The prior information cannot be easily specified since the components of a dataset are not always clear to us. For the GP techniques to be of value in practice, we must choose between different means and covariance [8]. This process is referred to as *training* the GP model.

2.3. Useful Properties of Gaussian Processes

There are several reasons why GPs, in particular, are well-suited for building a language of regression models [7]:

Logical inference: Given a kernel function and some observations, the predictive posterior distribution can be computed exactly in closed form. It is a rare property for non-parametric models to have.

Expressivity: Through the choice of covariance function, we can express a wide range of modeling assumptions.

Integration over hypotheses: The fact that a GP posterior, given a fixed kernel, lets us integrate precisely over a wide range of hypotheses which means that overfitting is less of an issue. For example, relatively few parameters need to be estimated compared to neural networks, which lessens the need for complex optimization or regularization schemes.

Model selection: A side benefit of integrating overall hypotheses is that we can compute the marginal likelihood of the data given a model. It gives us a principled way of comparing different models.

Closed-form predictive distribution: The predictive distribution of a GP at a set of test points is simply a multivariate Gaussian distribution, which means that GPs can quickly be composed with other models or decision procedures.

Easy to analyze: Simple models can be used as well-understood building blocks for constructing more exciting models. We can consider Gaussian processes as an extension of linear models which retain simplicity.

3. Kernels

Sometimes it is a good idea to add more features to the dataset to increase model accuracy. Whether it be a classification or a regression model, for example, if we have elements of radius and angle, the third feature area could be a more trusting feature for training. However, it is not always clear exactly which features we should add to improve the model. By creating all possible relations from existing features, capturing nonlinear aspects of data will be possible. The downside is getting a high-dimensional model that ordinary resources and limited time cannot compute. Nevertheless, if we express everything as a term of inner products, the problem would be much easier to solve.

3.1. Definition

A kernel (also called a covariance function, kernel function, or covariance kernel) is a positive-definite function of two inputs x, x′. The inputs are usually vectors in a Euclidean space, but kernels can also be defined on graphs, images, discrete or categorical inputs, or even text. Gaussian process models use a kernel to define the prior covariance between any two function values.

$$Cov\,[f(x), f(x')] = k(x, x') \tag{3.1}$$

The kernel specifies which functions are likely under the GP prior, which determines the model's generalization properties.

3.2. A few basic kernels

To begin understanding the types of structures expressible by GPs, we will start by briefly examining the priors on functions encoded by some commonly used kernels: the squared-exponential (SE), periodic (Per), and linear (Lin) kernels. (Figure 3.1)

Each covariance function corresponds to a different set of assumptions made about the function we wish to model. For example, using a squared-exp (SE) kernel implies that the function we are modeling has infinitely many derivatives. There are many variants of local kernels similar to the SE kernel, each encoding slightly different assumptions about the smoothness of the function being modeled.

Kernel parameters Each kernel has several parameters that specify the precise shape of the covariance function. These are sometimes referred to as hyper-parameters since they can be viewed as specifying a distribution over function parameters instead of being parameters that specify a function directly.

Stationary and Non-stationary The SE and Per kernels are stationary, meaning that their value only depends on the difference x−x′. It implies that the probability of observing a particular dataset remains the same even if we move all the x values by the same amount. In contrast, the linear kernel (Lin) is non-stationary. The corresponding GP model would produce different predictions if the data were moved while the kernel parameters were kept fixed.

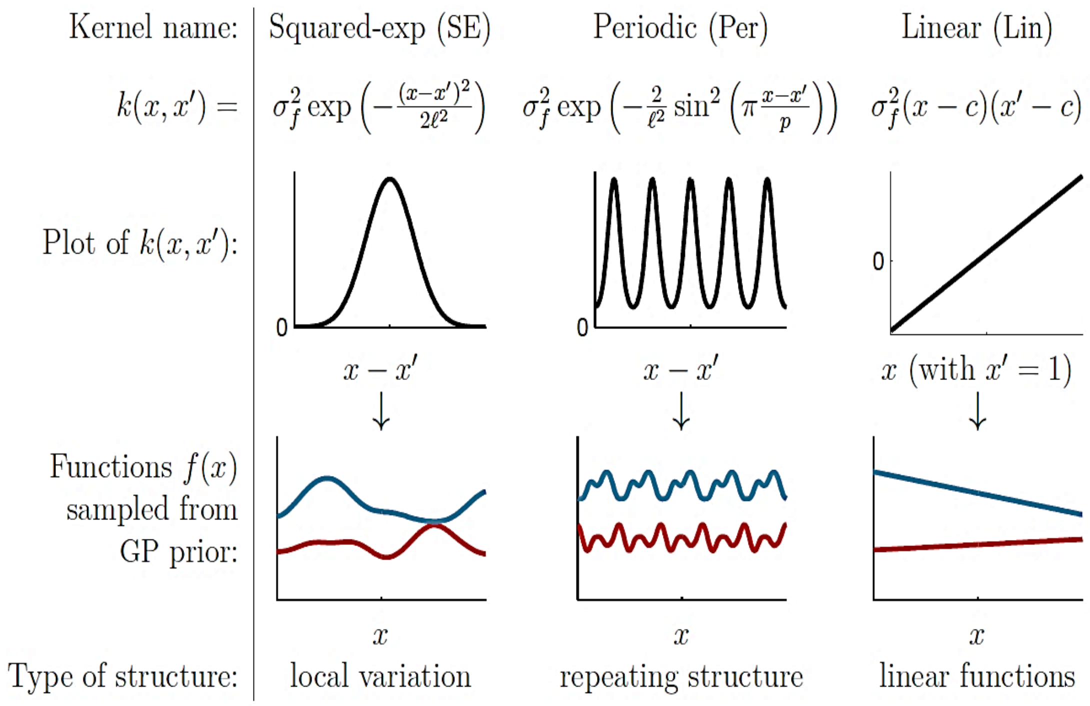


Figure 3.1. *Examples of structures expressible by some basic kernels [7].*

3.3. Combining kernels

Below, we will focus on two ways of combining kernels: addition and multiplication.

$$k_a + k_b = k_a(x,x') + k_b(x,x') \tag{3.2}$$

$$k_a \times k_b = k_a(x,x') \times k_b(x,x') \tag{3.3}$$

Kernels over multi-dimensional inputs can be constructed by adding and multiplying between kernels on different dimensions. A subscripted integer denotes the dimension on which a kernel operates. For example, SE_2 represents an SE kernel over the second dimension of vector x.

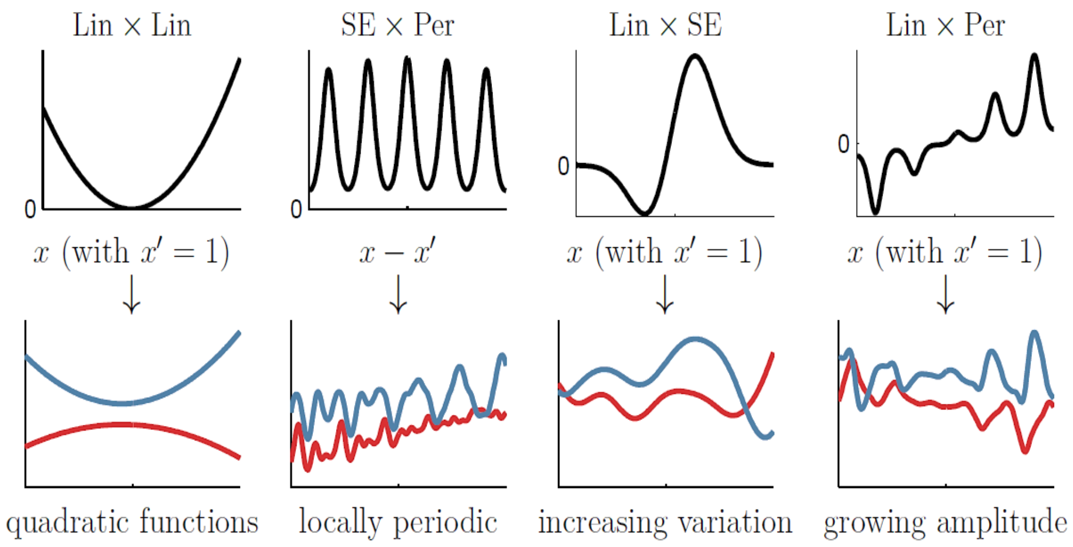


Figure 3.2. *Examples of one-dimensional structures expressible by multiplying kernels.*

Multiplying two positive-definite kernels together always results in another positive-definite kernel. (Figure 3.2)

Working with kernels, rather than the parametric form of the function itself, allows us to express high-level properties of functions that do not necessarily have a simple parametric form. Here, we discuss a few examples:

- Polynomial Regression: By multiplying together T linear kernels, we obtain a prior polynomial of degree T.
- Locally Periodic Functions. In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, the Per kernel corresponds to a strictly periodic structure, whereas Per×SE corresponds to a locally periodic structure.
- Functions with Growing Amplitude. Multiplying by a linear kernel means that the marginal standard deviation of the function being modeled grows linearly away from the location given by kernel parameter c.

One can multiply any number of kernels together in this way to produce kernels combining several high-level properties. For example, the kernel SE×Lin×Per specifies a prior on locally periodic functions with linearly growing amplitude.

A flexible way to model functions with more than one input is to multiply together kernels defined on each input. For example, a product of SE kernels over different dimensions, each having a different length scale parameter, is called the SE-ARD kernel. (Figure 3.3)

$$
\begin{aligned}
SE - ARD(x, x') &= \prod_{d=1}^{D} \sigma_d^2 \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) \\
&= \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) \qquad (3.4)
\end{aligned}
$$

SE-ARD kernels are the default kernel in most applications of GPs. It may be because they have relatively few parameters to estimate and because those parameters are relatively interpretable. In addition, there is a theoretical reason to use them: they are universal kernels, capable of learning any continuous function given enough data under some conditions. However, this flexibility means that they can sometimes be relatively slow to learn due to the curse of dimensionality. In general, the more structure we account for, the less data we need - the blessing of abstraction counters the curse of dimensionality. Below, we will investigate ways to encode more structure into kernels.

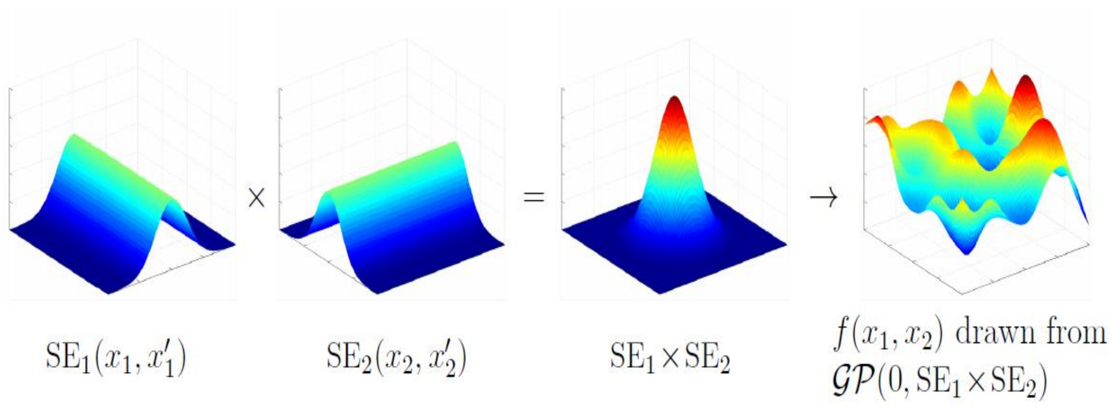


Figure 3.3. *A product of two one-dimensional kernels gives rise to a prior on functions. [7]*

3.4. Modeling sums of functions

An additive function can be expressed as $f(x) = f_a(x) + f_b(x)$. Additivity is a helpful modeling assumption in various contexts, especially if it allows us to make strong assumptions about the individual components. It is easy to encode additivity into GP models. Suppose functions f_a, f_b are drawn independently from GP priors:

$$f_a \sim \mathcal{GP}(\mu_a, k_a) \tag{3.5}$$

$$f_b \sim \mathcal{GP}(\mu_b, k_b) \tag{3.6}$$

Then the distribution of the sum of those functions is simply another GP,

$$f_a + f_b \sim \mathcal{GP}(\mu_a + \mu_b,\ k_a + k_b). \tag{3.7}$$

Kernels k_a and k_b can be of different types, allowing us to model the data as a sum of independent functions, each possibly representing a different type of structure. Any number of components can be summed this way.

3.4.1. Modeling noise

Additive noise can be modeled as an unknown, quickly-varying function added to the signal. This structure can be incorporated into a GP model by adding a local kernel such as an SE with a short length scale. (Figure 3.3) The limit of the SE kernel as its length-scale goes to zero is a "white noise" (WN) kernel. Function values drawn from a GP with a WN kernel are independent draws from a Gaussian random variable.

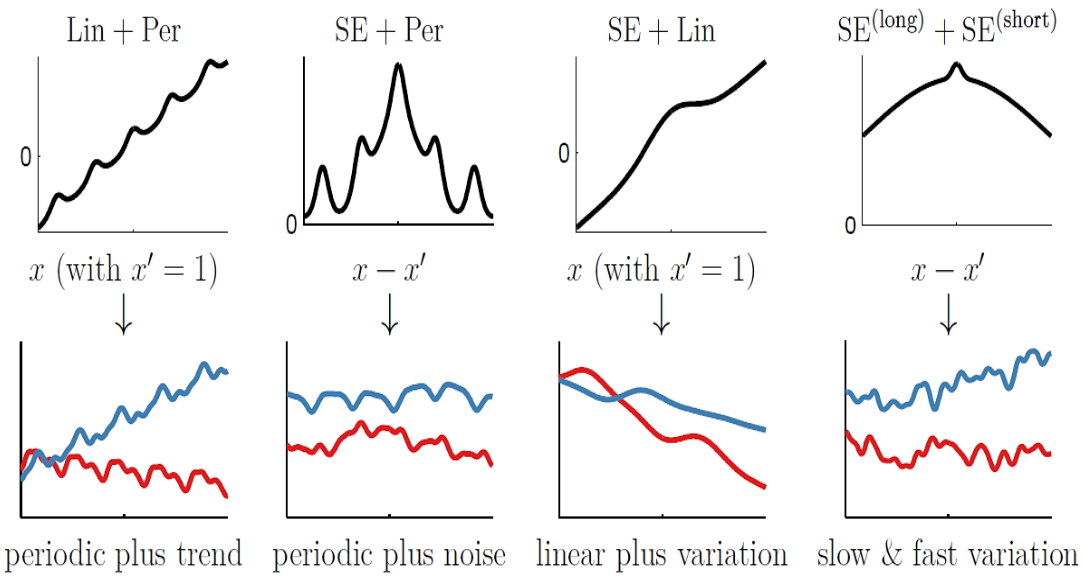


Figure 3.4. *Examples of one-dimensional structures expressible by adding kernels.*

3.4.2. Additivity across multiple dimensions

When modeling functions of multiple dimensions, summing kernels can give rise to additive structure across different dimensions. If the kernels being added together are each function of only a subset of input dimensions, then the implied prior over functions decomposes in the same way. For example,

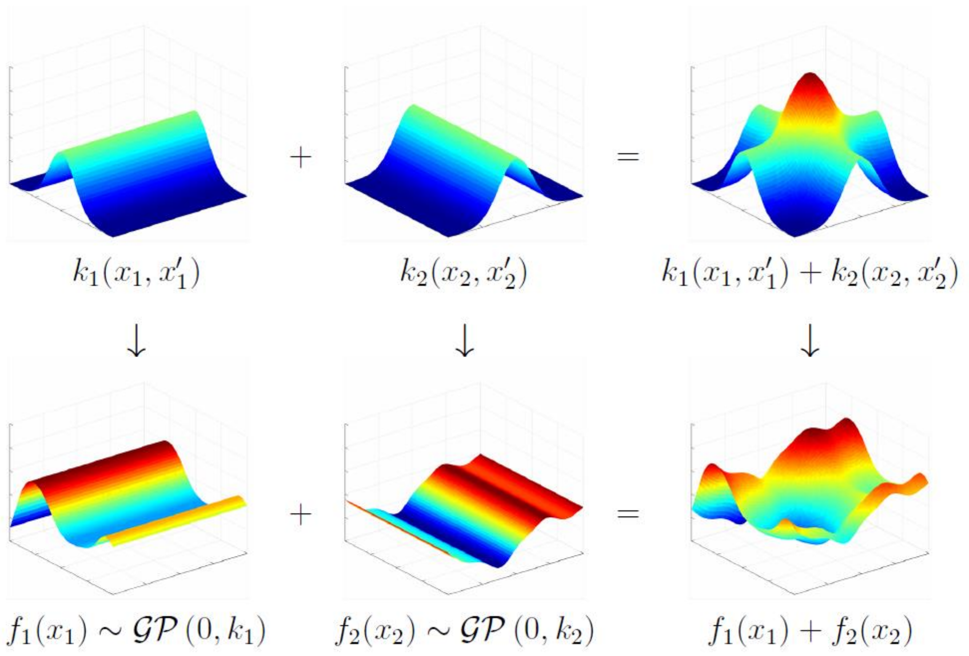
$$f(x_1, x_2) \sim \mathcal{GP}\left(0, k_1(x_1, x'_1) + k_2(x_2, x'_2)\right) \tag{3.8}$$

is equivalent to the model

$$f_1(x_1) \sim \mathcal{GP}\left(0, k_1(x_1, x'_1)\right), \tag{3.9}$$

$$f_2(x_2) \sim \mathcal{GP}\left(0, k_2(x_2, x'_2)\right), \tag{3.10}$$

$$f(x_1, x_2) = f_1(x_1) + f_2(x_2), \tag{3.11}$$



$k_1(x_1, x_1')$
+
$k_2(x_2, x_2')$
=
$k_1(x_1, x_1') + k_2(x_2, x_2')$
↓
↓
↓
$f_1(x_1) \sim \mathcal{GP}(0, k_1)$
+
$f_2(x_2) \sim \mathcal{GP}(0, k_2)$
=
$f_1(x_1) + f_2(x_2)$

Figure 3.5. *A sum of two orthogonal one-dimensional kernels. Top row: An additive kernel is a sum of kernels. Bottom row: A draw from an additive kernel corresponds to a sum of draws from independent GP priors, each having the corresponding kernel.*

3.4.3. Extrapolation through additivity

Additive structure sometimes allows us to make predictions far from the training data. Additive structure sometimes allows us to make predictions far from the training data. Figure 3.6 compares the extrapolations made by additive versus product-kernel GP models, conditioned on data from a sum of two axis-aligned sine functions. The training points were evaluated in a small, L-shaped area. In this example, the additive model can correctly predict the height of the function at an unseen combination of inputs.

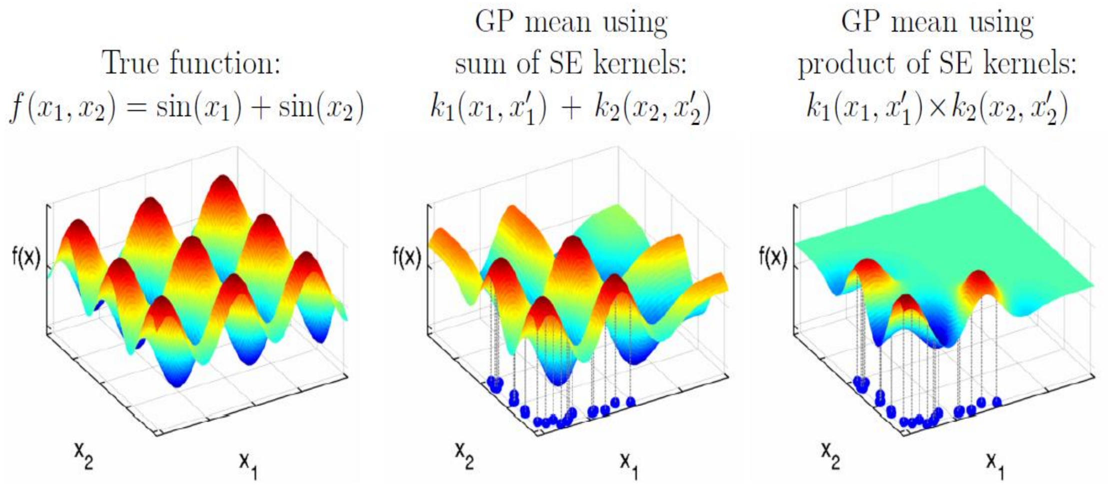


Figure 3.3. *Left: A function with additive structure. Center: A GP with an additive kernel can extrapolate away from the training data. Right: A GP with a product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data. This causes the predictions to revert to the mean.*

4. Multi-Output GPs

4.1. Multi-output GP

The Gaussian process (GP) is a Bayesian non-parametric model for time series that significantly impacts the machine learning community. GPs are designed through parametrizing a covariance kernel, meaning that constructing expressive kernels allows for an improved representation of complex signals. Recent advances extend the GP concept to multiple series (or channels). Both auto-correlations and cross-correlations among channels are designed jointly; we refer to these models as multi-output GP (MOGP) models. MOGP is a generalization of the robust Gaussian process (GP) predictive model. [9]It has been experimentally shown that by simultaneously exploiting correlations between multiple outputs and across the input space, it is possible to provide better predictions, particularly in scenarios with missing or noisy data. By definition, a m-channel multi-output Gaussian processes,

$$f(x) := \big(f_1(x), \dots, f_m(x)\big), \qquad x \in X$$

is a m-tuple of stochastic processes

$$f_p : X \to \mathbb{R} \qquad \forall p = 1, \dots, m$$

such that for any (finite) subset of inputs $\{x_i\}_{i=1}^N \subset X$, the random variables $\{f_{c(i)}(x_i)\}_{i=1}^N$ are jointly Gaussian for any choice of indices $c(i) \in \{1, \dots, m\}$.

For the single-output Gaussian process case, the usual practice for handling non-Gaussian likelihoods has been replacing the parameters or linear predictors of the non-Gaussian likelihood with one or more independent GP priors. Traditionally, each output is assumed to follow a Gaussian likelihood where the mean function is given by one of the outputs of the multi-output Gaussian processes. The variance in that distribution is treated as an unknown parameter [9]. The smoothness and generalization properties of GPs depend on the kernel function and its hyperparameters. Choosing an appropriate kernel function and its initial hyper-parameters based on prior knowledge from the data are the core steps of a GP. A classical approach to define cross-covariances for a multi-output GP is to linearly combine independent latent Gaussian processes, which is the case of the Linear Model of Coregionalization (LMC). Other popular multi-output GP models include the Cross-Spectral Mixture, the Convolutional Model, and the Multi-Output Spectral Mixture.

4.2. The Linear Model of Coregionalization (LMC)

The easiest way of building a permissible model of coregionalization is by building a set of intercorrelated random functions $f_d(x)$. The LMC builds each function $f_d(x)$ as a combination of independent random functions u_q^i with zero mean and primary covariance function [10]:

$$f_d(x) = \sum_{q=1}^{Q} \sum_{i=1}^{R} a_d^i, u_q^i \qquad (4.1)$$

Where:

$$cov\left[u\,{}_q^i(x),\, u_{q'}^{i'}(x')\right] = k_q(x, x') \qquad (4.2)$$

In the LMC, there are Q group of samples. For each group, there R_q samples obtained independently from the same GP with covariance $k_q(x, x')$. The cross-covariance between each two *f(x)* functions can be expressed as a linear combination of Q basic covariance models:

$$cov\,[\,f(x), f(x')] = \sum_{q=1}^{Q} A_q A_q^T\, k_q(x, x') = \sum_{q=1}^{Q} B_q k_q(x, x') \tag{4.3}$$

Where:

$$A_q = \left[a_q^1 a_q^2 \dots\ a_q^{R_q}\right] \tag{4.4}$$

In LMC, the resulting kernel is a function of both the covariance functions of the latent GPs and the parameters of the linear operator considered; this results in symmetric and centered cross-covariances. While these approaches are simple, they lack interpretability of the dependencies learned and force the auto-covariances to have similar behavior across different channels. The LMC method has also inspired the Cross-Spectral Mixture (CSM) kernel.

4.3. The Cross-Spectral Mixture (CSM)

MOGPs model temporal or spatial relationships among infinitely many random variables, as scalar GPs, and account for the statistical dependence across different data sources (or channels). For any two-input points *x*, x_0, the covariance function of an m-channel MOGP k (x, x_0) is a symmetric positive-definite m × m matrix of scalar covariance functions. The design of this matrix is challenging since we have to deal with the tradeoff between choosing a cross-variance of $\frac{m(m-1)}{2}$ cross covariances and meanwhile ensuring the positive definiteness of the symmetric matrix. In particular, unlike widely available families of auto-covariances, the cross-covariances are not bound to be positive definite. New covariance kernels have been proposed in, called Spectral Mixture (SM) kernels.

A Spectral Mixture (SM) kernel is a positive-definite stationary kernel given by

$$k(x, x') = \sum_{q=1}^{Q} w_q \exp\,(-\frac{1}{2}(x - x')^T \sum_{q} (x - x'))\cos\left(\mu_q^T(x - x')\right) \tag{4.5}$$

An SM kernel is derived through modeling spectral densities (Fourier transform of a kernel) with a Gaussian mixture. A desirable property of SM kernels is that they can be used to reconstruct other popular standard covariance kernels. SM kernel models phase differences across channels by manually introducing a shift between the cosine and exponential factors.

5. The MedGP Method

5.1. The Data

The large amount of electronic health records (EHRs) has helped understand the disease progress, early diagnosis, and personalized treatments for many clinical diseases. EHRs contain rich patient information (disease history, demographics, vital signs, and lab results) that clinicians use to diagnose and treat patients. *The BMC Medical Informatics and Decision Making* have published a paper [11] regarding the MedGP method. This statistical framework enables personalized, real-time monitoring of hospital patients using the medical data from a set of reference patients.

The datasets are from the hospitals at the University of Pennsylvania (HUP), containing information for over 260,000 patients and the public Multiparameter Intelligent Monitoring in Intensive Care (MIMIC-III) data set with more than 53,000 admissions from 38,000 patients in intensive care units (ICUs) [11].

The diagnosis of some diseases, such as sepsis, is difficult due to heterogeneous symptoms across patients. A time-to-event prediction for septic shock would significantly improve if built upon an underlying model of the patient state. Predicting septic shock without a model of patient state is challenging since many of the covariates (ex. Lab results) are sparsely sampled across patients. Time-to-event models thus benefit significantly from using a patient state model to avoid these challenging properties of medical data in the downstream analysis [11].

However, the time intervals between observations are non-uniform, and no two observations are generally taken simultaneously. The sparsity over patients and uncalibrated time series makes the physiological progression of the patient state within patients or joint analysis of time series across patients challenging to model using many existing time-series analyses. Gaussian processes (GPs) are practical approaches for time-series analysis because they can naturally capture irregular time-series observations and estimate prediction uncertainties in a probabilistic framework. For these reasons, GPs have been applied to medical time series data [11].

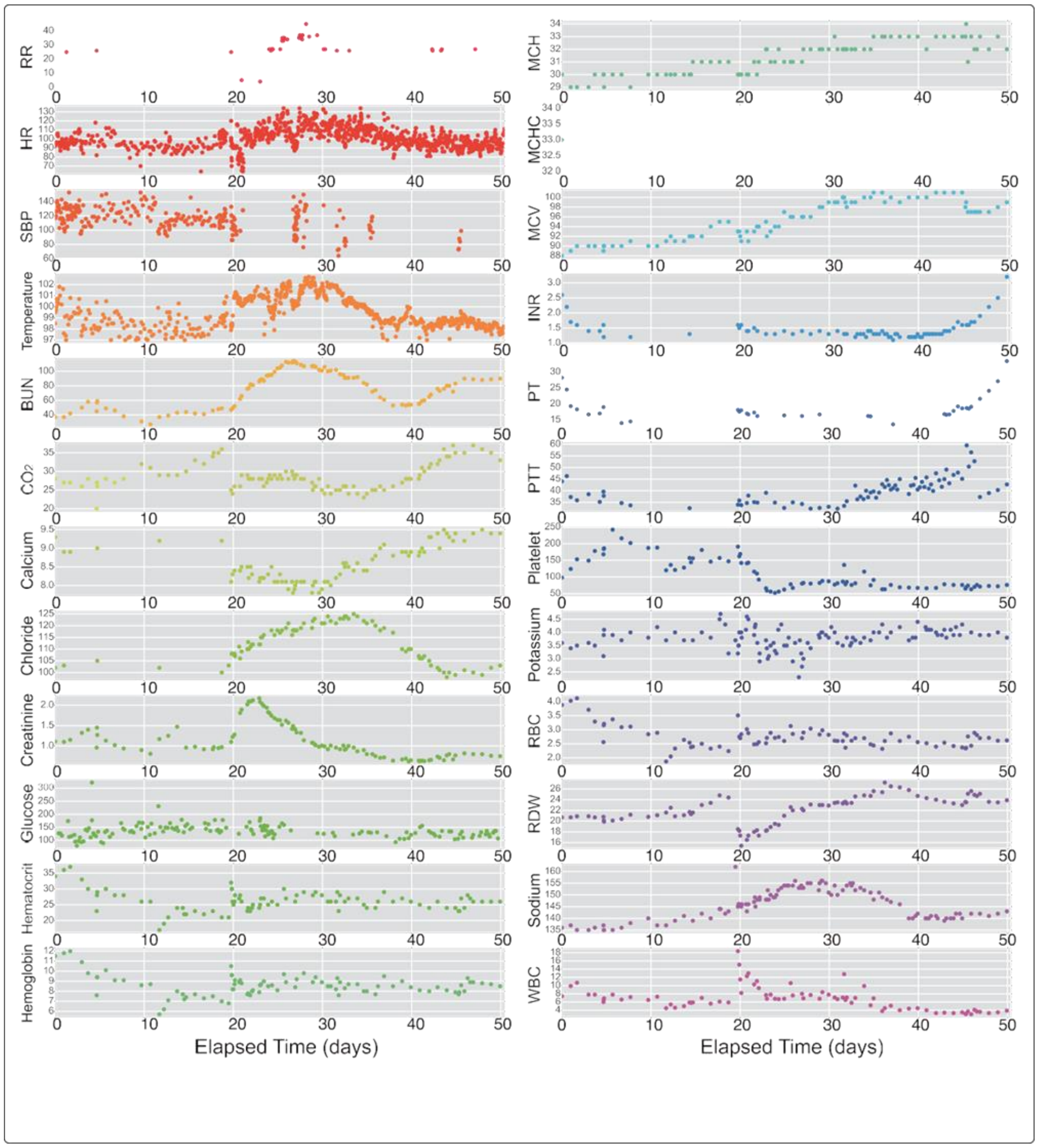


Figure 4.1. *An example of time series data of 24 clinical covariates for a septic patient in the HUP data. The 24 covariates include four vital signs—respiratory rate (RR), heart rate (HR), systolic blood pressure (SBP), body temperature—and 20 lab results. The time series are aligned by the patient's admission time. [11]*

5.2. Methods

The MedGP uses the gaussian process models for capturing the time-to-event models across covariates of each patient. For the covariance function, the squared exponential (SE) kernel is combined with the periodic function to capture the unknown source-specific smoothness and periodicity of the trajectories of clinical covariates [11]. (See section 3.4)

5.2.1. Gaussian process regression with multi-output kernel

The first goal is to model multiple clinical covariates jointly—vital signs and lab tests—over time for each patient using GP regression. For the ith patient, the time series of the dth covariate is donated as a vector $x_{i,d}$, representing the time points that the dth covariate was observed, and the corresponding observation vector $y_{i,d}$:

$$x_{i,d}^T = \left[x_{i,d,1}, x_{i,d,2}, \ldots, x_{i,d,T_{i,d}}\right], \tag{5.1}$$

$$y_{i,d}^T = \left[y_{i,d,1}, y_{i,d,2}, \ldots, y_{i,d,T_{i,d}}\right], \tag{5.2}$$

Where t indexes time and $T_{i,d}$ is the total number of observations for the dth covariate of the ith patient. The relationship between time and clinical observations is captured as a GP model,

$$\mathcal{F}_i \sim \mathcal{GP}_i\left(\mu_i(\boldsymbol{x}), \kappa_i(\boldsymbol{x}, \boldsymbol{x})\right). \tag{5.3}$$

Where mean mu = 0, and the linear model of coregionalization (LMC) is adapted for kernel function (See section 4.2). The kernel for the cross-covariance of any pair of covariate types is modeled by a weighted structured linear mixture of the Q basis kernels [11]. The entire joint kernel is written as a block-structured function,

$$k_i(x_i, x'_i) = \sum_{q=1}^{Q} \begin{bmatrix} b_{q,(1,1)} k_q\left(x_{i,1,}, x'_{i,1}\right) & \cdots & b_{q,(1,D)} k_q\left(x_{i,1,}, x'_{i,D}\right) \\ \vdots & \ddots & \vdots \\ b_{q,(D,1)} k_q\left(x_{i,D,}, x'_{i,1}\right) & \cdots & b_{q,(D,D)} k_q\left(x_{i,D,}, x'_{i,D}\right) \end{bmatrix}, \tag{5.4}$$

Where $b_{q,(D,D')}$ scales the covariance (defined by the qth basis kernel) between covariates d and d', and $k_i(x_i, x'_i) \in \mathbb{R}^{T_i \times T_i}$.

The time-series observations, such as periodicity and short-term dependencies, are captured in the Q basis kernels. The spectral mixture (SM) kernel is chosen as the basis kernel to handle the heterogeneity of patterns within covariates and across patients (See section 4.3). The SM kernel is a general form of stationary kernels such as SE and periodic functions. It has also shown good performance in modeling processes generated from more complex kernels through a mixture of kernels approaches [11]. The basis kernel $\kappa_q(x_t, x_{t'})$ is written as

$$\kappa_q(x_t, x_{t'}) = \ exp\left(-2\pi^2\rho^2 v_q\right) cos\left(2\pi\rho\mu_q\right), \tag{5.5}$$

5.2.2. Sparsity-inducing priors

As the number of medical covariates included in the model increases, we need to increase the number of basis kernels Q and corresponding Rq to allow greater representational flexibility. However, too many basis kernels may lead to overfitting and will become computationally intractable. To avoid this, we regularized the elements of each weight matrix B_q by applying structured sparsity-inducing priors on each A_q matrix [11].

The effect of element-wise sparsity is to perform model selection on the number of basis kernels that each pair of covariates uses for covariance representation. The sparse inducing prior is first applied to the columns of A_q, and secondly, it regularizes each element of the matrix.

5.2.3. Estimating the population-level model and online updating

We now describe the process of building a population-level empirical prior from a set of mixture kernels estimated from all training patients and how we apply this empirical prior to a new patient. The idea here is that when we estimate a set of patient-specific mixture kernels, we would like to understand the high-level properties of these mixture kernels shared across patients in the same patient group. Then, we can estimate the group-specific distributions of the hyperparameters through the estimates of basis kernels belonging to each cluster [11].

We can do this by building an empirical distribution using kernel density estimation (KDE) with a Gaussian kernel over the GP kernel hyperparameters assigned to that cluster. To allow online updating, we estimated the new empirical Aq matrix and λ_q vector corresponding to each new B_q matrix. The kernel is sequentially updated every time a new observation arrives. However, we do not update the values if the elements were set to near zero in the empirical prior to maintain the empirical sparsity structure.

References

[1] "Machine Learning," IBM Cloud, [Online]. Available: https://www.ibm.com/nl-en/cloud/learn/machine-learning.

[2] B. G. G. P. C. Crisci, "A review of supervised machine learning algorithms and their applications to," *Ecological Modeling,* 2012.

[3] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification," 2007.

[4] R. Parmar, "Common Loss functions in machine learning," [Online]. Available: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[5] B. O. Tayo, "Simplicity vs. Complexity in Machine Learning," 11 Nov 2019. [Online]. Available: Simplicity vs. Complexity in Machine Learning — https://towardsdatascience.com/simplicity-vs-complexity-in-machine-learning-finding-the-right-balance-c9000d1726fb.

[6] B. O. Tayo, "Bias-Variance Tradeoff Illustration Using Pylab," 7 May 2019. [Online]. Available: Bias-Variance Tradeoff https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78.

[7] D. K. Duvenaud, "Automatic Model Construction with Gaussian Processes," *University of Cambridge,* 2014.

[8] C. K. I. Williams and C. E. Rasmussen, Gaussian Processes for Machine Learning, 2006.

[9] A. A.-R. M. A. Á. Pablo Moreno-Muñoz, "Heterogeneous Multi-output Gaussian Process Prediction," 2018.

[10] P. Goovaerts, Geostatistics for Natural Resources Evaluation, Oxford University Press, 1997.

[11] S. m.-o. G. p. f. o. m. t. s. prediction, "Li-Fang Cheng, Bianca Dumitrascu, Gregory Darnell, Corey Chivers, Michael Draugelis, Kai Li, Barbara E Engelhardt," *BMC Medical Informatics and DecisionMaking,* 2020.

[12] A. E. M. N. Roya Nikjoo, "Non-parametric Regression Model for Continuous-time Day," *KTH Royal Institute of Technology, Stockholm, Sweden.*

[13] T. R. S. J. A. R. F. R. B. C. Glaura C. Franco, "Confidence Intervals for the Hyperparameters in Structural," *Federal University of Minas Gerais, MG, Brazil,* 2006.

[14] R. P. A. Andrew Gordon Wilson, "Gaussian Process Kernels for Pattern Discovery and Extrapolation."

[15] P. G. J. C. E. M. Kai Chen, "Generalized Spectral Mixture Kernels for Multi-Task," 2018.

[16] A. C. F. T. Taco de Wol, "MOGPTK: The Multi-Output Gaussian Process Toolkit," 2020.

[17] C. W. S. S. T. Christian Fiedler, "Practical and Rigorous Uncertainty Bounds for Gaussian Process Regression," 2021.

[18] D. K. Vladimir Joukov, "Fast Approximate Multi-output Gaussian," 2020.

[19] J. M.-M. M. C.-T. Anna Mateo-Sanchis, "Gap Filling of Biophysical Parameter Time Series," 2012.

[20] K. P. R. D. P. F. R. Daniele Gammell, "Generalized Multi-Output Gaussian Process Censored Regression," 2009.

[21] K. M. A. C. C. K. I. W. Edwin V. Bonilla, "Multi-task Gaussian Process Prediction."

[22] F. T. Gabriel Parra, "Spectral Mixture Kernels for Multi-Output Gaussian Processes," 2017.

وزارت علوم، تحقیقات و فناوری
دانشگاه تحصیلات تکمیلی علوم پایه
گاوازنگ، زنجان

مطالعه الگوریتم گاوسین چند-خروجی بررروی داده های ناهمگون پزشکی

پایان نامه کارشناسی

سولماز محمدی

استاد راهنما: دکتر محمدرضا فرجی

اسفند ۱۳۹۹

چکیده

فرآیند های گاوسین یک روش غیرپارامتری در یادگیری ماشین است که با استفاده از اصل بیز، می تواند کواریانس را به شکلی تخمین بزند که مدلی صاف و هموار داشته باشیم. الگوریتم های چند-خروجی یکی از شاخه های گسترش این روش هستند که پیشرفت چشمگیری را در تفسیر روابط بین داده های ناهمگون نشان داده‌اند. در این پروژه، اصول فرآیندهای گاوسین و الگوریتم های کواریانس مورد استفاده را مرور می کنیم. علاوه بر این، نتایج اعمال روش فرآیند های چند-خروجی بر روی حجم عظیمی از سوابق پزشکی که ناهمگون هستند را نشان می دهیم. در این روش، ترکیبی از الگوریتم های مختلف جهت طراحی یک تابع کواریانس استفاده شده است و با استفاده از روش فرآیند های چند خروجی مدلی ساخته می شود که قادر به تخمین بیماری هر یک از بیماران جدید به صورت لحظه ای باشد.