Gaussian Processes for Prediction of Metabolite Concentration in Time Series

Contents

Abstract

Metabolomics is the study of metabolites which underpin life. These metabolites can act as indicators of the physiological state of the human body based on their concentrations and give insight into potential disease diagnosis, as different diseases can be characterized by the abundance or deficiency of metabolites, such as prostate cancer and acute ischemic stroke. Biological data, by nature, is messy and noisy, and often requires heavy pre-processing and complex learning machine learning algorithms to map inputs to outputs using unknown functions. Gaussian processes (GP) are a form of non-parametric Bayesian machine learning which construct a prior distribution over the space of functions. GPs can be used for a multitude of different tasks and are a great tool for modelling unknown functions and noisy data such as biological data.

In this project we look at GP regression as a way to predict metabolite concentrations given a rate of change from its reaction. This would open up the possibility, in a clinical environment such as a doctors office, to test patients for a variety of diseases based on the concentration of a given metabolite. Samadhan *et al.* have developed a complimentary metal oxide semiconductor (CMOS) based device to measure concentrations of metabolites in serum, buffer and urine. The data from this research is used in this project. We first introduce GPs and the relevant mathematical background. We then develop two algorithms for data pre-processing specific to the dataset: (1) an algorithm to classify experimental pixels with metabolites from the CMOS-based device; and (2), an online algorithm to identify a metabolites reaction start over time. Lastly, we create GP regression models using the pre-processed data for prediction of metabolite concentration.

Keywords – Gaussian processes, machine learning, regression, prediction, time series, metabolomics, CMOS, point-of-care detection

Chapter 1

Introduction

With the ever increasing amount of data gathered and stored, artificial intelligence and machine learning have both become extremely popular and have seen many ground breaking and novel innovations within the past decade. While artificial intelligence concerns itself with attempting to recreate human intelligence in computers or robots, broadly speaking, machine learning is a subset which focuses on giving machines the ability to learn and make decisions based on statistical models. Machine learning has seen great success in areas such as finance, social media and science to predict stock prices or learn who is allowed loans, target users with specific adverts, design molecules, or attempt to discover cures for diseases such as ALS [1]. More than ever, the use of machine learning to process and learn from biological data is crucial in areas of biology such as medicine and drug discovery due to the sheer amount of data that is available, produced and researched in these areas. This chapter will give a short introduction to machine learning and its concepts alongside metabolomics, the study of chemical processes involving metabolites, and briefly discusses the experimental setup and data used in this project.

1.1 Metabolomics

Metabolites are the small molecules that underpin life and the chemical and biological reactions that occur in our bodies. A metabolite is the intermediate product of metabolism and is restricted to small molecules, such as glucose. Metabolism is the process that occurs in our bodies and every biological system. It can be split into anabolism and catabolism, which use energy to construct molecules from smaller units, and break down said specific molecules into smaller units which are then oxidized or used as energy, respectively. This is shown in Figure (1.1.1) which illustrates the breakdown of carbohydrates into its respective metabolites, of which glucose is one. It is an endless cyclic. Many diseases are accompanied by perturbations to normal levels of metabolites in our bodies within biofluids [2]. Based on these perturbations we can detect diseases such as recurrent breast cancer [3], carry out cardiac research [4], profile cancer cells [5] and personalize medicine [6]. As an example, in urine samples, sarcosine, which is a non-proteinogenic amino acid metabolite, normal levels are around 0.2μM. In individuals with prostate cancer it can reach elevated levels of $\sim 5\mu M$ [7].

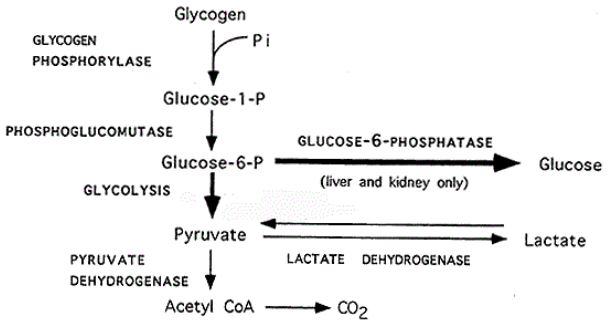


Figure 1.1.1: The breakdown of Glycogen resulting in Glucose as an intermediate metabolite [8]

While metabolomics has only recently been applied to drug development and discovery, it was first demonstrated in 1960s [9]. Central to the use of metabolomics in disease discovery have been technologies that measure

all small molecules within a given system. A problem, however, is that the technology for measuring such small molecules is either lacking sensitivity, such as Nuclear Magnetic Resonance (NMR), or lacking the ability to quantify metabolites, such as as Hyphenated Mass Spectrometry (HMS) [2]. Hence, Samadhan B. Patil *et al.* [2] have developed a *complimentary metal oxide semiconductor* device (CMOS), which is a low cost device for sensing metabolite reactions. Exploiting the ultra-high packing density of CMOS based sensors. An advantage of the CMOS-based chip surface is that many metabolites can be simultaneously measured and tested, lowering the overall cost for metabolite measurements. Samadhan *et al.* [2] and Hu *et al.* [10] have demonstrated the ability to measure glucose and cholesterol concentrations, simultaneously, using a CMOS-based device.

Samadhan *et al.* [2] demonstrate the ability to identify metabolite markers simultaneously in order to detect diseases which are of utmost importance. For example, Acute Myocardial Infraction (AMI), Acute Ischemic Stroke (AIS) and prostate cancer. In the article, 4 key metabolites are used: sarcosine, xanthine, choline and cholesterol. Each of these metabolites has been shown to be implicated in one or more critical diseases such as AMI, AIS, Acute Renal Failure and prostate cancer. Measurements were carried out on metabolites dissolved in various solvents including buffer, serum and urine. Assays for these metabolites were then optimized, standardised and quantified on the CMOS chip surface.

1.1.1 Setup & Data

The CMOS chip is a 3.4mM × 3.6mM chip with an array of 16 × 16 sensor pixels with an active area of 1.6mM × 1.6mM. These sensor pixels may also be referred to as "cells". The CMOS chip contains 4 micro-wells which are used to protect the active areas and are 600µM ×600µM in size [2]. The four micro-wells have pipettable access. It is inside these active area micro-wells where reactions take place with the assays, illustrated in Figures (A.1) and (A.2). Below the micro-wells are photo-diodes which allow for metabolite sensing and measuring the intensity values. This is the CMOS chip. Figure (1.1.2) shows the measured metabolites averaged over the CMOS chip.

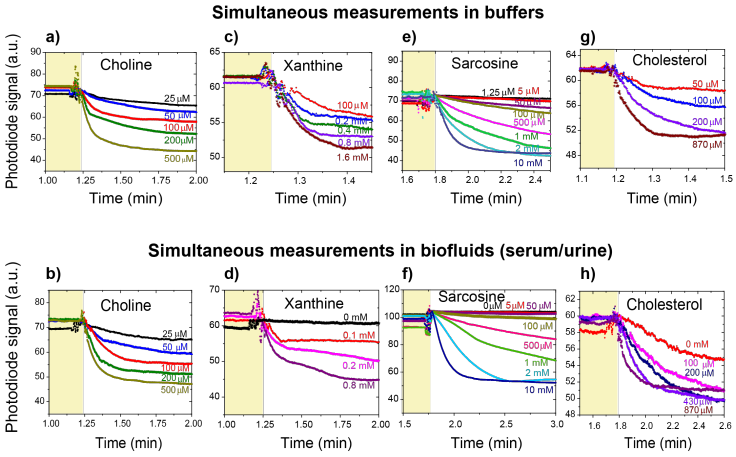


Figure 1.1.2: Response of photodiode sensor array from four micro-wells for simultaneous detection of four metabolites [2]

Each micro-well was intentionally designed to be an array of size 5 × 5 pixels on the 16 × 16 pixel CMOS chip [2]. Due to the limits in manufacturing precision, this was not always the case, and the active areas where reactions were occurring, deviated one row or column from the intended 5 × 5 pattern. Data from the entire CMOS chip is recorded and stored in TDMS files. Therefore, in the dataset, pixels which do not contain metabolites, and their values, are present. From this point onwards, pixels which are not contained within a micro-well, or active area, will be referred to as barrier pixels, and the others as non-barrier pixels. An important feature of our data is that only the top two micro-wells were the ones with the metabolites where reactions were happening. These are our experimental micro-wells. The bottom two micro-wells are control and contain water.

The TDMS files were labelled with the metabolite name, concentration, and whether the experiment was in buffer, serum or urine. The data within them was represented as 16 columns and 16.T rows, where T is the number of recorded timesteps. After reshaping the data in `numpy`, this equated in data arrays of size 16 × 16, as per the specifications, containing the pixel intensities. Each experiment contained around $5000 - 7000$ timesteps of data.

We can render this data at 10Hz to visualize reactions from start to finish.

1.2 Machine Learning & Gaussian Processes

Machine learning is the study that develops statistical models that learn from experience. Machine learning algorithms can be divided into groups, two of the most popular being supervised and unsupervised learning algorithms. In supervised learning, the data given to the machine learning model is associated with a ground-truth label so that the algorithm learns from examples to predict the label. Whereas in unsupervised learning, the data is not so clearly labelled and the machine learning model aims to discover patterns amongst the data, such as clustering pictures of faces together without having a concept of what a face is.

Biological data is commonly very noisy by nature and large in size, such as DNA data. For this reason it is often very challenging to learn a function to try and describe this data, as is the case in supervised parametric machine learning, such as linear regression. Because of the type and amount of data produced in biology and metabolomics, it is often very hard to find or derive functions that accurately fit this data to learn from. Problems such as optimizing combinations of genes for transgenics [11] require us to use unknown functions that map inputs to outputs, $f(x) = y$. This function is often difficult to evaluate, especially given the data and problem. Gaussian Processes (GP) however, and more specifically GP regression, can prove very useful in these situations where it is hard to estimate a function. GP regression is a form of non-parametric Bayesian machine learning that is able to capture these difficult and complicated relationships within the data by making use of infinite parameters through Bayesian inference [12].

1.3 Research Objective & Contributions

This project aims to use Gaussian process regression to predict the concentration of a metabolite based on a rate of change from the metabolites reaction start point. We use data from Samadhan et al. [2] and the CMOS chip to build our Gaussian process model. Also part of the project is the development of two algorithms for the data preprocessing which classify experimental pixels and estimate the reaction start point of a given experimental pixel. These algorithms are jointly used to produce the final dataset used in the training and testing of the Gaussian process model for prediction of metabolite concentrations.

1.4 Thesis Structure

- In Chapter 2 we begin by discussing linear regression as a starting point for regression in machine learning so that the move from to Gaussian Process regression is intuitive and easy to follow. As key background, we will also look at Bayesian regression and inference in weight-space view and function-space view, conditional probability, univariate and multivariate normal distributions and covariance matrices before looking at Gaussian Processes.
- Chapter 3 looks at the data gathered by the CMOS chip and the implementation and methodology for preparing the data for use with the GP regression model. This involves the development of two algorithms to classify and extract relevant data.
- Chapter 4 evaluates the proposed algorithms and Gaussian process model. We will look at the accuracy of both algorithms and how well they performed overall as well as the results of the GP regression model and its accuracy for the training data prepared in previous chapters.
- Finally, in Chapter 5 we discuss future work to improve results as well as drawing conclusions from this study.

Chapter 2

Background

The aim of this dissertation is to use Gaussian Process (GP) regression to estimate the concentration of a metabolite in buffer, serum or urine given a rate of change in the reaction, we will first introduce regression by looking at linear regression. Then, we will give intuition for Bayesian linear regression and Bayesian inference. We will then discuss joint, marginal and conditional probability, multivariate normal distributions, and covariance matrices, all of which are key in understanding GP regression.

2.1 Linear Regression

Regression algorithms are among the most commonly used in machine learning applications. They are parametric or non-parametric supervised learning algorithms and involve learning a mapping from inputs to continuously valued outputs, unlike classification which takes an input and maps it to a discrete label, such as classifying a picture of an animal as either a cat or a dog. An example of a regression problem is predicting the Mens 100m Olympic Race winning time for 2020 based on previous years' data. Regression, and more specifically, least squares regression, was first used and published in the 19th century by Johann Carl-Friedrich Gauss [13] and Adrien-Marie Legendre [14]. Linear regression is a useful but simple form of statistical learning. Because of its simplicity it is often times used as a starting point before exploring more complex machine learning techniques.

In regression, we have a training dataset $\mathcal{D}$ of n observations, $\mathcal{D} = \{(x_i, y_i)|i = 1, ..., n\}$, where x_i is some input data and y_i is the corresponding output variable. Given this training dataset $\mathcal{D}$, a regression model estimates a function $y = f(x_i, w)$ in order to make predictions of y for a new data point x^*. Linear regression assumes a linear relationship between the output y and input x. The one dimensional linear model can be written as

$$f(x_i) = w_0 + \sum_{j=1}^{D} w_j x_{ij}, \tag{2.1.1}$$

where D is the dimensionality of x_i. In a one dimensional linear regression our two parameters w_0 and w_i define the intercept and the slope on the response of increasing a predictor value, respectively. Linear methods of regression work well on data that clearly is linear but also on small data or data that has a low signal-to-noise ratio. Linear regression can also be applied to transformations of the inputs, which are called basis-function methods.

Parametric problems rely on the fact that we must first make assumptions about the function f, and in the case of our linear regression, that f is linear. Because we have made assumptions about f we now only need to estimate parameters. This can be accomplished by first introducing a loss function to fit our parameters. A loss function is a method of evaluating the magnitude of error of our model. The most commonly used method for this in linear regression is called the *least squares* function or *residual sum of squares*, eq. (2.1.3). Note that we use $\frac{1}{N}$ as we want the average loss across our whole dataset and argmin to find the best values for our parameters to minimize our loss function.

$$\mathcal{L} = \underset{w_0, w_j}{\operatorname{argmin}} \sum_{i=1}^{N} \Big(y_i - f(x_i) \Big)^2 \tag{2.1.2}$$

$$= \underset{w_0, w_j}{\operatorname{argmin}} \sum_{i=1}^{N} \Big(y_i - w_0 - \sum_{j=1}^{D} w_j x_{ij} \Big)^2 \tag{2.1.3}$$

For a simple two parameter linear regression function of the form $y = w_0 + w_1 x_i$, we need to differentiate the loss function in order to find the partial derivatives with respect to w_0 and w_1 [15]. We also can use least squares to accurately choose model complexity, because as the model complexity gets higher, we can begin to overfit the data. As an example, this may occur when using high order polynomial functions.

In non-parametric models, we do not make explicit assumptions about the parametrization of f. Instead, we make assumptions about the covariance structure in other data. It has the advantage that it can more accurately fit a greater selection of data as we can capture more subtle aspects of the data and the parameters can be of infinite dimensions, resulting in more freedom and flexibility within our model. A non-parametric model can generally be shown as $y = m(\mathbf{x}_i) + \epsilon_i,\ i = 1, ..., n$ where m is an unknown regression function.

2.2 Bayesian Regression & Inference

2.2.1 Joint, Marginal & Conditional Probability

To be able to understand GPs better we must first look at joint, marginal and conditional probability. It is important to note, we can manipulate among these probabilities to derive others. In a joint probability, we are told the probability of both $\mathbf{y}_A$ and $\mathbf{y}_B$ happening, as the intersection. Joint probabilities are expressed as

$$p(\mathbf{y}_A, \mathbf{y}_B ... \mathbf{y}_N). \tag{2.2.1}$$

The marginal probability is the probability of any single event occurring unconditioned on any other events. It tells us the probability of event $\mathbf{y}_A$ happening, such as whether the weather is going to be sunny or rainy on a particular day. Marginal probability can be represented in two different ways, depending if variables are continuous (2.2.2) or discrete (2.2.3),

$$p(\mathbf{y}_A) = \int p(\mathbf{y}_A, \mathbf{y}_B) d\mathbf{y}_B \tag{2.2.2}$$

$$p(X = x) = \sum_{y} p(X = x, Y = y). \tag{2.2.3}$$

Lastly, the conditional probability tells us the probability of $\mathbf{y}_B$ occurring if $\mathbf{y}_A$ is observed. For example, we can use it to calculate the probability of rain given the sun is out. Essentially, conditional probability measures the likelihood of something happening given something else has happened. It is defined as

$$p(\mathbf{y}_A | \mathbf{y}_B) = \frac{p(\mathbf{y}_A, \mathbf{y}_B)}{p(\mathbf{y}_B)} \tag{2.2.4}$$

$$p(\mathbf{y}_A | \mathbf{y}_B) = \frac{p(\mathbf{y}_A) p(\mathbf{y}_B | \mathbf{y}_A)}{p(\mathbf{y}_B)}, \tag{2.2.5}$$

where $p(\mathbf{y}_A)$ is our prior and $p(\mathbf{y}_B) > 0$ as it is futile to condition on an impossible event [16]. Also, using conditional probabilities we can obtain Bayes' theorem, eq. (2.2.5).

2.2.2 Bayesian Regression

Weight-space View - Regression tasks can be prone to noise, uncertainty and overfitting. As an example for linear regression from this point onwards, in order to try and predict the saturation time for a particular metabolite in buffer, such as Glucose, we would need to fit a linear model to our data, minimize the loss, find the correct parameters and then predict our results. During each step problems could arise which would negatively affect our model. The Bayesian model for linear regression aims to solve these problems and provide a

more intuitive model. The weight-space view for Bayesian linear regression allows for simple implementations and interpretability [16] but narrow flexibility since if there is no linear relationship the model will output inaccurate or wrong predictions.

The Bayesian standard linear regression model can be written as as $y_i = f(\mathbf{x}_i) + \varepsilon(\mathbf{x}_i)$ where f is the function value and ϵ is some Gaussian distribution with zero mean and variance σ_n^2 as noise, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ [16]. Also, the models noise assumption gives rise to the likelihood which is the probability density of the observations given the parameters. We then let $\hat{y} = \hat{w}_0 + \hat{w}_1 x_n$ be our prediction for y based on the n^{th} value of x. With standard linear regression, y was estimated as a single value, but with Bayesian linear regression it will be drawn from a probability distribution. Inference in the Bayesian linear model is based on the posterior distribution over the weights. This can be achieved using the standard Bayes' theorem

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{w}, \mathbf{X})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \tag{2.2.6}$$

where $\mathbf{X}$ is a vector of the transpose of elements in $\mathbf{x}_n$ and $p(\mathbf{w})$ is the prior probability density function of the parameters. We can capture all information about the parameters by combining the likelihood and prior. The denominator can be expanded as the marginal likelihood as

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \sigma^2) = \frac{p(\mathbf{y}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w})}{\int p(\mathbf{y}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w})d\mathbf{w}}. \tag{2.2.7}$$

Since we want to make predictions for our regression problem, we will need to take an expectation with respect to the posterior density of the weights. Therefore, to make a prediction y_* at a point $\mathbf{x}_*$ we can use equation (2.2.9). An interesting point about this is that we can also make predictions using probabilities such as our prediction y_* becomes $y_* < t$ for example.

$$p(y_*|\mathbf{x}_*, \mathbf{w}, \sigma^2) = \int p(y_*|\mathbf{x}_*, \mathbf{w}, \sigma^2)p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \sigma^2)d\mathbf{w} \tag{2.2.8}$$

$$= \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^\top A^{-1}\mathbf{x}_*), \tag{2.2.9}$$

where A^{-1} is a covariance matrix. eq. (2.2.9) can be derived with help by using the posterior in eq. (2.2.6) by writing only the terms from the likelihood and prior and completing the square [16]

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \propto \exp(-\frac{1}{2\sigma_n^2}(\mathbf{y} - \mathbf{X}^\top\mathbf{w})^\top(\mathbf{y} - \mathbf{X}^\top\mathbf{w}))\exp(-\frac{1}{2}\mathbf{w}^\top\textstyle\sum_p^{-1}\mathbf{w}) \tag{2.2.10}$$

$$\propto \exp(-\frac{1}{2}(\mathbf{w} - \bar{\mathbf{w}})^\top(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^\top + \textstyle\sum_p^{-1})(\mathbf{w} - \bar{\mathbf{w}})) \tag{2.2.11}$$

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{w}} = \frac{1}{\sigma_n^2}A^{-1}\mathbf{X}\mathbf{y}, A^{-1}), \tag{2.2.12}$$

where $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\mathbf{X}\mathbf{X}^\top + \sum_p^{-1})^{-1}\mathbf{X}\mathbf{y}$ and $A = \sigma_n^{-2}\mathbf{X}\mathbf{X}^\top + \sum_p^{-1}$. An important point to note is that for any Gaussian posterior, like the one in the model presented for Bayesian linear regression, eq. (2.2.12), the mean of the posterior is also its mode which is called the *maximum a posteriori* (MAP) [16].

2.3 Gaussian Process Regression

A Gaussian process (GP) is a stochastic process where the joint of all variables is drawn from a multivariate normal distribution that can be of infinite dimensions. Gaussian processes are a form of nonparametric Bayesian machine learning in which we construct a prior distribution over functions, rather than over parameters. While most machine learning algorithms (excluding neural networks) can be generally split up into two areas regression and classification, GPs have been developed extensively to be able to handle both areas along with unsupervised learning [17], reinforcement learning [18] and even deep models; DGPs [19]. One of the main advantages for using GPs for machine learning is that GPs provide a nice way of expressing the prior function when it is hard to make assumptions about a function for the data.

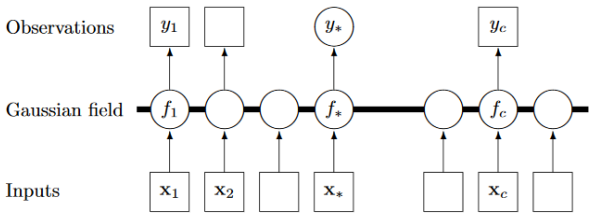


Figure 2.3.1: Graphical GP [20]

A non-mathematical and intuitive description of Gaussian processes can be described as playing a game of cards with two identical decks, where each card has a function, such as $f(x) = x^2$. A card is chosen and the value of the function on the card, at a particular point, is given. For example, the y value of an x value at point 3. Based on the answer, all cards in the other deck that do not match it, are thrown away. This can be thought of as the prior belief of functions. A new selection of cards, now the posterior belief of functions is presented. There are now two possible steps to take. Either, more questions are asked in order to update a confidence of what the function is, or a guess is made based on current data and beliefs. These final steps can be repeated infinitely many times. We can also describe a GP regression model graphically with Figure (2.3.1). Each input is mapped to an observation through a Gaussian field, represented as the thick horizontal line in the center. Each of these observations is conditionally independent of all other nodes given the corresponding latent variable, f_i [16].

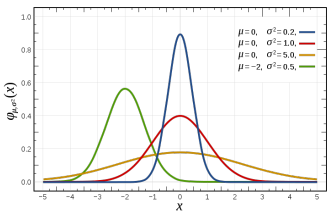
2.3.1 Normal Distributions

Before looking at the *multivariate normal distribution*, we will first go over the *univariate normal distribution*, or simply the normal distribution, so that we have the background to move forward as they are a crucial area in understanding the intuition behind GPs. By definition, a normal, or Gaussian, distribution is a distribution involving a one dimensional random variable. They are used because of their many features, such as being infinitely differentiable, that make them easy to work with and highly useful, such as only having two parameters; the mean and variance.

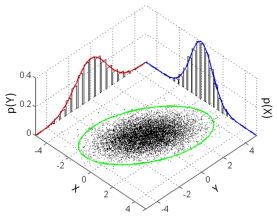
In a normal distribution, the probability density function (PDF) is defined as

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}, \tag{2.3.1}$$

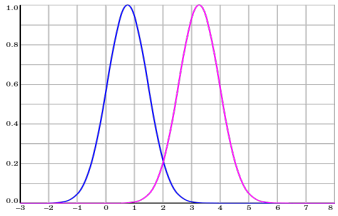
where $p(x)$ is the probability of x being generated or drawn from the distribution, μ is the mean, σ^2 is the variance and σ is the standard deviation. Changing the value of either μ and σ^2 we can adjust the position and height and length of the distribution Bell curve. In the Figure (2.3.2a), we can see that μ is always at the mode of the distribution, or in other words, it defines the mode. The distribution can also be described as "x is a random variable that has a normal, or Gaussian, distribution, with mean μ and variance σ^2, $x \sim \mathcal{N}(\mu, \sigma^2)$.



(a) Univariate normal distribution



(b) Multivariate normal distribution



(c) Unnormalized Gaussian RBF

Figure 2.3.2: Normal distributions, (a, b) and covariance function (c)

The main difference between a multivariate distribution and a univariate one is that a multivariate distribution works with random variables of multiple dimensions and has a covariance matrix. A multivariate normal distribution describes the probabilities of a continuous multidimensional random variable. With regards to multivariate Gaussians, these random variables follow a normal distribution, where each random variable in the distribution has

its own mean and variance. First, we have a vector of n random variables, $\mathbf{x} = (x_1, ..., x_n)$, that has a multivariate normal distribution, where each $x_n \sim \mathcal{N}(0, 1)$. We can denote the mean of x_n by μ_n and let $\boldsymbol{\mu} = (\mu_1, ..., \mu_n)$ be the $n \times 1$ vector of means. Furthermore, $\sum$ is an $n \times n$ matrix of covariances, $\sum = \text{Cov}(\mathbf{x}_i, \mathbf{x}_j)$. Then, the multivariate normal distribution has probability density function (PDF)

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^2 |\sum|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \sum^{-1} (\mathbf{x} - \boldsymbol{\mu})\right), \tag{2.3.2}$$

Notice we can also derive this PDF, *informally*, in a few steps from the univariate PDF [21]. In the context of GP regression, given a multivariate normal variable $\boldsymbol{y}$, we are interested in the conditional distribution of $\boldsymbol{y}_2$ given $\boldsymbol{y}_1$, which come from splitting up $\boldsymbol{y}$ into the two respective sub-vectors. Figure (2.3.2b) shows sample points from a multivariate normal distribution with $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\sum = \begin{bmatrix} 1 & 3/5 \\ 3/5 & 2 \end{bmatrix}$.

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \begin{pmatrix} \sum_{11} & \sum_{12} \\ \sum_{21} & \sum_{22} \end{pmatrix}\right) \tag{2.3.3}$$

Covariance Matrices - Covariance matrices, covariance functions or otherwise known as kernels, are crucial in Gaussian processes. A covariance function k is a function that maps pairs of inputs, x and x^*, into real values. The covariance function is very adaptable as its inputs can be multidimensional. In the context of Gaussian processes, these covariance functions correspond to the similarity of the inputs. We can construct the kernel $k(\mathbf{x}, \mathbf{x}')$ as $k_{ij} = k(x_i, x_j)$. There are many different kernel functions, but the most commonly used, and the one we shall use is the RBF kernel function, Figure (2.3.2c), otherwise known as the Radial Basis Function or squared exponential kernel, which is defined as either eq. (2.3.3) for a kernel with one-dimensional inputs, or eq. (2.3.4) for a kernel with multi-dimensional inputs. Note that we have two hyper-parameters present in the covariance function, where σ_f^2 is the variance and l is the length of the function, both of which can be varied to increase or reduce the correlation between data points.

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\Big(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2l^2}\Big) \tag{2.3.4}$$

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\Big(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{M} (\mathbf{x} - \mathbf{x}')\Big) \tag{2.3.5}$$

2.3.2 Function-space View

In a weight-space view, we describe uncertainty through a probability distribution over the weights. In this section, we look at the function-space view to deal with uncertainty with respect to functions. It is an different method of reaching the "same" results by instead considering inference directly in the function space. For this, we use a GP to describe this distribution over functions [16]. As a GP is specified by its mean and covariance, we can define this mean $m(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ as equations (2.3.6) and (2.3.7) with a GP as (2.3.8)

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.3.6}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \tag{2.3.7}$$

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.3.8}$$

Recall that a Gaussian process is a stochastic process with a collection of random variables, any finite number of which have a joint normal distribution. Given these random variables implies a marginalization property which specifies that upon examination of a larger set of variables does not change the distribution of the smaller set, as $(y_1, y_2) \sim \mathcal{N}(\boldsymbol{\mu}, \sum)$, then it must also specify $y_1 \sim \mathcal{N}(\mu_1, \sum_{11})$, where $\sum_{11}$ is the relevant sub-matrix of $\sum$. We can demonstrate a GP using the Bayesian linear regression model presented earlier $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$ with prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sum_p)$, where our mean is 0 and $\boldsymbol{\phi}(\mathbf{x})$ is a function which maps a D-dimensional input vector $\mathbf{x}$ into an N dimensional feature space, with

$$\mathbb{E}[f(\mathbf{x})] = \boldsymbol{\phi}(\mathbf{x})^\top \mathbb{E}[\mathbf{w}] = 0, \tag{2.3.9}$$

$$\mathbb{E}[f(\mathbf{x}) f(\mathbf{x}')] = \boldsymbol{\phi}(\mathbf{x})^\top \mathbb{E}[\mathbf{w}\mathbf{w}^\top] \boldsymbol{\phi}(\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \sum_p \boldsymbol{\phi}(\mathbf{x}') \tag{2.3.10}$$

Note that in this dissertation we will be working with the squared exponential covariance function. Hence we have $\text{cov}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k(\mathbf{x}_p, \mathbf{x}_q) = \exp\big(-\frac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2\big)$. In order to draw samples from the distribution of functions evaluated at any number of points we consider a test number of n input points, $\mathbf{X}_*$, and write out the corresponding covariance matrix elementwise [16]. We can now generate a random Gaussian vector with this covariance matrix, $\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_*, \mathbf{X}_*))$, and plot the outputs as a function of the inputs.

2.3.3 Regression & Prediction

We now continue our understanding of Gaussian process regression by bringing together the material from previous sections and describing how our standard GP function, eq. (2.3.8), is used in regression and prediction for noisy observations. We have a dataset, $\mathcal{D}$, containing n training points, $\mathcal{D} = \{(\mathbf{x}_i, f_i)|i = 1, ..., n\}$. In a realistic environment, we most likely will not have access to function values nor noise-free observations, thus we must compensate for this in the form of $y = f(\mathbf{x}) + \epsilon$. Our covariance matrix then becomes $\text{cov}(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq}$ or $\text{cov}(\mathbf{y}) = K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I$ where I is a diagonal matrix. The joint distribution of the training points outputs, $\mathbf{y}$, and the test outputs, $\mathbf{f}_*$, according to the prior and noise, is

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right), \tag{2.3.11}$$

where $\mathbf{X}$ and $\mathbf{X}_*$ denote training and test points respectively. Note that we can also think of $K(\mathbf{X}, \mathbf{X}_*)$ as the transpose of $K(\mathbf{X}_*, \mathbf{X})$. To get the posterior distribution over functions, we need to restrict this joint distribution to contain only functions which agree with observed data points [16]. We can arrive at a predictive distribution equation by conditioning the joint Gaussian prior distribution on the observations for our noisy GP regression model, resulting in

$$p(\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*) \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) \text{ where} \tag{2.3.12}$$

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_*||\mathbf{X}, \mathbf{y}, \mathbf{X}_*] = K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1}\mathbf{y}, \tag{2.3.13}$$

$$\text{cov}(\mathbf{f}_*) = K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1} K(\mathbf{X}, \mathbf{X}_*). \tag{2.3.14}$$

The results are key equations for GP regression, where $\bar{\mathbf{f}}_*$ is the mean and $\text{cov}(\mathbf{f}_*)$ is the covariance function. Simpler notation can be used to describe the predictive distribution, where expressions involving the kernel can be made more compact, giving us eqs. (2.3.17) and (2.3.18). In the case that there is only one test point $\mathbf{x}_*$, we write $\mathbf{k}(\mathbf{x}_*) = \mathbf{k}_*$ which denotes the vector of covariances between the test point and the given n training points. Therefore, when we take the noise into consideration, we can finally write the predictive distribution as $p(\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*) \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*) + \sigma_n^2 I)$. Note that we use the Cholesky decomposition instead of directly inverting the matrix, $(K + \sigma_n^2 I)$, as it is more numerically stable and faster [16].

$$\bar{f}_* = \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{y} \tag{2.3.15}$$

$$\mathbb{V}[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{k}_*. \tag{2.3.16}$$

2.3.4 Model Selection & Optimization

Learning Hyper-Parameters - As Gaussian processes are fully probabilistic, it gives us the ability and advantage to select the covariance hyper-parameters directly from the training data. A hyper-parameter is a free parameter from a covariance function. In the case of this dissertation, we will be using the squared-exponential function as our covariance function. This covariance function has three hyper-parameters; length-scale l, signal variance σ_f^2 and noise-variance σ_n^2. The hyper-parameters grow in size based on our data and dimensionality of the covariance matrix. For example, the length-scale has a parameter along each dimension of the full covariance matrix. The values of our hyper-parameters can greatly affect the strength of our GP model as the value of predictions *can* be greatly affected based on how the hyper-parameters are tuned.

In a Bayesian setting, we would like to learn these hyper-parameters by placing a prior and computing a posterior given the training data. This however is not analytically tractable and good approximations are not easily achieved [16]. Therefore, a general technique for learning hyper-parameters is to compute the integrals over the parameters, for which we can use the marginal likelihood - the integral of the likelihood times the prior, where the prior is Gaussian, $\mathbf{f}|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, K)$ or eq. (2.3.20), and the likelihood is a factorized Gaussian $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$

[16]. The hyper-parameters are then set by maximizing the marginal likelihood using the partial derivatives of the marginal likelihood w.r.t. the hyper-parameters, eq. (2.3.22).

$$\log p(\mathbf{f}|\mathbf{X}) = -\frac{1}{2}\mathbf{f}^\top K^{-1}\mathbf{f} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{2.3.17}$$

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top (K + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{n}{2}\log 2\pi \tag{2.3.18}$$

$$\frac{\partial}{\partial \theta_j}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^\top K^{-1}\frac{\partial K}{\partial \theta_j}K^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(K^{-1}\frac{\partial K}{\partial \theta_j}) \tag{2.3.19}$$

$$= \frac{1}{2}\mathrm{tr}\Big((\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - K^{-1})\frac{\partial K}{\partial \theta_j}\Big), \text{ where } \boldsymbol{\alpha} = K^{-1}\mathbf{y} \tag{2.3.20}$$

In this project we use the *Limited-memory BFGS* algorithm (L-BFGS) [24] for hyper-parameter optimization.

Chapter 3

Implementation & Methodology

This chapter details the algorithms developed in this project. In order to produce a training dataset for our GP model, we must develop a pipeline of algorithms to clean the dataset and extract only what is necessary. For this, two algorithms are devised which extract active non-barrier experimental pixels and find their reaction start. We will look in depth at the algorithms presented and their implementations as they are necessary key stages in preparing the final datasets for the Gaussian process model. Lastly, the GP model, implementation, and optimization will be discussed.

3.1 Extracting Non-Barrier Pixels

Given the data described in Chapter 1, we must first remove non-experimental data, ie. data from barrier pixels, dead pixels, and control micro-wells. To start with, it is crucial to first remove all pixels which are not contained in any of the four micro-wells, as otherwise they will interfere with the data and will provide the GP with incorrect data. This stage is done in two steps. First, we classify dead, barrier and non-barrier pixels. Second, we classify pixels in the control and experimental micro-wells. Dead pixels are defined as pixels which are broken and non-reactive.

We develop a heuristic for classifying pixels is to visualize all the data and try to identify patterns within it. Figure (3.1.1) shows the first column of pixels from the CMOS chip plotted with their intensity values over time. This includes pixel coordinates from $(0, 0)$ to $(0, 15)$. It is simple to point out the pixels that are experimental from this plot as they have a constant intensity that lasts approximately 1500 timesteps before noise is introduced and the reaction starts. The initial huge spikes of noise are a result of the metabolite being added to either the serum, buffer or urine. After the reaction start, there is an exponential growth in pixel intensity before the reaction saturates. All other pixels plotted in Figure (3.1.1) are completely linear and remain that way at intensity values of 160 and lower or 210 and higher. The lower value pixels are the pixels in the control micro-wells and the higher value pixels are barrier pixels. From this, we say that any pixels that grow over time and have values between $160 - 210$ are most likely going to be pixels in experimental micro-wells.

Figure (3.1.2) shows a histogram for a 5.6mM Glucose test for timestep 0, along with the mean, represented as the black dotted line, and the normal distribution. Figure (3.1.3) is a visualization at timestep 0 of the CMOS chip and all four micro-wells including the barrier section. The bottom two micro-wells are the control micro-wells which contain water while the top two contain the Glucose in buffer, serum or urine. A completely black pixel is also visible in the bottom left control micro-well - this indicates it is dead, or broken. Also visible is the approximate 5×5 pattern each micro-well has. Based on the data and the visualizations, a simple rule was developed to determine which pixels are barrier pixels and non-barrier pixels.

Based on the figures and the data, we can see that any values that are less than the mean, beside the dead pixel at intensity < 100, are non-barrier pixels. Therefore a simple extraction rule for the non-barrier pixels is to classify non-barrier pixels as those which are less than the mean. However, there are still some pixels ($\sim 5\%$) that are non-barrier but have values slightly greater than the mean which would not be extracted per the rule. Also, the dead pixel would have been classified as non-barrier. Therefore, to correctly classify these pixels, we can instead look at pixel intensities whose values are within some range, σ above and below the mean. For this, we assume that 3 standard deviations is enough of a margin to correctly classify between barrier and non-barrier pixels.

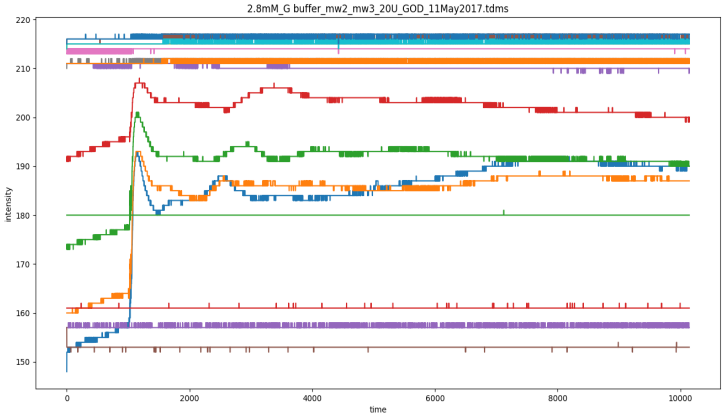


Figure 3.1.1: Pixels $(0, 0) - (0, 15)$ plotted over time

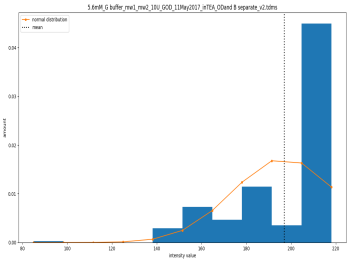


Figure 3.1.2: Histogram for Glucose in buffer at t_0 - (`x-axis` is the pixel intensity value, `y-axis` is the concentration)

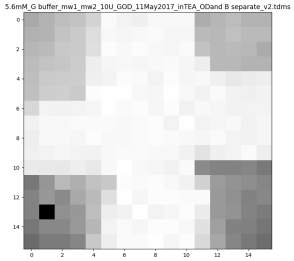


Figure 3.1.3: CMOS chip with micro-wells for Glucose in buffer at t_0

At any given timestep, there is a 16×16 matrix $\mathbf{M}$. The mean, standard deviation and variance across all 256 pixels are calculated. Where μ and σ are the mean and standard deviation of $\mathbf{M}$ respectively, and $\mathbf{M}_{ij}$ is the location of a pixel within the matrix. A pixel is identified as non-barrier if it has a value lower than $\mu + \sigma^2$ but greater than 3 times the standard deviation away from the mean. With reference to Figure (3.1.1), we extract only the pixels that are under the barrier pixel plots and that are not dead or broken. For added accuracy, each pixel then has its variance over time compared with some threshold θ to ensure it is a reactive pixel and not a false positive.

$$\text{non\_barrier\_pixel}_{ij} = (\mu + \sigma^2 > \mathbf{M}_{ij}) \wedge (\mathbf{M}_{ij} > \mu - 3\sigma) \tag{3.1.1}$$

Algorithm 1: Non-barrier pixel classifier

`data`: 2D array[$16 \times 16 \times t$]
`p`: pixel n
`M`: binary array mask of pixels at timestep $t = \bar{0}$
`non_barrier`: array of all non-barrier pixels
θ: variance threshold

for *timestep 0 to 1000* **do**
  `M`[p] $\leftarrow$ `data`[p] $\geq$ $mean$(`data`[p]) - $3std$(`data`[p]) **and** `data`[p] $<$ $mean$(`data`[p]) **or** `M`[p]
  for *every pixel* `p` *in* `M` **do**
    `c` $\leftarrow$ get coordinates of `M`[p]
  end
end
for *every pixel coordinate* `p` *in* `c` **do**
  if *over all timesteps* `p` $\geq \theta$ **then**
    `non_barrier` $\leftarrow$ `p`
  end
end

We now have an array of the coordinates of non-barrier pixels. It is necessary to further filter the array so that only pixel coordinates that are from experimental micro-wells are within it. As previously mentioned in our dataset, the control micro-wells are always the bottom two. Therefore we can exploit this and simply remove all pixels of coordinates that are higher than the midway point, in our case, $(8, 0)$.

3.2 Estimating Reaction Start

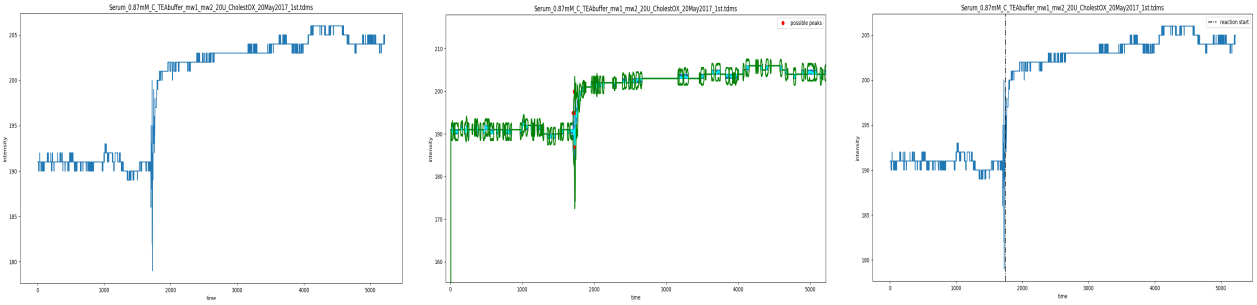
Now that we are left with an array of the coordinates from a given experiment that contains only non-barrier pixels, we need to automatically determine the start of each pixels' reaction. As stated in the previous section, when looking at any pixel reaction over time, it starts off relatively constant before a large spike of noise is introduced, to which the reaction start follows for an exponential growth period over n timesteps before reaching saturation. This was standard behaviour observed in a large number of pixels tested. In others, there were multiple isolated spikes, or peaks, **before** the reaction start peak is reached. There are two possible reasons for peaks of noise before the reaction start peak:

1. Metabolite being applied to the buffer/serum/urine/etc..., creating a human shadow over the pixel obstructing the LED above it and registering noisy intensity values for a brief period of time
2. Metabolite being applied to the buffer/serum/urine/etc..., and the natural reaction commencing

Therefore our goal is to within 100 timesteps (10Hz, 100 timesteps equals 10 seconds), identify the start of a reaction so that all data prior to the reaction start can be excluded from the final data for the GP model. An algorithm was devised to identify the reaction start based on the standard score, or z-score, through a windowed moving mean. The standard score in statistics is the number of standard deviations by which the value, or in our case, pixel intensity, of a point x at t_n differs from the moving mean. The algorithm uses 3 main parameters; a window size which determines how smooth and adaptive the algorithm will be to changes in data; an "effect" parameter which specifies how much of an effect certain signals and peaks have on the algorithm; and the standard deviation, or threshold, that controls how many standard deviations from the window a data point needs to be, to be classified as a peak. If the point x differs by more than some standard deviation σ a peak will be detected. This is an online algorithm since it only uses past data. We also check the total mean and standard deviation from the start, timestep 0, to the current point x. With this we have two mean values, a moving one and a continually updating one, as well as two standard deviation values. The reasoning for combining these two algorithms is to increase robustness.

When a peak is detected its timestep is recorded as the start of the reaction. If the algorithm finds a new reaction start, the previous value is discarded and the newer one is labelled as the reaction start. Usually, two different reaction start timesteps are returned from the algorithm, one from the moving mean and one from the total mean. The reaction start that is within 100 timesteps from the difference of both values is the final reaction start value chosen and used for the pixel. An example of the algorithm can be seen in Figure (3.2.1). In Figure

(3.2.1a) is the plot over time for a specific pixel from an experiment with cholesterol in buffer. Figure (3.2.1b) shows the end results of the algorithm, with added standard deviation peaks, a mean and identified signals/peaks as red dots. Lastly, Figure (3.2.1c) shows the output of the algorithm having marked the starting point of the reaction.



(a) Pixel plot over time for Cholesterol in serum

(b) Pixel plot over time during peak and start detection

(c) Pixel plot over time and reaction start

Figure 3.2.1: Reaction start algorithm

3.3 Metabolite Concentration Prediction with Gaussian Process Regression

In this section we present our GP regression model for predicting metabolite concentration in time series. We use the theoretical background introduced in Chapter 2 along with the GPy [25] library to create our model. In order to achieve this, we input into the model. We calculate the rate of change from the reaction start to some time, $t_s + d$, to use as input. This will make up our final dataset which will consist of a CSV file with pixel coordinates, experiment file name, concentration of metabolite used in the experiment, and rates of change for varying d.

3.3.1 Rates of Change

Combining the two algorithms presented in the previous chapter results in an array of the coordinates of experimental pixels along with the timestep at which their reaction starts. The last step needed to prepare the dataset for the GP regression model is to calculate individual rates of change for each correct pixel. Rates of change at 13 different intervals were calculated from the reaction start point, t_s. These 13 rates of change were calculated at $t_s + d$, where $d \in \{10, 20, 30, 40, 50, 75, 100, 125, 150, 200, 250, 300, 500\}$, using

$$\text{Rate of change} = \frac{y_2 - y_1}{t_2 - t_1}, \tag{3.3.1}$$

where y_1 was the median historical pixel intensity up until the reaction start point, as the noise-free baseline intensity, and y_2 is the intensity at time t_2.

3.3.2 Gaussian Process Regression Model

The Gaussian process regression model was built using the GPy framework [25] developed by the SheffieldML group. GPy is a free open-source library providing a strong and robust framework for Gaussian processes written in Python and using `numpy` for calculations. It is actively developed and used in both academia research and industry. To date, it is the most complete library for working with Gaussian processes including a variety of different kernels and optimization algorithms.

Given our preprocessed dataset of metabolite concentration measurements for experimental pixels and their respective rates of change from the reaction start point, $\mathcal{D} = \{(x_i, y_i)\}$, where $\mathbf{x}_i$ is a vector of rates of change and y_i is a concentration, the task is to now jointly learn and predict the assignment of a concentration given a new unlabelled feature vector $\mathbf{x}_*$. There are a total of 10 datasets containing $350 - 1000$ samples. There is also one extra dataset that was created by manually filtering falsely included pixels using the non-barrier pixel

classification algorithm. Each metabolite (cholesterol, choline, sarcosine, xanthine, glucose) has 2 datasets, one for an experiment in serum and one in buffer.

A GP regression model was created for each of these datasets using Leave-one-out-Cross-Validation (LOOCV) to train and test. Rather than using LOOCV on individual x and y values, the dataset was split up into folds based on the file name of the experiment they originate from so as to not train and test on data from the same experiment. Also, due to the fact that the concentrations of metabolites in experiments are non-negative and range across orders of magnitude, (mM - μM), we transformed y into the natural log domain, $y = \text{np.log}(y)$.

The implemented GP model uses an RBF (squared-exponential) kernel with a starting variance and length-scale of 1 and 2 respectively. We then optimize the kernel hyper-parameters through the use of the L-BFGS algorithm and optimize the model for 10 iterations using the log likelihood and the log likelihood gradient. Note that the initial noise of the model is set to 1 and is optimized alongside the hyper-parameters. It is also possible to set a prior mean function as this can improve the model predictions in uncertain areas with no data. This was set dependent on the model so it often varied. This is demonstrated in the code snippet below. Notice that because we are already in the log domain, we do not need to explicitly specify our mean prior in the log domain again.

```
...
mf = GPy.core.Mapping(1, 1)
mf.f = lambda x: n # set prior mean as log(n)
mf.update_gradients = lambda a, b: None
...
kernel = GPy.kern.RBF(input_dim = 1, variance = 1, lengthscale = 2) #
    ↪ create kernel and optimize hyper-parameters
model = GPy.models.GPRegression(x, y, kernel, mean_function = mf) #
    ↪ create GPR model with noise
model.optimize() # optimize model
...
model.predict(x_) # make predictions
...
```

LOOCV was used to evaluate the model, and so the model was trained repeatedly on $n - 1$ data where n is the number of total experimental files where the pixels originate from in the dataset. Once the model was trained, the testing was done using the remaining data, $\mathbf{x}_*$. Figure (3.3.1) shows an example of some of the data the model was trained on, along with samples from it drawn, Figure (3.3.2), with the y-axis in log-space.

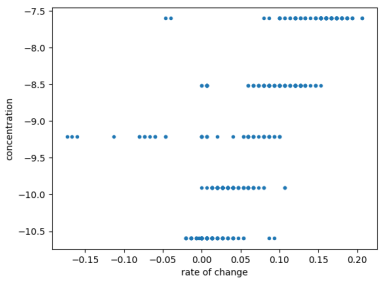


Figure 3.3.1: Rate of change data at $t_s + d150$ for choline in serum

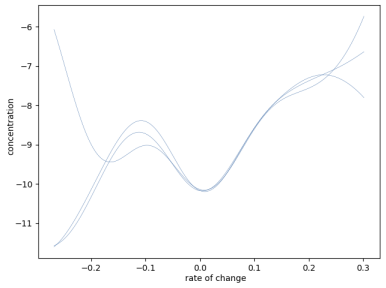


Figure 3.3.2: 3 samples from data

Chapter 4

Evaluation & Results

This chapter will look at the results of both algorithms presented in Chapter 3 for extracting non-barrier and experimental pixels and identifying the reaction start of a pixel, as well as the GP model.

4.1 Pixel Extraction

To quantitatively evaluate the pixel classification accuracy, ground truth labels were provided by manually marking barrier and non-barrier pixels. The ground truth consisted of an array mask of pixels. The array stores a 1 if the pixel in that position is non-barrier, a 0 if it is a barrier pixel, and "NaN" if the pixel is dead or broken. This array is referred to as `g_truth` and was constructed for 10 different experiment datasets. For each timestep, the algorithm used the extracted coordinates as pixels of value 1, for `g_truth`, pixels lower than the algorithm as "NaN" and 0 for all others. The `non_barrier` array was then compared to `g_truth` at each timestep to test how many pixels were correctly identified. Only the non-barrier pixels were reported as this is what we are interested in. The overall mean accuracy for all 10 experiments was 91.2%. As stated, we exploited the dataset property that the control micro-wells were in the lower bottom corners of the CMOS chip, and simply remove those pixel coordinates from the extracted data, and compare it with the `g_truth` array. The accuracy for this was $> 99\%$.

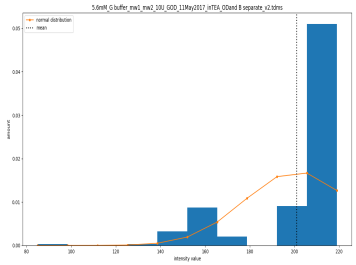


Figure 4.1.1: Histogram for Glucose in buffer at t_{6000}

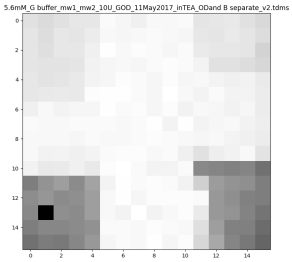


Figure 4.1.2: Pixel intensities for Glucose in buffer at t_{6000}

An overall mean accuracy of 91.2% was slightly under what was deemed acceptable as it meant there were still some experiments where $8 - 12$ pixels that were extracted were barrier pixels. This would lead to further problems down in the preprocessing pipeline. Examining the accuracy of each experiment individually after every 1000 timesteps, it was clear that as time went on, the accuracy decreased. This can be seen by comparing the plot and CMOS chip visualization from Figures (3.1.2) and (3.1.3) against Figures (4.1.1) and (4.1.2). At timestep 0 the mean overall accuracy is approximately $94\% - 96\%$. At timestep 4000 the mean overall accuracy decreased to 64%. This happens because over time the difference in intensity between barrier and non-barrier pixels was much harder to distinguish given the mean intensity values, especially as the reaction reaches an end and saturates completely. Again, this can be seen in Figures (4.1.1) and (4.1.2) where the mean point is at a value similar to that of the barrier pixels and the intensities between non-barrier experimental and barrier are very similar, respectively.

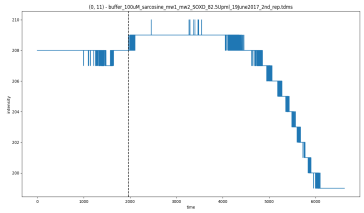
In order to resolve the decreasing accuracy of our rule over time, we instead look at only the first 1000 timesteps. During the proposed first 1000 timesteps, our initial rule has an accuracy ranging from $96\% - 99\%$. We can therefore use the coordinates of those identified non-barrier cells as our final non-barrier pixels, given that they are classified as non-barrier the majority of the time over the 1000 timesteps. We can also combine all of the non-barrier identified pixels from the first 1000 timesteps and remove all duplicates, so only unique values are kept. Given the new restriction on the algorithm and combining it with the removal of the control micro-well pixels, we can once again test the accuracy using our ground truth arrays. The results of this can be seen in the confusion matrices below which show the amount of correctly identified non-barrier, experimental pixels across all 10 ground truth experiment arrays.

		Ground truth	
		Positive	Negative
Pixel classifier	Positive	417 (TP)	16 (FP)
	Negative	36 (FN)	811 (TN)

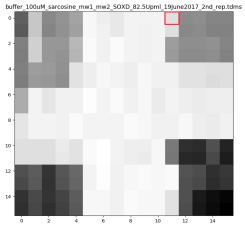
		Ground truth	
		Positive	Negative
Pixel classifier	Positive	0.92	0.02
	Negative	0.08	0.98

Note that the total number of pixels presented add to 1280 and not 2560 as half of the data, being the control and barrier pixels, were automatically removed. From the confusion matrix we can also calculate the *precision* and *misclassification rate* of the algorithm. The precision states how often the algorithm is correctly identifying non-barrier pixels and can be calculated from $\frac{TP}{TP+FP} = \frac{417}{417+16} = 0.96$. On the other hand, the misclassification rate is $\frac{FP+FN}{total} = \frac{16+36}{1280} = 0.04$. Also worth mentioning is the F1 score of the model which measures the precision and recall together, $\frac{2TP}{(2TP+FP+FN)} = 0.94$.

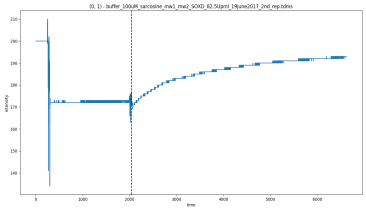
It is worth briefly comparing a true positive pixel with a false positive one. From Figure (4.1.3a), we can see that the start intensity value of the FP pixel is at 208, which is slightly higher than most other pixel intensities from Figure (4.1.3b), but was most likely chosen due to the fact that there are a few pixels in this dataset that are true positive and start off also at very high values, Figure (4.1.3c). This could be due to some calibration errors prior to the experiment start because after ~ 300 timesteps the intensity value falls back down to a standard of range of 170 ± 10. Given that the FP pixel also has this strange starting intensity similar to that of the TP pixel, which is still lower than other barrier pixels, and will also have some variance over time, this was taken to be a TP pixel. Note that at the end of the experiment, the FP pixel has an intensity value almost identical to that of the TP pixels. A quick fix for these type of initial potential calibration errors may be to discard the first 300 - 500 timesteps.



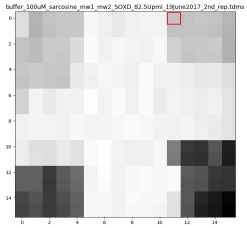
(a) FP pixel



(b) CMOS start



(c) TP pixel



(d) CMOS end

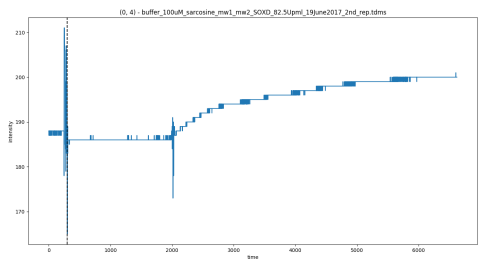
Figure 4.1.3: TP and FP comparison alongside the other individual pixel cells. The FP pixel (0, 11) is highlighted in red

4.2 Reaction Start Estimator

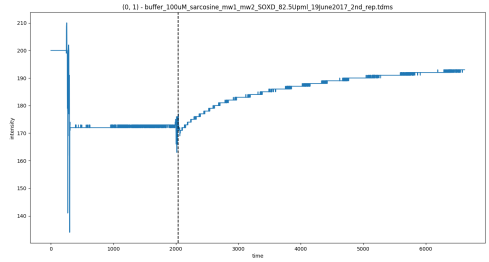
A large part of the evaluation for the reaction start estimator was done manually. The reason for this is that after the entire pre-processing pipeline, a random experiment file was chosen, "*serum_bufer.csv*", to be manually evaluated and checked for errors so that a new false positive free dataset, "*serum_buffer_TP.csv*", could be created so as to compare it with the original dataset after the Gaussian process regression. During this process, a total of 1000 extracted pixels were looked at, along with their plots, reaction start times, and rates of change.

Upon evaluation of each pixel, a time plot similar to Figure (4.2.1a) was presented and its detected start timestep would be printed out. If the printed reaction start time, also marked in the plot, was ± 100 timesteps from the ground truth, then the a score of true positive, or TP, was given to the pixel. It is worth noting that the variance of ± 100 timesteps was used due to the fact that if a discrete value is given as the ground truth, with no room for precision errors, it would result in many more false positives, or FPs, in the final dataset. It would be much harder to obtain the exact ground truth value predicted because small precision errors of even ± 1 are very difficult to correct. An example of this is having a ground truth of 1724 while the algorithm estimates the reaction start at 1726. While still accurate, a direct comparison would not work. Overall, 4.2% of pixels were FP, meaning they had their reaction start identified as the prior noise before the reaction start, or too far after the true reaction had started.

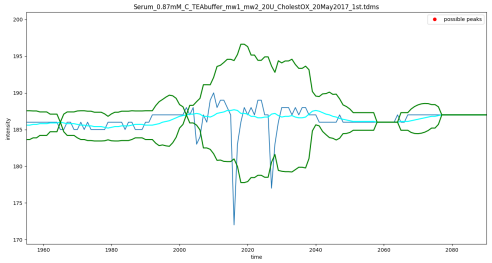
To further evaluate the algorithm, it is interesting to look at a couple of results in more detail. Figure (4.2.1a) shows a pixel which was correctly classified by the pixel extraction algorithm but had its reaction start point estimated at the incorrect timestep, while the pixel plot in Figure (4.2.1b) has an accurate estimate point for the reaction start. We will refer to the "bad" result as a false positive reaction (FP) start. Upon further examination, its prior noise is identified similar to a pixel with a true positive (TP) reaction start, although the peak at the reaction start time, was ignored. A pattern that emerged when manually evaluating these results for this dataset, is that in a large amount of the FP reaction start pixels, within the window of the reaction start, there are only significant negative peaks and no positive peaks, Figure (4.2.1c). Similarly, in TP reaction start pixels, the negative peaks within the reaction start window are smaller in respect and follow one or more positive peaks, Figure (4.2.1d).



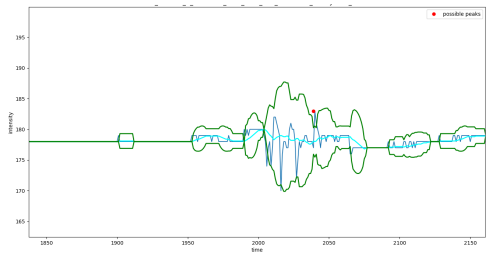
(a) FP reaction start



(b) TP reaction start



(c) FP reaction start within true reaction start window showing mean and standard deviation



(d) TP reaction start within true reaction start window showing mean and standard deviation

Figure 4.2.1: FP and TP reaction starts

Another interesting result is shown in Figure (4.2.2), where there was potential for a FP reaction start estimate, due to the large and sudden increase in pixel intensity for over 1000 timesteps, yet the reaction start time was accurately estimated.

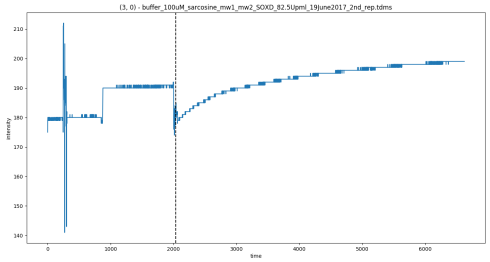
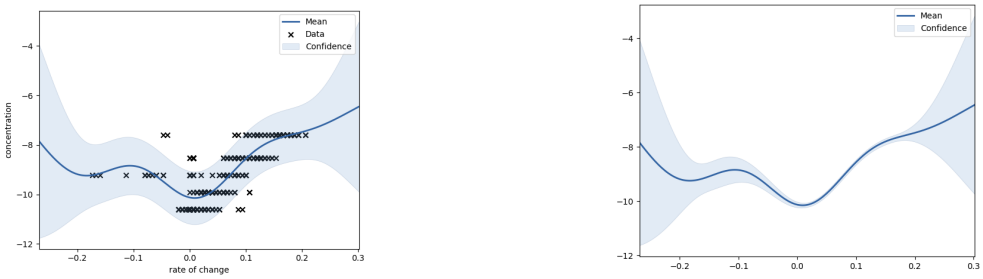


Figure 4.2.2: TP reaction start

4.3 Concentration Prediction

To empirically evaluate the implemented GP regression model, different evaluation measures are used in this chapter and the model is rigorously tested under different properties, such as alternating features and increasing the input vector dimensionality. As was also briefly mentioned in Chapter 4, a key aspect of the GP model was using LOOCV to evaluate. This was to make sure that data from the same experimental set were not cross contaminated.

We first show general plots of the Gaussian process regression. Figure (4.3.1a) shows a GP model trained on "*choline_serum*" using d150 as the rate of change for the input data where the training points, mean, and prediction interval for the data are visible. The confidence interval for the mean is shown in Figure (4.3.1b).



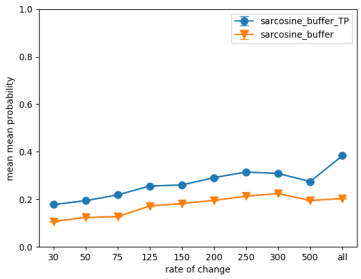
(a) Gaussian process for choline_serum (d150) (b) Confidence interval for choline_serum (d150)

Figure 4.3.1: Gaussian process regression model

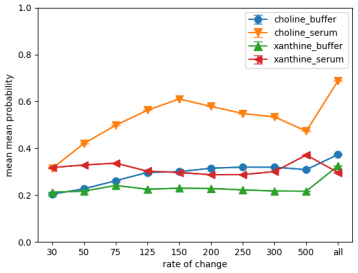
Using LOOCV, models were trained and tested using a single rate of change as the input feature, for all rates of change, before finally using all rates of change, resulting in a 9-dimensional feature input space. Each model is optimized 6 different times. This was decided based on the fact that the initial optimization iteration value was set at 10, but all models reached ideal optimized hyper-parameters by the 5^{th} or 6^{th} iteration. For any given dataset, there were 10 models trained and tested (9 different rates of change and *all*). The mean posterior probability (MPP) was calculated for each model and averaged over all folds, shown in eq. (4.3.1), to measure model performance. The standard deviation in error was also calculated. The raw results can be seen in Table (4.1), where the standard deviation is in brackets. Metabolites have either a (B) or (S) in their name, indicating if the experiment was done in buffer or serum, respectively. Also, as was previously mentioned in the evaluation for the reaction start estimator, a false positive free dataset was created during evaluation. This dataset is labelled "TP" in the table and contains no falsely classified experimental pixels and accurate reaction start estimates.

$$\frac{1}{M}\sum_{i} p(y_i) \tag{4.3.1}$$

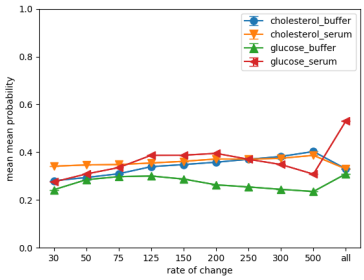
From the data in Table (4.1), we can plot the mean posterior probability at every rate of change feature input to get an idea of the best rate of change value for predicting metabolite concentration. Figure (4.3.2a) shows the mean posterior probability for "*sarcosine_buffer*" and "*sarcosine_buffer_TP*". We can see that there is a significant difference between the standard dataset and the manually cleaned dataset. Based on the data, we also see that the best rate of change for prediction is at d250 for "*sarcosine_buffer_TP*" and d300 for "*sarcosine_buffer*". For all metabolites the best rate of change for prediction tends to occur after d150, which is a 15 second wait when applying this to a real time environment such as a doctors office. Another noteworthy metabolite turns out to be "*choline_serum*", Figure (4.3.2b), which has the highest mean posterior probability and peaks at d150. Choline levels in serum are an emerging biomarker to detect early onset of troponin-positive cardiac ischemia [2]. We can also see that all metabolites in serum produced better results with the GP model. Figure (4.3.3) shows this, where each bar is the average mean posterior probabilities for each metabolite from Table (4.1). This may suggest that testing for metabolite concentrations in serum can provide better results.



(a) *sarcosine_buffer* & *sarcosine_buffer_TP*



(b) *choline_{buffer, serum}* & *xanthine_{buffer, serum}*



(c) *cholesterol_{buffer, serum}* & *glucose_{buffer, serum}*

Figure 4.3.2: Mean posterior probability plots

Metabolite	**d30**	**d50**	**d75**	**d125**	**d150**
Sarcosine (B) TP	0.177 (.031)	0.194 (.044)	0.218 (.053)	0.255 (.049)	0.259 (.053)
Sarcosine (B)	0.106 (.013)	0.123 (.019)	0.127 (.021)	0.172 (.027)	0.182 (.031)
Glucose (B)	0.242 (.020)	0.284 (.040)	0.297 (.040)	0.300 (.037)	0.287 (.033)
Glucose (S)	0.275 (.074)	0.308 (.073)	0.335 (.084)	0.386 (.128)	0.387 (.123)
Choline (B)	0.203 (.021)	0.226 (.026)	0.261 (.036)	0.296 (.045)	0.299 (.048)
Choline (S)	0.313 (.053)	0.419 (.096)	0.497 (.140)	0.562 (.151)	0.609 (.148)
Xanthine (B)	0.212 (.020)	0.216 (.032)	0.241 (.035)	0.225 (.029)	0.229 (.021)
Xanthine (S)	0.317 (.076)	0.328 (.087)	0.336 (.079)	0.302 (.059)	0.296 (.055)
Cholesterol (B)	0.280 (.043)	0.298 (.057)	0.309 (.052)	0.339 (.068)	0.348 (.071)
Cholesterol (S)	0.341 (.000)	0.346 (.054)	0.348 (.068)	0.354 (.087)	0.361 (.087)
Metabolite	**d200**	**d250**	**d300**	**d500**	**{all}**
Sarcosine (B) TP	0.291 (.068)	0.314 (.075)	0.308 (.081)	0.274 (.056)	0.393 (.180)
Sarcosine (B)	0.195 (.037)	0.213 (.041)	0.224 (.043)	0.195 (.035)	0.203 (.052)
Glucose (B)	0.263 (.026)	0.254 (.023)	0.244 (.023)	0.235 (.023)	0.309 (.069)
Glucose (S)	0.394 (.087)	0.370 (.118)	0.348 (.106)	0.308 (.055)	0.530 (.130)
Choline (B)	0.314 (.054)	0.319 (.057)	0.319 (.058)	0.308 (.055)	0.373 (.076)
Choline (S)	0.578 (.194)	0.548 (.196)	0.533 (.185)	0.479 (.112)	0.689 (.176)
Xanthine (B)	0.228 (.021)	0.222 (.020)	0.217 (.020)	0.216 (.028)	0.325 (.084)
Xanthine (S)	0.287 (.062)	0.287 (.066)	0.300 (.079)	0.371 (.099)	0.296 (.016)
Cholesterol (B)	0.357 (.077)	0.370 (.080)	0.382 (.083)	0.402 (.081)	0.330 (.081)
Cholesterol (S)	0.371 (.073)	0.370 (.064)	0.374 (.050)	0.386 (.041)	0.329 (.081)

Table 4.1: Mean posterior probability

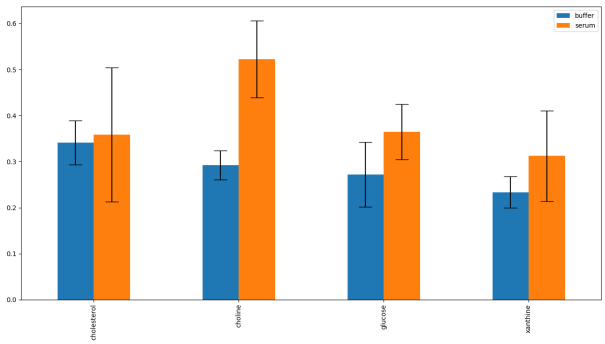
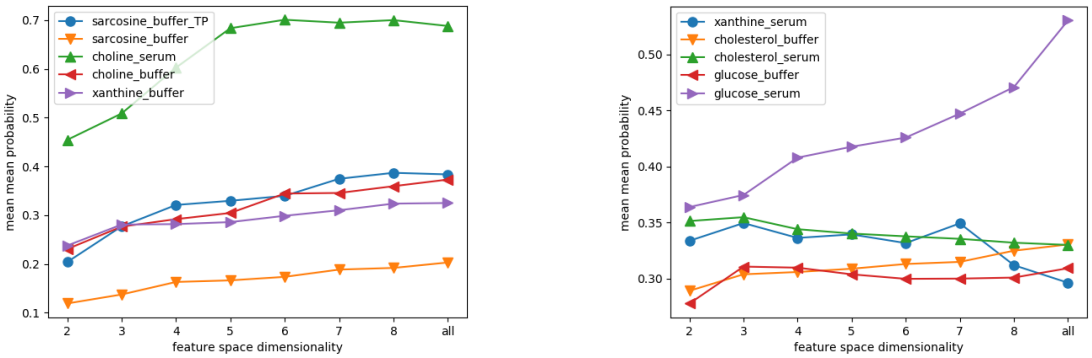


Figure 4.3.3: Serum vs Buffer

Giving all the rates of change as the feature input space produced substantially better results in almost all the metabolites tested. A few, such as cholesterol and glucose in buffer had either no difference or produced worse results. This suggests that for certain metabolites it is better to use more than 1 rate of change as an input feature, while for specific others, a new approach may need to be taken; either further increasing the input features (more rates of change), or only using specific rates of change which individually have higher probabilities. Models were re-trained with increasing feature inputs, from 2 to 8, as we have already tested all metabolites on 1 and all, to find the optimal amount of features needed to achieve higher performance. Figures (4.3.4a, b) both show the mean posterior probability for the feature space dimensionality. We can see that for almost all metabolites, either in serum or buffer, the higher the feature space, the higher the mean posterior probability is. Some peak before, such as with 7 for "*xanthine_serum*" or 8 for "*choline_serum*", while others appear to get worse, for example, "*cholesterol_serum*".

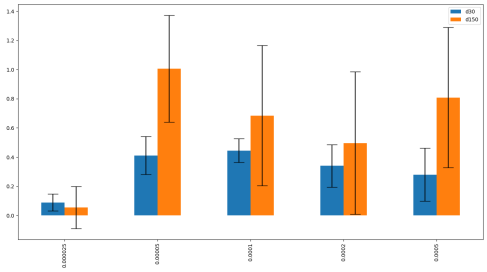


(a) *sarcosine_{buffer, buffer_TP}, choline_{S, B}* & *xanthine_buffer* (b) *xanthine_serum, cholesterol_{S, B}* & *glucose_{S, B}*

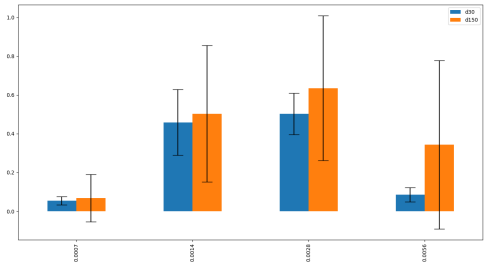
Figure 4.3.4: Mean posterior probability over input feature spaces

In order to further evaluate the GP model it is important to look at the features themselves and how they affect the models prediction. For this, we briefly looked at the mean posterior probability for choline and glucose at two rates of change, d30 and d150, for specific concentrations. This gives us an idea which concentrations of metabolites have a higher mean posterior probability value making them more reliable for regression and prediction. Figure (4.3.5a, b) shows this. Both metabolites appear to have a normal distribution over the concentration mean posterior probabilities, with the concentrations that have the highest mean posterior probability being nearer the center, or the mode. For choline, it appears that a smaller concentration of $5.e-05$ has a higher mean posterior probability, whereas for glucose it is 0.0028.

As stated, the noise for the GP model is manually set to 1 at the start, and is later learnt and optimized with L-BFGS along with the kernel hyper-parameters. To succinctly demonstrate the effects of changing this noise parameter value manually, LOOCV was used to evaluate different set values. The Table (4.2), shows the results of manually setting the noise parameter on a small set of different metabolites and rates of change.



(a) Choline in serum MPP at concentrations {2.5e-05, 5.e-05, 0.0001, 0.0002, 0.0005}



(b) Glucose in serum MPP at concentrations {0.0007, 0.0014, 0.0028, 0.0056}

Figure 4.3.5: Mean posterior probability based on concentrations for d30 & d150

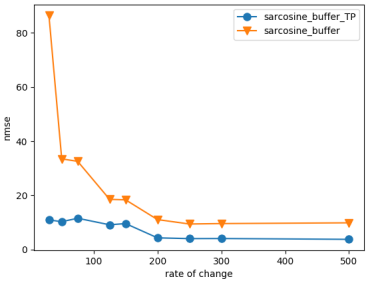
	0.001	**0.01**	**0.1**
Choline (S) - d150	1.922	1.256	0.836
Glucose (S) - d150	3.797	0.188	0.642
Xanthine (S) - d150	2.251	0.325	0.389

Table 4.2: Mean posterior probabilities acquired with manual noise values

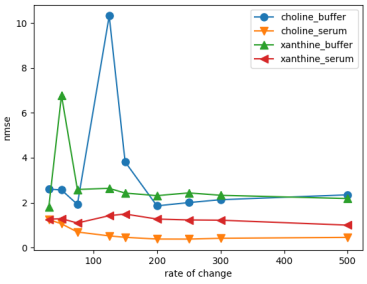
Also used as a performance measure was the mean squared error (MSE). However, when calculating the MSE, the results were very small, and thus the normalized mean squared error (NMSE) was used instead. This is shown in eq. (4.3.2). The results for this are shown in Table (4.3) and in Figure (4.3.5). From directly looking at the plots, in terms of performance, it is relatively similar to the mean posterior probability, with "*choline_serum*" performing the best once again. What is interesting is that others, such as "*glucose_serum*", seem to be performing closer to the levels of "*choline_serum*" than when compared with the MPP results.

$$NMSE = \frac{1}{N}\sum_i \frac{(y_i - \mu_i)^2}{\bar{y}\bar{\mu}}, \text{ where} \tag{4.3.2}$$

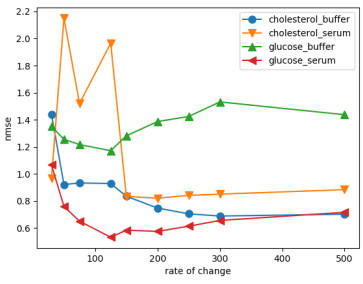
$$\bar{y} = \frac{1}{N}\sum_i y_i, \qquad \bar{\mu} = \frac{1}{N}\sum_i \mu_i \tag{4.3.3}$$



(a) *sarcosine_buffer & sarcosine_buffer_TP*



(b) *choline_{buffer, serum} & xanthine_{buffer, serum}*



(c) *cholesterol_{buffer, serum} & glucose_{buffer, serum}*

Figure 4.3.6: NMSE plots

Metabolite	**d30**	**d50**	**d75**	**d125**	**d150**	**d200**	**d250**	**d300**	**d500**
Sarcosine (B) TP	10.942	10.291	11.559	9.149	9.596	4.327	4.033	4.067	3.781
Sarcosine (B)	86.357	33.484	32.564	18.542	18.365	11.067	9.439	9.591	9.868
Glucose (B)	1.349	1.255	1.216	1.171	1.281	1.386	1.424	1.531	1.437
Glucose (S)	1.066	0.760	0.659	0.532	0.584	0.571	0.614	0.656	0.716
Choline (B)	2.602	2.563	1.922	10.317	3.816	1.857	2.002	2.129	2.348
Choline (S)	1.240	1.056	0.629	0.520	0.455	0.379	0.376	0.413	0.452
Xanthine (B)	1.793	6.776	2.588	2.637	2.427	2.312	2.434	2.328	2.182
Xanthine (S)	1.260	1.281	1.093	1.420	1.493	1.270	1.229	1.218	0.998
Cholesterol (B)	1.339	0.921	0.929	0.835	0.748	0.748	0.705	0.689	0.702
Cholesterol (S)	0.961	2.149	1.520	1.961	0.834	0.819	0.842	0.850	0.884

Table 4.3: NMSE

Chapter 5

Conclusion

The primary motivation for undertaking this project was to be able to successfully predict a metabolite concentration in time series data using the rate of change and the data captured by the CMOS-based device. This work was done with hopes that by using Gaussian processes alongside the CMOS-based device, doctors could test patients metabolite concentrations in serum, buffer, or urine for fast and easy disease diagnosis.

In this dissertation we have successfully implemented two data pre-processing algorithms to: (1) classify experimental metabolite pixels from the CMOS-based device data; and (2) estimate the reaction start of a metabolite in buffer and serum. We have also effectively created a Gaussian process regression model used for the prediction of metabolite concentrations given a reaction rate of change. We have shown the ability to correctly classify experimental metabolite pixels from the CMOS chip with a precision and F1 score of 0.96 and 0.94 respectively. We also show that it is possible to estimate the reaction start of a metabolite in buffer, serum, and urine as true positive 95.8% of the time. The Gaussian process models implemented demonstrate that metabolites have greater prediction accuracy when in serum than when in buffer and that the rate of change used, as well as the amount of input features (rates of change), can significantly affect the results. Furthermore, the outcomes from the Gaussian process evaluation section demonstrate that the overall, best performing metabolite for concentration prediction was choline in serum. Also, for the results demonstrated in Chapter 4, we can, for this project, state that in order to create Gaussian process regression models that best suits the metabolite data, both the input features, rates of change after the reaction start, and metabolite concentration need to be carefully looked at.

5.1 Future Work

While the results of this dissertation are successful, further work is needed to be done to advance the capabilities and accuracy of predicting metabolite concentrations based on rate of change. This is so the CMOS-based device can eventually become a self-contained medical device that doctors can use with patients to quickly and accurately discover metabolite concentrations for disease diagnosis and discovery. We explore a handful of these improvements in this section.

- **Improve the estimated reaction start point** - In the algorithm presented to estimate the reaction start in real time, 4.2% of the evaluated results were false positives. In order to improve the algorithm and increase its accuracy and decrease false positives, a solution worth considering is using a windowed median instead of mean, and replacing the standard deviation with a more robust measure of scale. An example is the median absolute deviation (MAD), which is the median of the absolute values of the differences between the data values and the overall median of the data. Decreasing the amount of false positives in this stage would result in far cleaner data for the GP as false positives create incorrect rate of change values which affect the accuracy of the GP if trained on.
- **Rate of change** - Currently, the rate of change is calculated by simply finding the slope of the curve at two points. This gives an approximate answer depending on how close the points are. In future, it may be interesting to instead compute the derivative of the curve to give the rate of change at any point as this will yield a more precise rate of change.
- **GP feature selection & hyper-parameter optimization** - From the evaluation chapter, we saw how much results could change based on the feature input space, hyper-parameter and noise optimization, and metabolite concentration used. Based on this information, future work involving carefully selecting and

extracting all of these parameters and features may be crucial to obtaining a higher standard of Gaussian process regression model, ultimately giving better predictions for metabolite concentrations.

- GP regression, classification, and more - Lastly, another interesting piece of future work may be to attempt and transform this regression problem into one of classification or using deep learning and deep Gaussian processes which can learn from complex data in a non-parametric fashion also and may provide greater insight into the metabolite data.

Appendix

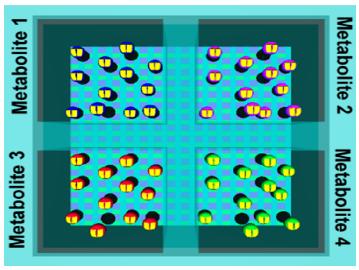


Figure A.1: Birds-eye view of the CMOS chip and 4 micro-wells

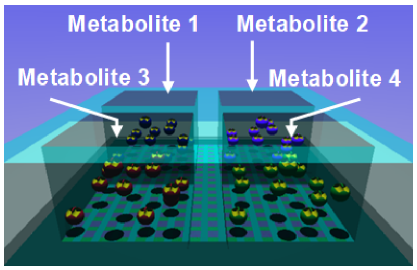


Figure A.2: Side view of the 4 micro-well areas

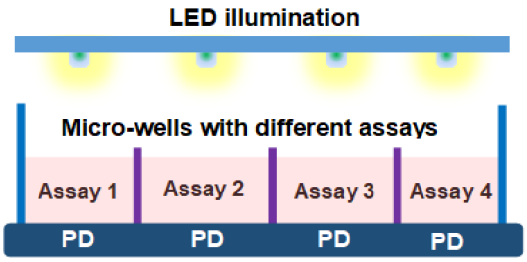


Figure A.3: Employed principle for metabolite sensing schematic design

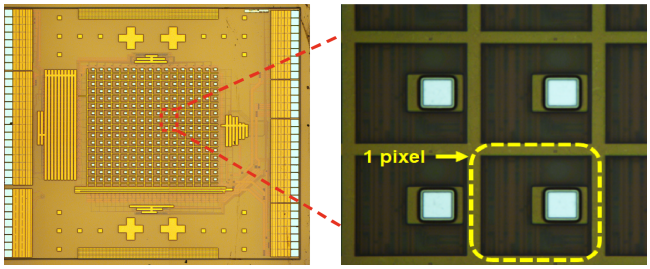


Figure A.4: CMOS chip and magnification of pixels

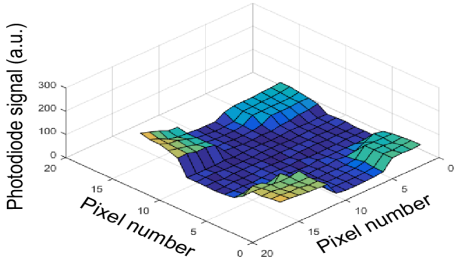


Figure A.5: Photo diode signal from micro-wells as a surface plot

Bibliography

[1] D. Neil, M. Segler, L. Guasch, M. Ahmed, D. Plumbley, M. Sellwood, and N. Brown, "Exploring deep recurrent models with reinforcement learning for molecule design," 2018.

[2] S. B. Patil, D. S. Dheeman, M. A. Al-Rawhani, S. Velugotla, B. Nagy, B. C. Cheah, J. P. Grant, C. Accarino, M. P. Barrett, and D. R. S. Cumming, "An integrated portable system for single chip simultaneous measurement of multiple disease associated metabolites," Electronics and Nanoscale Engineering, Wellcome Center for Molecular Parasitology, University of Glasgow, 2018.

[3] V. M. Asiago, L. Z. Alvarado, N. Shanaia, G. N. Gowd, K. Owusu-Sarfo, R. A. Ballas, and D. Rafter, "Early detection of recurrent breast cancer using metabolite profiling," 2010.

[4] J. L. Griffin, H. Atherton, J. Shockcor, and L. Atzori, "Metabolomics as a tool for cardiac research," 2011.

[5] J. L. Griffin and J. P. Shockcor, "Metabolic profiles of cancer cells," 2004.

[6] J. Sun, R. D. Beger, and L. K. Schnackenberg, "Metabolomics as a tool for personalizing medicine," 2012.

[7] N. Cernei, Z. Heger, J. Gumulec, O. Zitka, M. Masarik, P. Babula, T. Eckschlager, M. Stiborova, R. Kizek, and V. Adam, "Sarcosine as a potential prostate cancer biomarker," 2013.

[8] C. S. U. Noel Sturm, 2017.

[9] L. Pauling, A. B. Robinson, R. Teranishi, and P. Cary, "Quantitative analysis of urine vapor and breath by gas-liquid partition chromatography," 1971.

[10] C. Hu, M. A. Al-Rawhani, B. C. Cheah, S. Velugotla, and D. R. S. Cumming, "Hybrid dual mode sensor for simultaneous detection of two serum metabolites," 2017.

[11] D. Tominaga, K. Mori, and S. Aburatani, "Linear and nonlinear regression for combinatorial optimization problem of multiple transgenesis," 2016.

[12] C. K. Williams, "Prediction with gaussian processes: From linear regression to linear prediction and beyond," 1997.

[13] J. C.-F. Gauss, *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum.*

[14] A.-M. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes.*

[15] S. Rogers and M. Girolami, *A First Course in Machine Learning.*

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

[17] N. Lawrence, "Probabilistic non-linear principal component analysis with gaussian process latent variable models," University of Sheffield, 2005.

[18] Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: The gaussian process approach to temporal difference learning," 2003.

[19] A. C. Damianou and N. D. Lawrence, "Deep gaussian processes," 2013.

[20] R. Frigola-Alcalde, "Bayesian time series learning with gaussian processes," University of Cambridge, 2015.

[21] A. Francis and N. Golmant, "Derivations of the univariate and multivariate normal density."

[22] K. Krauth, E. V. Bonilla, K. Cutajar, and M. Filippone, "Autogp: Exploring the capabilities and limitations of gaussian process models," 2017.

[23] M. Filippone and R. Engler, "Enabling scalable stochastic gradient-based inference for gaussian processes by employing the unbiased linear system solver (ulisse)," 2015.

[24] R. Malouf, "A comparison of algorithms for maximum entropy parameter estimation," 2002.

[25] SheffieldML. (). Gpy, [Online]. Available: `https://github.com/SheffieldML/GPy`.