



Probabilistic Machine Learning for Chronic Kidney Disease Progression Forecasting

Candidate: Theofanis Cheras ¹

Supervisor: Prof. Mark Herbster

Industry partner: UCB Pharma

Submission date: 12th September 2022

¹**Disclaimer:** This report is submitted as part requirement for the MSc Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

Chronic kidney disease (CKD) has always been a major worldwide public health problem, as it is widely known that patients suffering from CKD display increased risk for a range of adverse clinical events. The severity of a patient’s condition can be determined by the estimated glomerular filtration rate (eGFR), but currently, there are no models that can accurately predict CKD progression to a desired level of accuracy. In this thesis, we investigate two approaches to tackling the CKD progression forecasting problem, with an emphasis on probabilistic machine learning methods capable of quantifying uncertainty. More precisely, we firstly attempt to forecast patient eGFR time-series trajectories directly, using a cutting-edge class of mixed-effects models, that have shown impressive predictive capabilities over traditional models in the field. Secondly, we present a novel methodology for converting the CKD progression prediction problem to a classification task. To this end, we propose a data-driven method of assigning CKD progression status labels to individual patients, by applying time-series clustering techniques on their eGFR trajectories. In turn, we use those labels as target values for training and evaluating various classification models. Results based on a real CKD patient dataset are highly suggestive that the probabilistic models developed in this thesis could be invaluable in assisting healthcare professionals in making critical decisions.

The code for all the experiments performed for this thesis can be found at <https://github.com/pcheras/Masters-thesis>.

Acknowledgements

I would like to thank Professor Mark Herbster, Dr. Karim Malki, Dan Manela and Jason Zhang for their continuous support and guidance, and for providing regular feedback and insightful ideas. I am also grateful to UCB Pharma, for providing the dataset which formed the basis of the experiments performed in this thesis.

List of Figures

2.1	Draws from a Gaussian Process prior with a Matérn kernel, using different smoothness parameter values.	23
2.2	Covariance matrices and sampled functions from a Gaussian Process prior with a squared-exponential kernel under different hyperparameter settings.	25
2.3	Optimal path (in white) superimposed on a similarity matrix of Euclidean distances between time-series points.	38
2.4	Dendrogram example in Hierarchical agglomerative clustering, where each color corresponds to a distinct cluster.	41
3.1	Random-effects vectors $R_i(\mathbf{t}_i)$ across different <i>kernel lengthscale</i> values under a common GP.	50
3.2	Random-effects vectors $R_i(\mathbf{t}_i)$ across different <i>kernel marginal variance</i> values under a common GP.	50
3.3	Simulated random effects and draws from the corresponding GP prior of the true data-generating process.	60
3.4	GP posteriors and the corresponding true random effects of patient 3 simulated data.	63
3.5	GP posteriors and the corresponding true random effects of patient 1 simulated data.	63
3.6	GP posteriors and the corresponding true random effects of patient 20 simulated data.	63
4.1	Boxplots for the eGFR values of patients in the two medical sites. . .	68
4.2	GP Boost feature importance estimates on the Sheffield data.	72
4.3	GP Boost feature importance estimates on the Patras data.	74
4.4	Individual patient GP posteriors using the top-performing model of Table 4.3 and the corresponding predicted random effects across three different patients.	76

5.1	A subset of the patients' eGFR time-series trajectories.	78
5.2	Samples from the data-generating process described in Section 5.1.1. .	80
5.3	Subset of eGFR trajectories from the clusters formed using k -Means with $k = 3$	86
5.4	The architecture of the feed-forward neural network models 4-6, with the number of hidden units in each layer displayed at the bottom. For example, a layer labelled as \mathbb{R}^{500} indicates that there are 500 hidden units in that layer.	90
5.5	ROC curves for test set predictions, at a specific run of the experiments.	92
5.6	Predicted label probabilities on the test set, at a specific run of the experiments.	93
A.1	Histogram of the Silhouette coefficient values for the k -Means algorithm.	102
A.2	Posterior samples (right column) and their respective approximate dis- tributions (left column) shown for two distinct Markov chains, com- puted using data from patient 3.	103
A.3	Snapshot of our longitudinal CKD patient dataset.	103
A.4	Partial dependence plot for the three most important features <i>in the Sheffield data</i> . The vertical and horizontal axes denote eGFR and the corresponding feature values respectively, the line shows the condi- tional mean function, and the shaded area is a 95% confidence interval centred at the mean function.	104
A.5	Partial dependence plot for the three most important features <i>in the Patras data</i> . The vertical and horizontal axes denote eGFR and the corresponding feature values respectively, the line shows the condi- tional mean function, and the shaded area is a 95% confidence interval centred at the mean function.	105
A.6	Conversion table relating a patient's eGFR value and Chronic Kidney Disease stage.	106
A.7	Snapshot of our cross-sectional CKD patient dataset.	106

List of Tables

3.1	Mean and standard deviation of RMSE scores of experiments on synthetic data in the <i>sparse data regime</i>	57
3.2	Mean and standard deviation of RMSE scores of experiments on synthetic data in the <i>full data regime</i>	58
3.3	GP kernel hyperparameter estimates using the GP Boost algorithm and the proposed Bayesian approach, calculated using patient 3 training data.	62
4.1	CKD dataset features, their descriptions and data types.	67
4.2	Mean and standard deviation of RMSE scores on the <i>Sheffield data</i> . .	72
4.3	Mean and standard deviation of RMSE scores on the <i>Patras data</i> . . .	74
5.1	Mean and standard deviation of Rand index scores for the experiments on the synthetic data.	83
5.2	Silhouette scores for different number of clusters, k , for the three algorithms considered.	85
5.3	Mean and standard deviation of test set accuracy, MCC and ECE scores for the CKD patient progression status classification task. . . .	91

Contents

1	Introduction	9
1.1	Research motivation and objective	9
1.1.1	Background on chronic kidney disease	9
1.1.2	The task of chronic kidney disease progression forecasting . . .	10
1.2	Thesis structure	11
1.3	Related work	13
1.4	Our contribution	16
2	Theoretical Background	18
2.1	Linear mixed-effects models	18
2.1.1	Estimating the model coefficients	19
2.2	Gaussian Processes	21
2.2.1	Covariance functions	22
2.2.2	Gaussian Process regression	24
2.2.3	Hyperparameter selection	26
2.2.4	Large-scale approximations	28
2.3	Gaussian Process boosting	28
2.3.1	Model assumptions	29
2.3.2	Model fitting	30
2.3.3	Model prediction	33
2.3.4	Vecchia approximation	34
2.4	Time-series clustering	36
2.4.1	Dynamic time warping as a similarity measure	36
2.4.2	Time-series k -Means algorithm	37
2.4.3	Hierarchical agglomerative clustering	40
2.5	Bayesian neural networks	42
2.5.1	Laplace approximation	44
2.5.2	Stochastic gradient Langevin dynamics	45

3	Experiments on Synthetic Longitudinal Data	47
3.1	Experimental setup	47
3.1.1	Data-generating process	47
3.1.2	Performance evaluation	51
3.1.3	Models specification	52
3.2	Results & analysis	55
3.3	A Bayesian approach towards hyperparameter selection	56
3.3.1	Motivation and setup	56
3.3.2	Approach & findings	60
3.4	Conclusion	64
4	Forecasting Patient eGFR Trajectories	65
4.1	Dataset description & pre-processing	65
4.2	Experimental setup	68
4.2.1	Models specification	68
4.2.2	Hyperparameter selection	70
4.3	Results & analysis	70
4.3.1	Sheffield medical site	70
4.3.2	Patras medical site	73
4.4	Conclusion	75
5	Forecasting CKD Patient Progression Status	77
5.1	Time-series clustering for eGFR trajectories	77
5.1.1	Synthetic data experiments	79
5.1.2	Clustering patient eGFR trajectories	83
5.2	CKD progression forecasting as a classification task	86
5.2.1	Dataset description	86
5.2.2	Evaluation metrics	87
5.2.3	Models specification	89
5.2.4	Results & analysis	91
5.3	Conclusion	93
6	Conclusion and Future Directions	94
6.1	Summary	94
6.2	Future directions	94
	Bibliography	101

Appendix	101
A Additional figures	102

Chapter 1

Introduction

In this chapter, we introduce the CKD patient progression forecasting problem, and motivate the approach taken throughout this thesis. In turn, we provide a description of the layout of this thesis by briefly going through the contents of each chapter. Then, we present the relevant work that has been previously done in the field. Finally, we summarise the contribution of this thesis towards the CKD patient progression task.

1.1 Research motivation and objective

1.1.1 Background on chronic kidney disease

Chronic kidney disease (CKD) is a long-term condition in which kidney functionality is not as high as it should be. In many cases, CKD worsens over time and in some cases patients might even experience kidney failure, which leads to death. As CKD is commonly linked to increased risk for other clinical events, public health organisations view CKD diagnosis and early treatment as a key goal [Go et al., 2004]. Even though a cure for CKD does not yet exist, early stage detection of CKD is of utmost importance for an effective treatment, which can dramatically reduce the rate of CKD progression. An early treatment can thus reduce the chance of a patient progressing to end-stage renal disease, which is an advanced CKD stage. We note that in the United States (US), it is estimated that around eight million adults who suffer from CKD have progressed to the moderate and severe stages. In fact, the yearly healthcare costs associated with end-stage renal disease in the US are nearly 23 billion dollars [USRDS, 2003].

Early stage diagnosis of diseases has been fundamental to the field of medicine for a very long time. Doctors are constantly required to make accurate predictions, and take important decisions regarding the provision of treatment to patients, and in many cases, in the presence of partial medical information. For example, specific to CKD, a doctor might have to predict the likelihood that a patient in the early stages of CKD will quickly progress to the more severe stages. In this way, reliable decision-making can aid in making the healthcare system more efficient in terms of resource allocation, and reduce unnecessary costs by prioritising CKD patients who are expected to progress more quickly than others. Hence, one can easily see how accurate predictive models could prove to be very handy in helping doctors and physicians with early stage detection, and disease diagnosis. Traditionally, clinical decision-making used to be more subjective, and was guided by a doctor’s professional experience. Nowadays, advanced algorithms and statistical techniques are revolutionising all aspects of modern medicine by extracting hidden patterns within the ever-expanding notion of big data. Nevertheless, being able to accurately quantify the uncertainty in critical decisions is undoubtedly a very challenging and important task. Therefore, having mentioned the importance of precise uncertainty quantification, it is clear that probabilistic machine learning can be a very useful framework to achieve this [Chen et al., 2020].

1.1.2 The task of chronic kidney disease progression forecasting

The danger of CKD progression to its more advanced stages, such as end-stage renal disease, is highly dependent on an individual’s Glomerular Filtration Rate (GFR), which can be roughly described as a measure of one’s kidney functionality. At its early stages, a GFR test could be useful in diagnosing CKD even if the typical symptoms are not present. However, GFR tests are quite complex in nature and are only possible through specialised medical providers. As a result, it is common that instead of directly measuring GFR, one can simply use a mathematical formula to come up with an estimate of the GFR, *also known as the estimated Glomerular Filtration Rate (eGFR)*. This has the benefit that eGFR can be obtained using easily accessible patient information, such as blood creatinine levels, age, weight etc. Moreover, it has been empirically shown in [Hafeez AR, 2016] that eGFR can be used as a reliable estimate of GFR in most cases. Thus, CKD progression can be accurately predicted if future eGFR estimates can be obtained precisely through

predictive modelling. A table relating eGFR values and CKD progression stage is seen in Figure A.6. However, the task of accurately predicting CKD progression based on patient eGFR trajectories has for several reasons proved to be a rather difficult one. Firstly, one of the biggest difficulties is around missing data in patients' electronic health records, as well as data sparsity for some patients. For example, being able to predict a patient's CKD progression using very few data points is a hard task. This is something that we will explore in the later parts of this thesis.

Furthermore, an additional complication is that the type of datasets used for this task are of longitudinal nature (in contrast to cross-sectional datasets which are the most common in machine learning applications). Longitudinal datasets basically contain the same covariate measurements for a subject across different points in time. A snapshot of the longitudinal and cross-sectional datasets we will be using can be seen in Figures A.3 and A.7 respectively. In particular, this means that different within-patient observations are not independent, which is a common assumption in most supervised learning algorithms. As a result, special types of statistical models are required to capture within-patient correlations, and improve the model performance. Lastly, a further complication common in longitudinal data is the fact that within-patient measurements are often made at irregular points across time, and hence each subject has a different number of data points. For the above reasons, the accurate forecasting of future eGFR values for CKD patients still remains an interesting problem, and one of the contributions of this thesis is towards solving it.

1.2 Thesis structure

We will be now presenting the outline of this thesis, and what each chapter is about.

Chapter 1 - Introduction

So far in this chapter we have provided background information around CKD, and highlighted how public health organisations could benefit from accurate predictive models, with the ability to quantify uncertainty. In the next section, we will present our literature review, and see what has been already explored in the area of CKD progression prediction. Finally, in the last section, we clearly state the contributions of this thesis.

Chapter 2 - Theoretical Background

In the second chapter, we will go into the details behind the theory of the models and methods used throughout the later chapters of this thesis. In particular, we will dive deeper into the modelling of longitudinal data through two types of mixed-effects models, linear mixed-effects models and the state of the art Gaussian Process Boosting model. Also, we will be presenting theory on probabilistic machine learning topics such as Bayesian neural networks and Gaussian Processes. Towards the end, we will also provide background on time-series clustering.

Chapter 3 - Experiments on Synthetic Longitudinal Data

In Chapter 3, our aim is to compare and evaluate the predictive performance of the two aforementioned types of mixed-effects models (and their variants), under different data-generation settings. We also dive deeper into Gaussian Process Boosting models, and explore how the learning process can be improved through the use of prior distributions.

Chapter 4 - Forecasting Patient eGFR Trajectories

In this chapter, we take the insights from the experiments performed in chapter 3, and evaluate the models we developed on a real CKD patient dataset. Specifically, the focus of this chapter is on forecasting patient progression through our first approach, that is, by directly modelling patients' future eGFR values.

Chapter 5 - Forecasting CKD Patient Progression Status

In Chapter 5, we explore our second proposed approach towards CKD progression forecasting. This includes a novel method for assigning data-driven CKD patient progression status labels, using time-series clustering techniques on patient eGFR trajectories. Then, using the resulting clustering labels as target values, we convert our dataset from longitudinal to cross-sectional and treat the problem as a three-class classification problem.

Chapter 6 - Conclusion and Future Directions

In the final chapter, we provide a succinct summary of our findings based on the experiments conducted, and also discuss potential directions on how the

research of this thesis can be further advanced.

1.3 Related work

Within the literature around the CKD progression problem, most of the work on predictive modelling has been using cross-sectional datasets, and only takes into account covariates across a single point in time to make future predictions for a specific time period. It is often the case, that models built for this setting are most useful for explaining the variability in eGFR values by conditioning on the features present in the dataset. For example, in [Tangri et al., 2011], the authors model a CKD patient’s time until kidney failure using a proportional hazards model, which belongs to the famous class of statistical survival models. However, such types of models are unable to make personalised patient predictions in an on-line fashion, as the model parameters have to be re-estimated every time a new patient observation arrives. This renders proportional hazards models unsuitable for assisting medical professionals in real-time decision-making. A range of papers have been published on building predictive models for CKD, but the vast majority are logistic or Cox regression models for cross-sectional datasets [van Dijk et al., 2008]. Furthermore, such models are typically used on data coming from windows of fixed-size, as compared to windows which rather grow with time.

To address the challenge of longitudinal data modelling, and incorporate personalised predictions in an on-line manner, state space models [Lintu et al., 2022], and several variants including the famous hidden Markov model have been proposed for modelling CKD progression. Nevertheless, these models have been found to work well in settings where patient data are observed at evenly-spaced time intervals, and the number of covariates at each point in time is relatively small. However, in many medical settings, including ours, these assumptions are certainly invalid and therefore render such models inappropriate. In [Rizopoulos, 2011], the author proposes a methodology based on joint models for an on-line dynamical predictive model about time-to-event outcomes using longitudinal data. This approach improves over the previous methods in the sense that future model predictions are of sequential nature and effectively incorporate previous observations, while also accounting for unobserved sources of variability in the data.

Regarding work done on mixed-effects models for longitudinal data, traditionally, linear mixed-effects models (LMEs) [Verbeke, 1997], have been a popular choice across a range of different fields, also including the healthcare domain. Their attrac-

tiveness is due to their relative simplicity over alternative approaches, and the fact that they are well understood in the statistical and machine learning communities. However, a strong assumption made by LMEMs is that their functional form implies that the target variable is a linear combination of the fixed and random effects covariates. We are well aware that in many cases, and when working with real datasets, the relationship between covariates and the target variable can be in fact be highly non-linear. As a result, many extensions have been proposed, including generalised linear mixed models [Stroup, 2012], which can be viewed as a natural extension to LMEMs in the same way that generalised linear models are to the linear regression model. Specifically, these types of models provide a unifying framework for modelling a range of continuous or discrete target variables, under the longitudinal data setting. However, it has been noted in [Gad and El Kholy, 2012], that non-Gaussian random effects introduce computational and analytical difficulties in the parameter estimation procedure of these models.

Moving on to generalised additive mixed models, an extension to generalised additive models which can also account for random effects, have also been used to capture correlations present in longitudinal data. In [Groll and Tutz, 2012], generalised additive mixed models were successfully used to model disease progression on a real dataset from the multicentre AIDS cohort study. To be more precise, these models were able to estimate the underlying regression function more accurately, and performed effective feature selection in a high-dimensional setting where other models had failed to do so. Other methods which relax the linearity assumption in longitudinal data analysis tasks have been around the use of tree-based methods. For example, in [Hajjem et al., 2011], the authors come up with the mixed-effects regression tree algorithm, an extension to the classical regression tree approach for clustered data. A few years later, they also propose the mixed-effects random forest algorithm [Hajjem et al., 2014], which utilises random forests, a state of the art tree-based ensemble algorithm which performs extremely well in a range of supervised learning tasks. The two tree-based methods can be seen as semi-parametric approaches to mixed-effects modelling, where a tree-based model is used to model the fixed-effects part. What made these models interesting was their unique fitting procedure, which involved estimating the model parameters in an Expectation Maximisation (EM) style algorithm, by alternating between the estimation of the fixed and random effects parameters respectively. However, an important drawback is that the fitting process does not correspond to a properly defined EM algorithm, adjusted for mixed-effects models. As mentioned in [Sigrist, 2020], in many cases convergence

of these algorithms during training was not observed, and the authors state that it is unclear to what these algorithms are converging to.

Most relevant to our work are Gaussian Process-based approaches for longitudinal data [Cheng et al., 2018], [Vantini et al., 2022]. Gaussian Processes (GPs) [Rasmussen and Williams, 2017], a flexible class of Bayesian non-parametric models, are frequently used in continuous-time medical applications. Their suitability for the CKD progression forecasting task, is due to the fact that GPs are distributions over functions and hence provide a very natural framework for modelling highly non-linear eGFR patient trajectories while also producing reliable uncertainty estimates [Futoma et al.,], [Soleimani et al., 2018]. We mention though that, as highlighted in [Roberts et al., 2013], the selection of the GP mean function is critical for the resulting model. The importance of a suitably chosen mean function is also shown in [Schulam and Saria, 2016], where the authors propose a model for predicting disease trajectories using a GP model with a highly structured mean function consisting of different observed and latent components. In this thesis, we explore GP-based mixed-effects models, in which a gradient boosted tree is used for the corresponding GP mean functions. Other relevant work in this area includes [Dürichen et al., 2014], in which the authors model longitudinal clinical data using multi-task GPs, which can be thought of as a multi-task learning framework in the context of GPs. A limitation of the approach taken in [Dürichen et al., 2014] is that there needs to be an individually trained model for each patient, and this does not allow for effective modelling of population-level correlations (i.e. the sharing of information across all patients). We also note that in a scenario where the number of observations per individual patient is limited, such models would not be expected to perform very well.

Finally, we provide a short summary of the literature around the unsupervised learning area of clustering [Jain et al., 1999], with applications to disease progression. The ultimate aim of clustering for disease progression is to identify the different latent subgroups in patients that display similar patterns in the quantity of interest (eGFR values in our case). Previous work done on this topic includes [Liao, 2005], where the authors measure the similarity between the different time-series using proximity measures such as cross-correlation, in order to account for the temporal nature of the task. Such approaches are ill-equipped to deal with time-series found in medical data, since in many cases patient time-series are quite sparse, and time-points are irregularly sampled across time. Alternative approaches to overcome these issues include model-based clustering methods [Grün, 2019]. However, their

practical usefulness has proved to be limited as they mostly involve complicated inference procedures [Xiong and Yeung, 2002], or make assumptions regarding the structure of the time-series data which are often unrealistic [Li and Biswas, 1999]. Work done in explaining individual patient disease progression patterns using a variety of components includes the probabilistic sub-typing model [Schulam et al., 2015], which was specifically applied to longitudinal CKD data by [Luong et al., 2017] in an attempt to identify distinct patient subgroups. Although this model offers a nice probabilistic interpretation to the clustering task, it is very sensitive to hyperparameter tuning, which causes the clustering quality to vary a lot depending on the problem. Similar to the techniques explored in this thesis, an adapted k -Means algorithm suitable for sparse and irregularly sampled time-series data was proposed in [Anh Luong and Chandola, 2017]. Though, a fundamental and perhaps restrictive assumption underlying this approach was that, each individual time-series within a specific cluster could be approximated as a weighted sum of polynomial functions.

1.4 Our contribution

We begin this section by clearly stating that *the focus of this thesis is on probabilistic machine learning methods*. As already mentioned, the rationale behind focusing on such methods over their non-probabilistic counterparts boils down to the domain of application of these models, the healthcare domain in our case. Perhaps in a different application setting, one would be mostly interested in the predictive accuracy of these models, without being too interested in the model’s ability to “know when it doesn’t know” what the correct answer is. As an example, in the eGFR prediction task of this thesis, we focus on Gaussian Process-based models because of their ability to perform extremely well in terms of accuracy, but also provide well-calibrated estimates of uncertainty in their predictions [Bhatt et al., 2017].

The main contributions of this thesis can be concisely summarised as follows:

- Firstly, we experimentally show that Gaussian Process Boosting, a cutting-edge mixed-effects model, *significantly outperforms* traditional linear mixed-effects models on the task of eGFR forecasting. Additionally, we propose a modification to the Gaussian Process Boosting learning procedure, in which we *improve the estimation* of the Gaussian Process kernel hyperparameters by taking a Bayesian approach using prior distributions. In turn, we *empirically*

confirm that the values learned using this modified approach are not only more reliable, but also result in higher predictive accuracy and more robust learning.

- Secondly, we present *a novel methodology* based on time-series clustering techniques which allows us to assign CKD progression labels, where each patient’s progression status is set to “improving”, “worsening” or “stable”. We then use these assigned labels, to transform the original longitudinal dataset to a cross-sectional one. The cross-sectional version of the dataset then forms the basis of our second approach to tackling CKD progression forecasting, by treating it as a three-class classification problem.
- Finally, for this classification task, we implement state of the art Bayesian neural network models and evaluate their performance against various well-established classification algorithms. For these Bayesian neural network models, inference using the posterior distribution of the network weights is performed using two different methods. Firstly, through an approximate inference technique, and secondly, through Markov chain Monte Carlo methods. *Through multiple experiments, we conclude that these Bayesian neural network models are not only superior in terms of predictive performance, but also provide significant improvements in issues like prediction overconfidence, commonly exhibited by deep learning models.*

Chapter 2

Theoretical Background

The aim of this chapter is to introduce the theory of the methods used throughout this thesis, and to equip readers with an understanding of these methods. Specifically, we begin by introducing linear mixed-effect models, we move on to present theory on Gaussian Processes and then introduce the Gaussian Process Boosting model. Towards the later sections of this chapter, we also introduce theory on time-series clustering techniques, and finally give background on Bayesian neural networks.

2.1 Linear mixed-effects models

Linear mixed-effects models (LMEMs) are a class of statistical models commonly used to capture correlation in the data by having both fixed and random effects components. In addition, a fundamental model assumption is that the target variable is assumed to be a linear combination of the covariates. Below we present the general setting of LMEMs, in matrix-vector notation

$$\mathbf{y} = X\boldsymbol{\beta} + Z\mathbf{u} + \boldsymbol{\epsilon}, \tag{2.1}$$

such that

- $n, p, q \in \mathbb{N}$ represent the number of data points, fixed-effects and random-effects covariates in the dataset respectively;
- $\mathbf{y} \in \mathbb{R}^n$ is the target output vector, and is such that $\mathbb{E}(\mathbf{y}) = X\boldsymbol{\beta}$;
- $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of the fixed-effects coefficients;

- $\mathbf{u} \in \mathbb{R}^q$ is the vector of the random-effects coefficients, where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, G)$ and $\text{Cov}(\mathbf{u}_i, \mathbf{e}_j) = 0$, for all $1 \leq i \leq q$ and $1 \leq j \leq n$;
- $\mathbf{e} \in \mathbb{R}^n$ is a vector of random errors, such that $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$;
- $X \in \mathbb{R}^{n \times p}$ and $Z \in \mathbb{R}^{n \times q}$ are the design matrices containing the fixed and random effects covariates respectively.

It follows from the underlying model assumptions of (2.1) that the observations of the target vector \mathbf{y} are correlated with each other, since

$$\text{Var}(\mathbf{y}) = \text{Var}(X\boldsymbol{\beta} + Z\mathbf{u} + \mathbf{e}) = \text{Var}(Z\mathbf{u} + \mathbf{e}) = \text{Var}(Z\mathbf{u}) + \text{Var}(\mathbf{e}) = ZGZ^\top + \sigma^2 I,$$

where the second equality follows by noting that $X\boldsymbol{\beta}$ is fixed, and the third equality follows as \mathbf{u} and \mathbf{e} are uncorrelated. Note also that the above assumptions also imply that $\mathbf{y}|\mathbf{u} \sim \mathcal{N}(X\boldsymbol{\beta} + Z\mathbf{u}, \sigma^2 I)$.

Since the observations within \mathbf{y} are correlated (non-diagonal form of the covariance matrix), this violates a fundamental assumption underpinning many supervised learning algorithms, which highlights the importance of using mixed-effects models in settings where observations are not independent, such as when having covariate measurements across different points in time for the same patient. LMEMs notation specific to the longitudinal data setting will be shortly presented.

2.1.1 Estimating the model coefficients

Model fitting in LMEMs is performed using the method of maximum likelihood estimation, one of the most common techniques to fit statistical models. We can express the model likelihood as $\mathcal{L}(\boldsymbol{\beta}, \mathbf{u}) := p(\mathbf{y}, \mathbf{u}|\boldsymbol{\beta}) = p(\mathbf{y}|\mathbf{u}, \boldsymbol{\beta})p(\mathbf{u})$, and then maximise the likelihood over the coefficient vectors $\boldsymbol{\beta}$ and \mathbf{u} . Equivalently, to simplify things we can maximise the log-likelihood, which is specified as

$$\log \mathcal{L}(\boldsymbol{\beta}, \mathbf{u}) = -\frac{1}{2} \log |2\pi\sigma^2 I| - \frac{1}{2\sigma^2} (\mathbf{y} - X\boldsymbol{\beta} - Z\mathbf{u})^\top (\mathbf{y} - X\boldsymbol{\beta} - Z\mathbf{u}) - \frac{1}{2} \log |2\pi G| - \frac{1}{2} \mathbf{u}^\top G^{-1} \mathbf{u}$$

Differentiating the above expression with respect to $\boldsymbol{\beta}$ and \mathbf{u} yields

$$\begin{aligned}\frac{\partial \log \mathcal{L}(\boldsymbol{\beta}, \mathbf{u})}{\partial \mathbf{u}} &= \frac{1}{\sigma^2} Z^\top (\mathbf{y} - X\boldsymbol{\beta} - Z\mathbf{u}) - G^{-1}\mathbf{u} \\ \frac{\partial \log \mathcal{L}(\boldsymbol{\beta}, \mathbf{u})}{\partial \boldsymbol{\beta}} &= \frac{1}{\sigma^2} X^\top (\mathbf{y} - X\boldsymbol{\beta} - Z\mathbf{u})\end{aligned}$$

In turn, setting both of these equations to zero, and with a bit of algebra, we obtain Henderson's mixed-model equations

$$\begin{aligned}\frac{1}{\sigma^2} Z^\top \mathbf{y} &= \frac{1}{\sigma^2} Z^\top X \hat{\boldsymbol{\beta}} + \left(\frac{1}{\sigma^2} Z^\top Z + G^{-1} \right) \hat{\mathbf{u}} \\ \frac{1}{\sigma^2} X^\top \mathbf{y} &= \frac{1}{\sigma^2} X^\top X \hat{\boldsymbol{\beta}} + \frac{1}{\sigma^2} X^\top Z \hat{\mathbf{u}}\end{aligned}$$

The resulting maximum likelihood estimate solutions to the above equations, $\hat{\boldsymbol{\beta}}$ and $\hat{\mathbf{u}}$, are the best linear unbiased estimates for $\boldsymbol{\beta}$ and \mathbf{u} respectively. We also state that predicting on test data using the estimated coefficients is straightforward, and is essentially the same task as in the classical linear regression setting.

To help readers familiarise themselves with the notation that will be used in the later sections, we also introduce LMEMs in the longitudinal data setting, where for each group, there are multiple repeated measurements across time. More precisely, assume we have a total number of $s \in \mathbb{N}$ distinct groups, where within the i^{th} group there are $n_i \in \mathbb{N}$ different measurements across time, then we have

$$\mathbf{y}_i = X_i \boldsymbol{\beta} + Z_i \mathbf{u}_i + \boldsymbol{\epsilon}_i, \quad (2.2)$$

such that

- $\mathbf{y}_i \in \mathbb{R}^{n_i}$ is the i^{th} group target output vector, and is such that $\mathbb{E}(\mathbf{y}_i) = X_i \boldsymbol{\beta}$;
- $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of the fixed-effects coefficients, shared across all groups;
- $\mathbf{u}_i \in \mathbb{R}^q$ is the i^{th} group vector of the random-effects coefficients, where $\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, G)$ and $\text{Cov}(\mathbf{u}_{ij}, \boldsymbol{\epsilon}_{ik}) = 0$, for all $1 \leq j \leq q$ and $1 \leq k \leq n_i$;
- $\boldsymbol{\epsilon}_i \in \mathbb{R}^{n_i}$ is a vector of random errors, such that $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$;
- $X_i \in \mathbb{R}^{n_i \times p}$ and $Z_i \in \mathbb{R}^{n_i \times q}$ are the i^{th} group design matrices containing the fixed and random effects covariates respectively.

Two special cases of these models include the random intercept and random slopes model. A random slopes model arises when the random effects design matrix of each group only consists of a vector of ones, in which case only intercepts are allowed to vary across different groups. Similarly, a random slope model is one in which the slope coefficients for one or more covariates are allowed to vary across the different groups. Combining both, we can get the so-called random intercept and slope models, in which both the intercept and the slope coefficients (for different covariates) can vary across groups.

We state for completeness that the model coefficients estimation process is similar to the one described in 2.1.1. However, since this involves estimating a per group random-effects vector, the prediction process is slightly different. When making predictions on data coming from a group which was already seen during training time, the process is the same as before. When predicting on data coming from an unseen group though, we simply predict using the fixed-effects part of the model, as there are no available estimates of the random-effects coefficients for the new group.

2.2 Gaussian Processes

Gaussian Processes (GPs) are a flexible Bayesian non-parametric class of models that allow us to use prior distributions over function spaces. In a sense, a Gaussian Process is a stochastic process which can be viewed as an infinite-dimensional generalisation of the multivariate Normal distribution. Specifically:

Definition 1 *Given an arbitrary index set \mathcal{X} , a collection of random variables $\{f_x\}_{x \in \mathcal{X}}$ is a Gaussian Process if and only if, for every finite set of indices $\{x_1, \dots, x_n\}$, the random vector $[f_{x_1}, \dots, f_{x_n}]^\top$ has a multivariate Normal distribution on \mathbb{R}^n .*

More intuitively, a Gaussian Process is a collection of random variables such that any finite number of which have a joint distribution which is multivariate Normal. To ease with notation, we assume throughout the remainder of this section that our input space is defined to be the real line \mathbb{R} , in which case a Gaussian Process would be associated with a random function $f: \mathbb{R} \rightarrow \mathbb{R}$, such that $f(x) = f_x$, for all $x \in \mathbb{R}$. A fundamental property of Gaussian Processes is that they are fully specified by their mean and covariance functions, i.e.

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)], \\ k(x, x') &= \text{Cov}(f(x), f(x')) = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))], \end{aligned}$$

where the expectation operator is taken with respect to the latent function f , and x, x' are fixed elements of the set \mathbb{R} . Then, we can denote the respective Gaussian Process by

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \quad (2.3)$$

Following from the above definition of a Gaussian Process, consider a finite collection of input points $\mathbf{x} = [x_1, \dots, x_n]^\top \in \mathbb{R}^n$, then the associated set of evaluated function values $f(\mathbf{x}) \in \mathbb{R}^n$ has a jointly multivariate Normal distribution, specified as

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}))$$

In the above equation, $m(\mathbf{x}) \in \mathbb{R}^n$ denotes the mean vector, where $[m(\mathbf{x})]_i = m(x_i)$. In addition, $K(\mathbf{x}, \mathbf{x}) \equiv K_{\mathbf{x}, \mathbf{x}} \in \mathbb{R}^{n \times n}$ represents the covariance matrix between the vector of all function values $f(\mathbf{x})$. That is, an individual entry of the matrix is specified as $[K(\mathbf{x}, \mathbf{x})]_{ij} = k(x_i, x_j) = \text{Cov}(f(x_i), f(x_j))$. It is frequently assumed that the prior mean function of a Gaussian Process is simply the zero function, i.e. $m(x) = 0$, for all $x \in \mathbb{R}$, even though this is not a strict requirement.

2.2.1 Covariance functions

One of the most important aspects in Gaussian Process models are covariance (also known as kernel) functions, through which we can incorporate our prior beliefs regarding the properties of the functions we are trying to learn, and thus determine the model's ability to generalise effectively. A kernel is a positive-definite function and can be thought of as a way of measuring "similarity", that takes two input points (in this case in \mathbb{R}) and returns a scalar value, i.e. $k: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. In the context of Gaussian Processes, a kernel function is used to measure the covariance between any two function values.

Below, we present and briefly describe two of the most popular types of covariance functions, namely, the squared-exponential and the Matérn covariance functions. Both of these covariance functions are stationary, which means that the value of the covariance function only depends on the distance between two input points and not their location in the input space. More concretely, the Matérn kernel evaluated at two input points $x, x' \in \mathbb{R}$, is specified as follows

$$k_\nu(x, x') = \sigma^2 \cdot \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \cdot \frac{|x - x'|}{l} \right)^\nu K_\nu \left(\sqrt{2\nu} \cdot \frac{|x - x'|}{l} \right),$$

where Γ is the gamma function, K_ν is the modified Bessel function of the second kind, and σ^2 , l and ν are the variance, lengthscale and smoothness hyperparameters. In Figure 2.1, we can see the effect the hyperparameter ν has on the smoothness of the functions which are possible under this Gaussian Process prior.

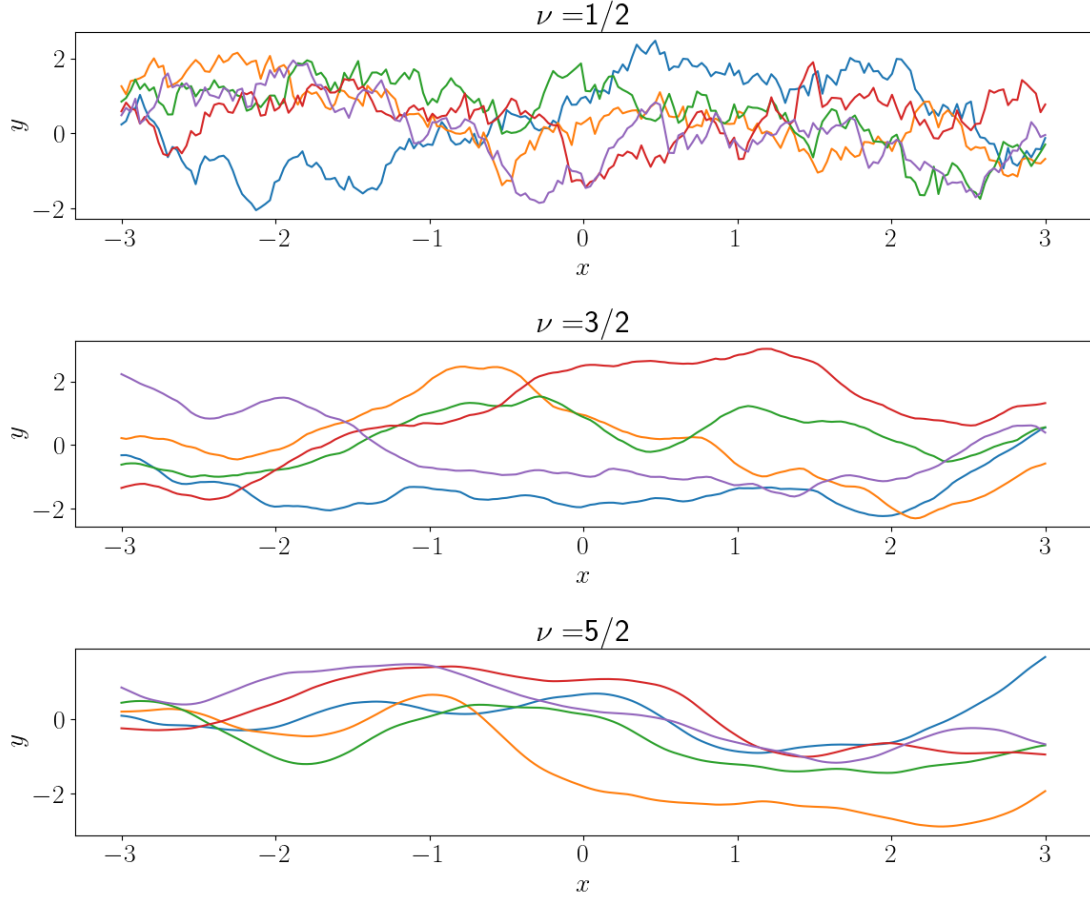


Figure 2.1: Draws from a Gaussian Process prior with a Matérn kernel, using different smoothness parameter values.

Moving on to the squared-exponential covariance function, *the kernel function used in our Gaussian Process-based models* in the later sections, evaluated at two input points, $x, x' \in \mathbb{R}$, it can be expressed as

$$k(x, x') = \sigma^2 \cdot \exp \left(\frac{-(x - x')^2}{l} \right), \quad (2.4)$$

where σ^2 and l are the marginal variance and lengthscale hyperparameters respectively. Intuitively, the marginal variance hyperparameter controls the magnitude of the range of function values that are possible. As for the lengthscale hyperparameter, informally, this hyperparameter controls how wiggly the sampled functions can be. Let us also re-state that the covariance matrix of a Gaussian Process directly controls the properties of the resulting function draws, as observed in Figure 2.2. We see that higher values of the lengthscale hyperparameter l result in functions which vary less rapidly, as the function values become more correlated. An interesting connection between the two kernels is that as $\nu \rightarrow \infty$, the Matérn kernel function converges to the squared-exponential kernel function.

2.2.2 Gaussian Process regression

As relevant to the context of this thesis, let us assume that we have data of time-series nature, $\mathcal{D} = \{\mathbf{x} = [x_1, \dots, x_n]^\top, \mathbf{y} = [y_1, \dots, y_n]^\top\}$, with the following data-generating process, $y_i = f(x_i) + \epsilon_i$, such that f is the true latent underlying regression function and $\epsilon_i \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$ is independent and identically distributed Gaussian noise. Treating f as a random function, we can model it by using a Gaussian Process prior, $p(f)$, and then estimate its corresponding posterior $p(f|\mathcal{D})$ after observing the data. To be more precise, the posterior distribution can be obtained using Bayes' Theorem, which is a principled way of performing Bayesian inference by incorporating prior information and the observed data,

$$p(f|\mathcal{D}, \theta) = \frac{p(\mathcal{D}|f, \theta)p(f|\theta)}{p(\mathcal{D}|\theta)} = \frac{p(\mathcal{D}|f, \theta)p(f|\theta)}{\int p(\mathcal{D}|f, \theta)p(f|\theta) \, df}, \quad (2.5)$$

where θ represents the hyperparameters of the Gaussian Process kernel and mean functions.

Our assumption of independent and identically distributed Gaussian noise implies that conditioning on the latent function f makes the observations y_i independent of each other. As a result, the model likelihood conveniently factorises in the following way

$$p(\mathcal{D}|f, \theta) = p(\mathbf{y}|f, \mathbf{x}, \theta) = \prod_{i=1}^n p(y_i|f(x_i), \theta) = \prod_{i=1}^n \mathcal{N}(y_i|f(x_i), \sigma^2) = \mathcal{N}(\mathbf{y}|f(\mathbf{x}), \sigma^2 I), \quad (2.6)$$

where I is an $n \times n$ identity matrix.

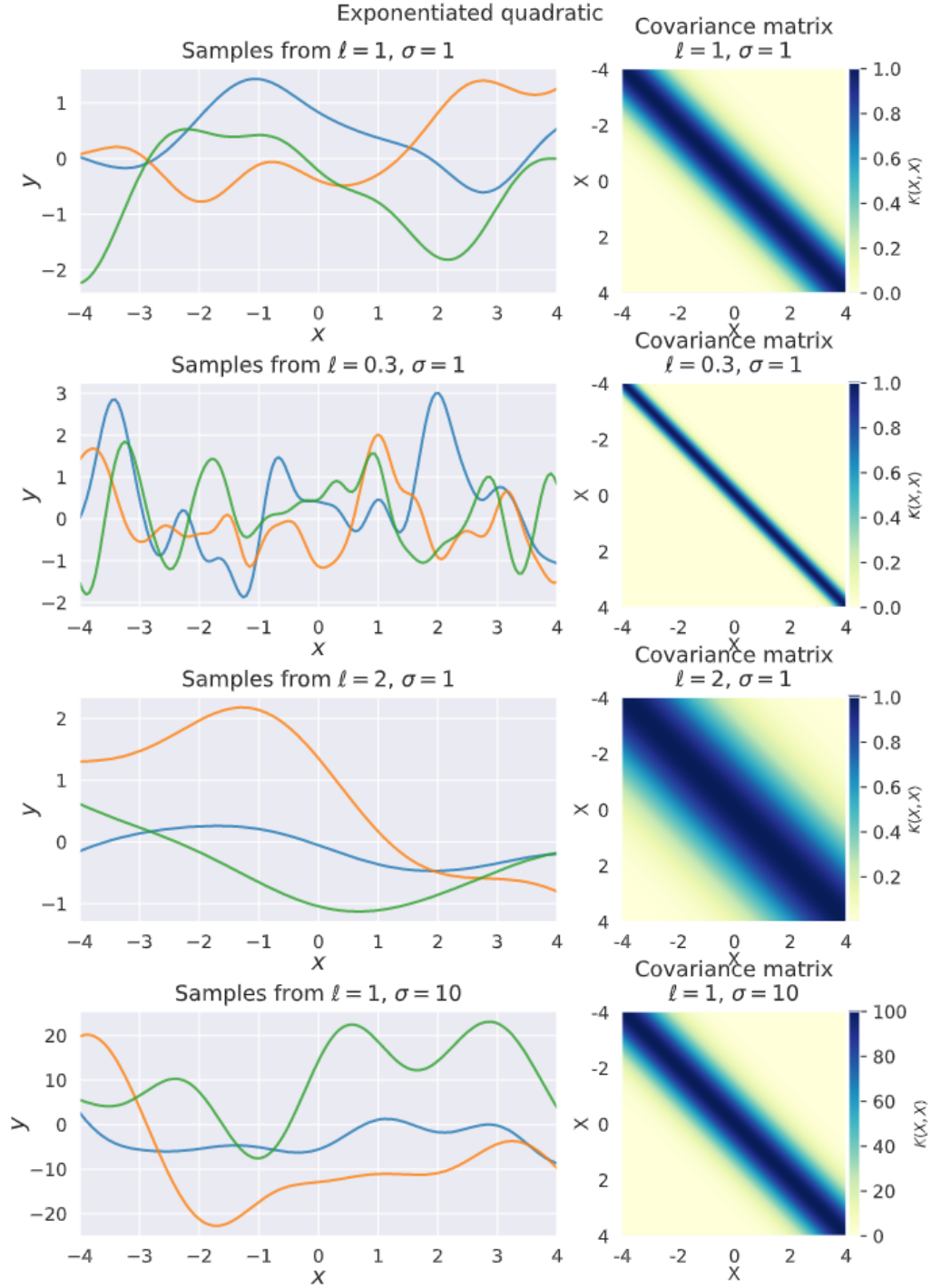


Figure 2.2: Covariance matrices and sampled functions from a Gaussian Process prior with a squared-exponential kernel under different hyperparameter settings.

Posterior inference

Since both the Gaussian Process prior (evaluated at the data \mathcal{D}) in (2.3) and the model likelihood in (2.6) are Gaussian, this means that the distribution of the vector

$[f, \mathbf{y}]^\top \in \mathbb{R}^{2n}$ is also Gaussian. Now, it easily follows by applying the well-known Gaussian conditioning identity that the Gaussian Process posterior distribution is given by

$$\begin{aligned} f | \mathcal{D}, \theta &\equiv f | \mathbf{y}, \mathbf{x}, \theta \sim \mathcal{GP}(m_{\text{post}}(x), k_{\text{post}}(x, x')), \\ m_{\text{post}}(x) &= k_{x, \mathbf{x}}(K_{\mathbf{x}, \mathbf{x}} + \sigma^2 I)^{-1} \mathbf{y} \\ k_{\text{post}}(x, x') &= k(x, x') - k_{x, \mathbf{x}}(K_{\mathbf{x}, \mathbf{x}} + \sigma^2 I)^{-1} k_{\mathbf{x}, x'} \end{aligned} \quad (2.7)$$

Note that $k_{x, \mathbf{x}} = [k(x, x_1), \dots, k(x, x_n)]^\top \in \mathbb{R}^n$ is a vector of covariances between an input point $x \in \mathbb{R}$ and all training input points $\mathbf{x} \in \mathbb{R}^n$, and that $k_{\mathbf{x}, x'} = k_{x', \mathbf{x}}^\top$.

Posterior predictive distribution

When it comes to predicting at a vector of input test points \mathbf{x}^* , the posterior predictive distribution becomes

$$\begin{aligned} f(\mathbf{x}^*) | \mathbf{y}, \mathbf{x}, \theta &\sim \mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*), \\ \boldsymbol{\mu}^* &= K_{\mathbf{x}^*, \mathbf{x}}(K_{\mathbf{x}, \mathbf{x}} + \sigma^2 I)^{-1} \mathbf{y} \\ \Sigma^* &= K_{\mathbf{x}^*, \mathbf{x}^*} - K_{\mathbf{x}^*, \mathbf{x}}(K_{\mathbf{x}, \mathbf{x}} + \sigma^2 I)^{-1} K_{\mathbf{x}, \mathbf{x}^*} \end{aligned} \quad (2.8)$$

By observing (2.8) we can see that we obtain a distribution over the latent function f evaluated at the test points \mathbf{x}^* . This implies that not only we can predict at test points by taking the vector $\boldsymbol{\mu}^*$ as our prediction estimate, but we can also use the covariance matrix Σ^* to come up with confidence intervals for our predictions, which is a great way to quantify predictive uncertainty.

2.2.3 Hyperparameter selection

So far throughout this section, we have assumed that the kernel hyperparameters θ and the model likelihood noise variance σ^2 were fixed. While it is true that sometimes prior information in the form of expert knowledge could be very useful in directly setting these hyperparameter values, it is often the case that we need to learn them in a principled way that would prevent overfitting. This is exactly where the use of the marginal likelihood comes in. Let us denote the model hyperparameters as $\boldsymbol{\phi} := (\theta, \sigma^2)$, then the marginal likelihood is defined as

$$p(\mathbf{y}|\phi) = \int p(\mathbf{y}|f, \phi)p(f|\phi) \, df = \mathcal{N}(\mathbf{y}|\mathbf{0}, K_\phi),$$

where $K_\phi := K_\theta + \sigma^2 I$. The result follows by using the conjugacy property of the Normal distribution.

Thus, we can write the marginal log-likelihood as

$$\log p(\mathbf{y}|\phi) = -\frac{1}{2}\log|K_\phi| - \frac{1}{2}\mathbf{y}^\top K_\phi^{-1}\mathbf{y} - \frac{n}{2}\log(2\pi). \quad (2.9)$$

To obtain the optimal values which maximise the marginal log-likelihood, we can take the derivative of expression (2.9) with respect to the i^{th} hyperparameter to get

$$\frac{\partial}{\partial \phi_i} \log p(\mathbf{y}|\phi) = -\frac{1}{2}\text{Tr}\left(K_\phi^{-1} \frac{\partial K_\phi}{\partial \phi_i}\right) + \frac{1}{2}\mathbf{y}^\top K_\phi^{-1} \frac{\partial K_\phi}{\partial \phi_i} K_\phi^{-1} \mathbf{y}. \quad (2.10)$$

However, we note that the marginal log-likelihood function is often non-convex and can have multiple local maxima, which makes optimising it a non-trivial task. Thus, typically one resorts to gradient-based numerical optimisation methods such as gradient ascent.

An alternative approach to maximising the marginal log-likelihood is to take a Bayesian approach by placing prior distributions over the hyperparameters, and inferring their posterior distributions. This approach will be specifically investigated and be contrasted against the maximum marginal likelihood approach in the context of the Gaussian Process Boosting algorithm, which will be presented subsequently. More precisely, one can set a prior on the hyperparameters, $p(\phi)$, and use it to draw posterior samples as follows

$$p(\phi|\mathbf{y}) = \frac{p(\mathbf{y}|\phi)p(\phi)}{p(\mathbf{y})} \propto p(\mathbf{y}|\phi)p(\phi) = \int p(\mathbf{y}|f, \phi)p(f|\phi) \, df \cdot p(\phi). \quad (2.11)$$

Approximate computation using the posterior distribution of the hyperparameters usually requires advanced Markov chain Monte Carlo sampling techniques, such as Hamiltonian Monte Carlo [Neal et al.,]. Then, using our samples from the hyperparameter posterior distributions, we can either use point estimates by calculating quantities such as the posterior mean or median, or even take a fully Bayesian approach which would involve integrating the uncertainty over the hyperparameters into our predictions. The latter would of course involve approximating the below

analytically intractable integral for calculating the posterior predictive distribution

$$p(f^*|\mathbf{y}) = \int p(f^*|\mathbf{y}, \phi)p(\phi|\mathbf{y}) \, d\phi.$$

2.2.4 Large-scale approximations

The main limitation of using Gaussian Processes lies within their computational and memory complexity. For example, the cost of training and prediction scales in $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ respectively, where n denotes the number of training points. These costs arise as a result of having to store and invert the matrix $(K_{\mathbf{x},\mathbf{x}} + \sigma^2 I)$, which is present in expressions (2.8), (2.9) and (2.7). For these reasons, working with Gaussian Processes on large datasets (e.g. $n > 10^5$) is viewed as practically infeasible.

To address issues related to scalability, different methods have been proposed and explored. To mention a few, sparse Gaussian Processes [Snelson and Ghahramani, 2005], utilise low-rank approximations for the kernel matrix via the inducing points method. Without going in too much detail, we state that the computational complexity of this technique provides significant improvement over exact inference as it scales in $\mathcal{O}(nm^2)$, where $m \ll n$ is the number of inducing points. Perhaps more relevant to the content of this thesis, is the Vecchia approximation method [Katzfuss and Guinness, 2021], which reduces the computational cost of training to be in $\mathcal{O}(nk^3)$, where again $k \ll n$. We will provide more details for this kind of approximation in Section 2.3. We conclude this section by stating that scalable Gaussian Processes that maintain similar predictive performance as exact inference methods are still a very active area of research.

2.3 Gaussian Process boosting

In this subsection, we go in detail into how the Gaussian Process boosting (GP Boost) algorithm [Sigrist, 2020] works. We begin by mentioning that GP Boost is designed to provide a flexible mixed-effects model that can perform well in settings where the true underlying regression function is linear but also highly non-linear. In essence, the beauty of this algorithm is two-fold. Firstly, in the majority of mixed-effect models involving GPs, it is assumed that the underlying mean function is linear. GP Boost provides an improvement in the sense that it relaxes this assumption by using a Gradient Boosted Tree (GBT) model [Freund et al., 1996]

for the fixed-effects part, which has proved to be one of the most competitive supervised learning algorithms across a range of different tasks. However, it is well-known [Freund and Schapire, 1999] that most boosting algorithms assume that different data points are independent of each other. This is where the second novel improvement of GP Boost comes in, by allowing for more efficient learning of the fixed-effects part function under longitudinal data settings in which data points are correlated with each other. As a bonus, when using a GP for the random-effects part of the algorithm, we obtain confidence intervals for our predictions. We also state that the GP Boost algorithm also allows for a random intercept model, and does not have to necessarily use a GP to model the random effects of the data. In Section 2.3.1, *we present the theory for the more interesting case where the random-effects part is modelled by a GP.*

2.3.1 Model assumptions

Similar to the LMEM longitudinal setting introduced in 2.2, the GP Boost algorithm assumes a mixed-effects model of the form

$$\mathbf{y}_i = F(X_i) + R(Z_i|\boldsymbol{\theta}) + \boldsymbol{\epsilon}_i, \quad (2.12)$$

such that

- $n_i \in \mathbb{N}$ represents the number of data points (varying across time) of the i^{th} patient;
- $\mathbf{y}_i \in \mathbb{R}^{n_i}$ is the target output vector, and is such that $\mathbb{E}(\mathbf{y}_i) = F(X_i)$;
- $\boldsymbol{\epsilon}_i \in \mathbb{R}^{n_i}$ is a vector of random errors, such that $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$;
- $X_i \in \mathbb{R}^{n_i \times p}$ and $Z_i \in \mathbb{R}^{n_i \times q}$ are the i^{th} patient design matrices containing the fixed and random-effects covariates respectively;
- $F(X_i) \in \mathbb{R}^{n_i}$ is the fixed-effects vector, such that $F(X_i) := (F(X_{i,1}), \dots, F(X_{i,n_i}))^\top \in \mathbb{R}^{n_i}$ is the row-wise evaluation of some possibly non-linear function $F: \mathbb{R}^p \rightarrow \mathbb{R}$;
- $R(x) \sim \mathcal{GP}(0, k_{\boldsymbol{\theta}}(x, x'))$ is a zero-mean Gaussian Process for the i^{th} patient random effects, with kernel hyperparameters $\boldsymbol{\theta} \in \mathbb{R}^m$;
- $R(Z_i|\boldsymbol{\theta}) \in \mathbb{R}^{n_i}$ is a finite-dimensional realisation of the above Gaussian Process, obtained using the i^{th} patient's random-effects design matrix. Note that in our

case each Z_i would only consist of a column containing the measurement times of each patient, i.e. $Z_i \in \mathbb{R}^{n_i}$;

- Furthermore, we assume that the vectors $R(Z_i|\boldsymbol{\theta})$ and $\boldsymbol{\epsilon}_i$ are uncorrelated, and by Gaussian Process properties we have $R(Z_i|\boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{0}, K_{\boldsymbol{\theta}})$, where $K_{\boldsymbol{\theta}} := K(Z_i, Z_i|\boldsymbol{\theta}) \in \mathbb{R}^{n_i \times n_i}$.

Moreover, we specify that the fixed-effects function $F(\cdot)$ belongs to some normed function space \mathcal{H} , where this space is assumed to be the linear span of a set of weak learners \mathcal{S} . A few examples of such weak learners include decision trees, linear functions and splines.

Let us also highlight a fundamental modelling choice in the GP Boost algorithm. As is the case in our CKD longitudinal patient data, it is assumed that a GP model can capture temporal correlations within the patient-specific level. This can be done *in two different ways*. Firstly, one can assume that for each patient there is a GP to model the patient's random effects, and that each GP is independent of the rest of the patient GPs. Under this setting, *we need to learn a kernel hyperparameter vector $\boldsymbol{\theta}_i$ for each distinct patient*. Alternatively, we can also assume that there is a single common GP that is related to all patient temporal observations, which is the setting we have just described above and where *we only learn a single kernel hyperparameter vector $\boldsymbol{\theta}$* . In our experiments, we refer to these two types of models as GBT with independent GP, and GBT with common GP respectively.

2.3.2 Model fitting

In this section, we provide details on the derivation of the model likelihood in (2.12). To ease with notation, we drop the patient subscripts and further define $\mathbf{u} := R(Z|\boldsymbol{\theta}) \in \mathbb{R}^n$. Let us also use $F = F(X)$ for short, where the notation $F(\cdot)$ denotes the fixed-effects random function, and F denotes the respective function evaluation at the design matrix X .

From the above model specification, we can easily see that $\mathbf{u}|\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, K_{\boldsymbol{\theta}})$, and $\mathbf{y}|\mathbf{u}, \boldsymbol{\theta}, F \sim \mathcal{N}(F, \sigma^2 I)$. Using standard rules of conditional probability, and by marginalising the joint distribution $p(\mathbf{y}, \mathbf{u}|F, \boldsymbol{\theta})$, we obtain the conditional density of the target variable in the following way

$$p(\mathbf{y}|F, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{u}, F, \boldsymbol{\theta})p(\mathbf{u}|\boldsymbol{\theta}) \, d\mathbf{u}, \quad (2.13)$$

such that

$$p(\mathbf{u}|\boldsymbol{\theta}) = (2\pi)^{-n/2} |K_{\boldsymbol{\theta}}|^{-1/2} \exp\left(-\frac{1}{2} \mathbf{u}^\top K_{\boldsymbol{\theta}}^{-1} \mathbf{u}\right),$$

$$p(\mathbf{y}|\mathbf{u}, F, \boldsymbol{\theta}) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2} (\mathbf{y} - F - \mathbf{u})^\top (\mathbf{y} - F - \mathbf{u})\right),$$

where $\boldsymbol{\theta} \in \mathbb{R}^{m+1}$ is now a vector also containing the variance of the likelihood noise term σ^2 , as well as the GP kernel hyperparameters.

Once again, using linear Gaussian math, we obtain a closed-form expression for the above integral, by noting that $\mathbf{y}|F, \boldsymbol{\theta} \sim \mathcal{N}(F, K_{\boldsymbol{\theta}} + \sigma^2 I)$. As a result, the log-likelihood of our model is given by the following expression

$$\mathcal{L}(\boldsymbol{\theta}, F|\mathbf{y}) = -\frac{n}{2} \log(2\pi) - \frac{1}{2} \log(|K_{\boldsymbol{\theta}} + \sigma^2 I|) - \frac{1}{2} (\mathbf{y} - F)^\top (K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} (\mathbf{y} - F). \quad (2.14)$$

Then, in the model fitting process, our end-goal is to minimise the corresponding risk functional, which plays the role of a loss function, and is specified as

$$\mathcal{R}(\cdot, \cdot): \mathcal{H} \times \mathbb{R}^{m+1} \rightarrow \mathbb{R}, \quad (2.15)$$

where this risk functional is defined as $\mathcal{R}(F(\cdot), \boldsymbol{\theta}) := -\mathcal{L}(\boldsymbol{\theta}, F = F(X)|\mathbf{y})$.

Risk functional minimisation for fixed $\boldsymbol{\theta}$

We firstly present material regarding the optimisation of the risk functional objective specified in (2.15), in the simpler case where $\boldsymbol{\theta}$ is assumed to be known in advance. In this setting, Boosting works by empirically minimising the risk functional sequentially, where the m^{th} iteration (amongst a total of M iterations) follows the updating equation given by

$$F_m(\cdot) = F_{m-1}(\cdot) + f_m(\cdot), \quad f_m \in \mathcal{S}, \quad 1 \leq m \leq M, \quad (2.16)$$

such that

$$f_m(\cdot) = \underset{f(\cdot) \in \mathcal{S}}{\operatorname{argmin}} \left\| (K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} (\mathbf{y} - F_{m-1}) - f \right\|^2, \quad (2.17)$$

where $f = (f(X_1), \dots, f(X_n))^\top \in \mathbb{R}^n$. The form of the minimisation objective in (2.17) comes by noting that $f_m(\cdot)$ corresponds to the least-squares vector approxi-

mation to the gradient of the risk functional, evaluated at $F_{m-1}(\cdot)$. More concretely, this gradient expression is given by

$$-\frac{\partial}{\partial F}\mathcal{L}(\boldsymbol{\theta}, F|\mathbf{y})\Big|_{F=F_{m-1}} = (K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}(\mathbf{y} - F_{m-1}).$$

As the optimisation step in (2.17) requires an approximation, it is often the case that the risk functional is approximated using a Taylor expansion around $F_{m-1}(\cdot)$. Essentially, if we take a first-order Taylor expansion, the resulting algorithm can be thought of as gradient descent in the function space \mathcal{H} . As commonly observed in many optimisation schemes, we can further add a learning rate parameter to equation (2.16), to make navigation in the optimisation landscape smoother. Then, this results to the following updating equation

$$F_m(\cdot) = F_{m-1}(\cdot) + \epsilon f_m(\cdot), \quad f_m \in \mathcal{S}, \quad 1 \leq m \leq M, \quad \epsilon > 0.$$

For the interested reader, more technical details regarding the optimisation procedure of the risk functional can be found in [Sigrist, 2018].

Risk functional minimisation for unknown $\boldsymbol{\theta}$

The main idea behind fitting the GP Boost model relies on the idea of combining functional gradient descent steps for updating $F(\cdot)$, with coordinate descent steps for updating $\boldsymbol{\theta}$. To be more precise, at each step of the optimisation procedure, we keep our function $F(\cdot)$ fixed and minimise the risk functional using coordinate descent in $\boldsymbol{\theta}$ space. Then, we keep $\boldsymbol{\theta}$ fixed, and update $F(\cdot)$ using functional gradient descent as described in Section 2.3.2. Even though this process seems rather simple, it has been shown to not only combat issues related to overfitting, but is also computationally efficient. A concise summary of the GP Boost algorithm can be found in Algorithm 1. We state that in the coordinate descent part (step 1 in the for loop), the minimisation process is performed using first or second-order optimisation methods such as vanilla gradient descent, gradient descent with Nesterov acceleration or Fisher scoring. By initialising at the optimal parameter vector found at the previous iteration, we can skip the full estimation of these parameters at each boosting iteration, and dramatically speed things up in terms of computation time. Let us provide expressions regarding the gradient of the negative log-likelihood function with respect to $\boldsymbol{\theta}$. To make expressions more compact, define $\Psi := K_{\boldsymbol{\theta}} + \sigma^2 I$, then we have

$$-\frac{\partial \mathcal{L}(\boldsymbol{\theta}, F|\mathbf{y})}{\partial \boldsymbol{\theta}_k} = -\frac{1}{2}(\mathbf{y}-F)^\top \Psi^{-1} \frac{\partial \Psi}{\partial \boldsymbol{\theta}_k} \Psi^{-1}(\mathbf{y}-F) + \frac{1}{2} \text{Tr} \left(\Psi^{-1} \frac{\partial \Psi}{\partial \boldsymbol{\theta}_k} \right), \quad 1 \leq k \leq m+1,$$

where the term involving the trace operator is given by

$$\text{Tr} \left(\Psi^{-1} \frac{\partial \Psi}{\partial \boldsymbol{\theta}_k} \right) = \sum_{i,j=1}^n [\Psi^{-1}]_{ij} \left[\frac{\partial \Psi}{\partial \boldsymbol{\theta}_k} \right]_{ij}.$$

Algorithm 1 GP Boost: Gaussian Process Boosting

Input: Initial GP hyperparameter values $\boldsymbol{\theta}_0 \in \mathbb{R}^{m+1}$, learning rate $\epsilon > 0$, number of boosting iterations $M \in \mathbb{N}$.

Output: Mean function $\hat{F}(\cdot) = F_M(\cdot)$ and GP hyperparameters $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_M$.

Initialise $F_0(\cdot) \leftarrow \underset{c \in \mathbb{R}}{\text{argmin}} -\mathcal{L}(\boldsymbol{\theta}_0, c \cdot \mathbf{1}|\mathbf{y})$.

for $t = 1 : M$ **do**

1. Find $\theta_t = \underset{\boldsymbol{\theta} \in \mathbb{R}^{m+1}}{\text{argmin}} -\mathcal{L}(\boldsymbol{\theta}, F_{t-1}|\mathbf{y})$, using coordinate descent initialised at $\boldsymbol{\theta}_{t-1}$.
2. Find $f_t(\cdot) = \underset{f(\cdot) \in \mathcal{S}}{\text{argmin}} \| (K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}(\mathbf{y} - F_{t-1}) - f \|^2$, with functional gradient descent.
3. Update $F_t(\cdot) = F_{t-1}(\cdot) + \epsilon f_t(\cdot)$.

end for

2.3.3 Model prediction

We now describe how predictions on test data are made using the GP Boost algorithm. Let $\mathbf{y}_p \in \mathbb{R}^{n_p}$ denote the target vector which we aim to predict. Then, the joint distribution of our training and test data target vectors is given by

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_p \end{pmatrix} = \begin{pmatrix} F(X) \\ F(X_p) \end{pmatrix} + \begin{pmatrix} \mathbf{u} \\ \mathbf{u}_p \end{pmatrix} + \begin{pmatrix} \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon}_p \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} F(X) \\ F(X_p) \end{pmatrix}, \begin{pmatrix} K_{\boldsymbol{\theta}} + \sigma^2 I_n & (K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op}) \\ (K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op})^\top & \begin{pmatrix} K_{\boldsymbol{\theta}} & K_{\boldsymbol{\theta}}^{op} \\ (K_{\boldsymbol{\theta}}^{op})^\top & K_{\boldsymbol{\theta}}^p \end{pmatrix} + \sigma^2 I_{n_p} \end{pmatrix} \right), \quad (2.18)$$

where $\mathbf{u}_p \in \mathbb{R}^{n_p}$ is the random-effects vector, defined as $\mathbf{u}_p := R(Z_p|\boldsymbol{\theta})$, where $Z_p \in \mathbb{R}^{n_p \times m_p}$ is the random-effects design matrix for the test data. Also, $K_{\boldsymbol{\theta}}^p :=$

$K_{\boldsymbol{\theta}}(Z_p, Z_p) \in \mathbb{R}^{n_p \times n_p}$. Finally, $(K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op}) \in \mathbb{R}^{n \times (n+n_p)}$, such that $K_{\boldsymbol{\theta}}^{op} := \text{Cov}(\mathbf{u}, \mathbf{u}_p) \in \mathbb{R}^{n \times n_p}$. The definition of the remaining elements in equation (2.18) that have not been specified follows analogously from the notation previously introduced in this section.

Following from (2.18), using Gaussian conditioning identities we get the predictive distribution

$$\mathbf{y}_p | \mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_p, \Xi_p),$$

where

$$\begin{aligned} \boldsymbol{\mu}_p &= F(X_p) + (K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op})^\top (K_{\boldsymbol{\theta}} + \sigma^2 I_n)^{-1} (\mathbf{y} - F(X)), \\ \Xi_p &= \begin{pmatrix} K_{\boldsymbol{\theta}} & K_{\boldsymbol{\theta}}^{op} \\ (K_{\boldsymbol{\theta}}^{op})^\top & K_{\boldsymbol{\theta}}^p \end{pmatrix} + \sigma^2 I_{n_p \times n_p} - (K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op})^\top (K_{\boldsymbol{\theta}} + \sigma^2 I_n)^{-1} (K_{\boldsymbol{\theta}}, K_{\boldsymbol{\theta}}^{op}). \end{aligned}$$

2.3.4 Vecchia approximation

As already discussed, exact GP inference approaches utilising a full covariance matrix do not scale well with large datasets. Due to the size of the dataset used in our experiments, and when it comes to using a GP Boost model with a common GP across all patients, any form of hyperparameter tuning is practically infeasible. This is because fitting the model just once with exact GP inference techniques requires more than four days of training on a machine with 12 GB of RAM, and so the experiments could not be realistically completed within our time frame. To overcome this issue, we decided to use the Vecchia approximation [Katzfuss and Guinness, 2021], for training GP Boost models with common GPs across all patients. We also state for clarity that exact GP inference in GP Boost models with independent GPs for each patient is much faster, since we invert kernel matrices of much smaller dimensionalities.

The main gist of the Vecchia approximation is to get a sparse precision matrix using an approximation to the Cholesky factor of the corresponding precision matrix. The approximation can be essentially viewed as a particular type of a composite likelihood technique [Varin et al., 2011], where the likelihood function $p(\mathbf{y} | F, \boldsymbol{\theta})$ can be approximated as

$$\begin{aligned}
p(\mathbf{y}|F, \boldsymbol{\theta}) &= \prod_{i=1}^n p(y_i|(y_1, \dots, y_{i-1}), F, \boldsymbol{\theta}) \\
&\approx \prod_{i=1}^n p(y_i|\mathbf{y}_{N(i)}, F, \boldsymbol{\theta}),
\end{aligned} \tag{2.19}$$

where $\mathbf{y}_{N(i)}$ is a subset of $\{y_1, \dots, y_{i-1}\}$, and $N(i)$ is its corresponding subset of indices, i.e. $\mathbf{y}_{N(i)} \subset \{y_1, \dots, y_{i-1}\}$, and $N(i) \subset \{1, \dots, i-1\}$. It is often the case that one selects the index set $N(i)$ to be the indices of the m_v nearest neighbors to the input point of interest s_i . Using Gaussian conditioning identities, we then get

$$p(y_i|\mathbf{y}_{N(i)}, F, \boldsymbol{\theta}) = \mathcal{N}(y_i | F_i + A_i(\mathbf{y}_{N(i)} - F_{N(i)}), D_i),$$

such that $A_i \in \mathbb{R}^{1 \times |N(i)|}$, $D_i \in \mathbb{R}$ and $|N(i)|$ is the cardinality of the index set $N(i)$. A_i and D_i are also specified as

$$\begin{aligned}
A_i &= (K_{\boldsymbol{\theta}})_{i, N(i)} ((K_{\boldsymbol{\theta}} + \sigma^2 I_n)_{N(i)})^{-1}, \\
D_i &= (K_{\boldsymbol{\theta}} + \sigma^2 I_n)_{i, i} - A_i (K_{\boldsymbol{\theta}})_{N(i), i},
\end{aligned}$$

where $K_{\boldsymbol{\theta}}$ represents the covariance matrix of the Gaussian Process specified as $[K_{\boldsymbol{\theta}}]_{ij} = k_{\boldsymbol{\theta}}(s_i, s_j)$, with covariance function $k_{\boldsymbol{\theta}}(\cdot, \cdot)$. Note also that $M_{i, N(i)}$ denotes the sub-matrix of matrix M which consists of M 's i^{th} row and columns $N(i)$. Similarly, by $M_{N(i)}$ we refer to the sub-matrix of M which consists of rows and columns $N(i)$.

Let us also define the matrix B to be a lower triangular matrix consisting of ones on its diagonal, with off-diagonal elements given by

$$(B)_{i, N(i)} = -A_i,$$

and zeros elsewhere. Furthermore, define the matrix D to be a diagonal matrix consisting of the elements D_i . Then, we get the following approximate distribution

$$\mathbf{y} \stackrel{\text{approx.}}{\sim} \mathcal{N}(F(X), B^{-1} D B^{\top}), \tag{2.20}$$

with a precision matrix given by $B^{\top} D^{-1} B$. The significance of this approximate distribution comes from the fact that the Cholesky factor B , and hence the precision matrix, are both sparse.

In the above approximation, the most costly operation is calculating the Cholesky factors of the matrix $(K_{\theta} + \sigma^2 I_n)_{N(i)}$. Therefore, the Vecchia approximation scales in $\mathcal{O}(nm_v^3)$ and $\mathcal{O}(nm_v)$ in terms of computational and memory complexity respectively. This provides a significant improvement over exact GP inference since we can easily see that for a fixed number of neighbors m_v , the computational and memory complexity is linear in the number of data points n . The details regarding the gradient estimation of the approximate log-likelihood used in the Vecchia approximation, as well as the prediction procedure are outside the scope of this thesis. However, we point the interested reader to [Sigrist, 2020] for the full details.

2.4 Time-series clustering

In this section, we provide background material on the time-series clustering methods that will be used and discussed in Chapter 5. Clustering techniques fall under the unsupervised learning class of methods, which aim to identify hidden structure in the data. Ultimately, through clustering, one aims to divide the dataset into distinct and non-overlapping groups, such that “similar” data points belong to the same group (also known as cluster). The “similarity” between two data points can be quantified through a range of different metrics, which we discuss below. Even though clustering can be applied to a wide range of datasets, in the subsequent sections we focus on the special case of time-series clustering, with the overall goal of applying these methods to patient eGFR time-series, as shown in Figure 5.1. Let us state that time-series clustering is somewhat special compared to other types of clustering due to the temporal nature of the data. For example, we might be interested in the overall trend of the time-series, and not the actual values. In addition, it is often the case that time-series data contain individual time-series of different lengths, that are misaligned and potentially of different scales. Blindly applying clustering algorithms to such time-series will clearly yield bad results, if one does not account for the temporal aspect of the data. As such, transformations and pre-processing steps are necessary before applying these clustering algorithms. More details on the specific techniques used are provided in Section 5.1.

2.4.1 Dynamic time warping as a similarity measure

Let us consider two univariate time-series $\mathbf{x} := (x_0, \dots, x_{n-1}) \in \mathbb{R}^n$ and $\mathbf{y} := (y_0, \dots, y_{m-1}) \in \mathbb{R}^m$, of lengths n and m respectively. Even though some of the

most popular similarity metrics include the Euclidean distance, and Pearson correlation, they both require the two time-series to be of the same length. As individual eGFR trajectories in our CKD patient dataset are often of different lengths, and frequently misaligned, these metrics are not suitable for the task. In this section, we focus on Dynamic Time Warping (DTW) as our measure of similarity, which put simply, is a method to find the optimal alignment between two time-series. More formally, the DTW distance between the time-series \mathbf{x} and \mathbf{y} is defined through the following optimisation problem

$$\text{DTW}(\mathbf{x}, \mathbf{y}) = \min_{\pi} \sqrt{\sum_{(i,j) \in \pi} d(x_i, y_j)^2}, \quad (2.21)$$

where $\pi := [\pi_0, \dots, \pi_K]$ is a path with the following properties:

- It contains index pairs, i.e. $\pi_k = (i_k, j_k)$, where $0 \leq i_k \leq n - 1$ and $0 \leq j_k \leq m - 1$;
- The first and last elements of the path are specified as $\pi_0 = (0, 0)$ and $\pi_K = (n - 1, m - 1)$;
- For $k > 0$, two consecutive index pairs on the path, $\pi_k = (i_k, j_k)$ and $\pi_{k-1} = (i_{k-1}, j_{k-1})$, are related as $i_{k-1} \leq i_k \leq i_{k-1} + 1$ and $j_{k-1} \leq j_k \leq j_{k-1} + 1$.

Note also that in equation (2.21), the distance metric used is the Euclidean distance. Intuitively, the DTW distance between two time-series can be seen as the corresponding temporal alignment of the two time-series such that the Euclidean distance between the aligned time-series is minimised. Figure 2.3 shows the optimal path found for the two time-series displayed in the figure. Strictly speaking, DTW is a pseudo-metric, since it does not satisfy the triangle inequality. Finally, we state that DTW can be efficiently computed using dynamic programming approaches, with a time complexity in $\mathcal{O}(nm)$ [Sakoe and Chiba, 1978], which involve minimising the cost of sub-paths in the corresponding similarity matrix.

2.4.2 Time-series k -Means algorithm

The k -Means algorithm belongs to the class of partitioning clustering methods, in which k non-overlapping clusters are formed in such a way that at least one element belongs to each cluster. The concept behind the k -Means algorithm is that it works by minimising the sum of squared distances between the cluster centres and

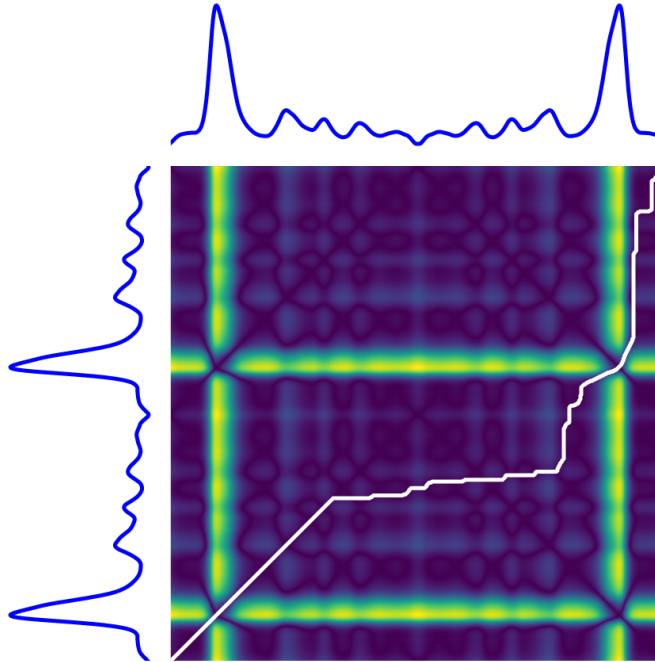


Figure 2.3: Optimal path (in white) superimposed on a similarity matrix of Euclidean distances between time-series points.

all data points belonging to each of the clusters. The notion of averaging is hence fundamental to this algorithm, as it relies on repeatedly computing the cluster averages (equivalently known as centroids) until the cluster assignments do not change. However, when the data is of time-series nature, and the similarity between two time-series is quantified using the DTW pseudo-metric, the task of properly averaging is non-trivial since it has to be consistent with the fact that the DTW pseudo-metric is based on the idea of realigning time-series. Below, we present a high-level overview of how the centroid calculation averaging procedure works in the context of *time-series k-Means*.

DTW Barycenter Averaging (DBA) is a global averaging technique which, for a given cluster, involves refining an average sequence (can be thought of as a centroid) in an iterative fashion, to minimise the squared DTW distances to the rest of the sequences in that cluster. Let us introduce the concept of a barycenter, which is defined as the average vector over a set of associated sequences (viewed as objects in Euclidean space), where association is measured using some pre-specified method. In DBA, one tries to estimate the coordinates of the average sequence as the barycenters of their associated coordinates in the remaining sequences. Once all the coordinate barycenters have been estimated, we obtain the refined average sequence. In this

way, the Within-Cluster Sum of Squares (WCSS) is minimised.

To make sure the reader gets this clear, we summarise the DBA procedure as follows, at every DBA iteration (where the total number of DBA iterations is user-defined):

- We calculate the DTW distance between the average sequence to be refined and every other sequence, and estimate the associations between the coordinates of the average sequence and the coordinates of the remaining sequences;
- Then, we update the coordinates of the average sequence using the barycenters of the associated coordinates of the rest of the sequences.

More technical details regarding the initialisation of the average sequence, convergence guarantees and the complexity analysis of DBA can be found in [Petitjean et al., 2011].

Now that we understand how averaging is performed using DBA for time-series data, we also provide some technical details on the well-known k -Means algorithm. Let us be clear in that time-series k -Means is different from the standard k -Means algorithm in the sense that the centroid estimation procedure is performed using DBA.

Given an ordered set of time-series sequences $(\mathbf{x}_1, \dots, \mathbf{x}_n)$, where different sequences can be of varying lengths, the goal of k -Means clustering is to partition the sequences in $k \leq n$ clusters, $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$. The partitioning process is done in a way that minimises the WCSS objective, which is more formally expressed as

$$\operatorname{argmin}_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \operatorname{DTW}(\mathbf{x}, \boldsymbol{\mu}_i)^2,$$

where $\boldsymbol{\mu}_i$ is the average sequence of S_i , obtained using the DBA algorithm.

Then, using an initial set of average sequences $(\boldsymbol{\mu}_1^{(1)}, \dots, \boldsymbol{\mu}_k^{(1)})$, we repeatedly alternate between the two below steps, where on iteration t the steps are specified as

1. **Assignment step:** Assign each sequence to the cluster whose centroid is closest to that sequence in terms of DTW distance. More rigorously, the i^{th} cluster is given by

$$S_i^{(t)} = \left\{ \mathbf{x}_p : \operatorname{DTW}(\mathbf{x}_p, \boldsymbol{\mu}_i^{(t)})^2 \leq \operatorname{DTW}(\mathbf{x}_p, \boldsymbol{\mu}_j^{(t)})^2, \quad 1 \leq j \leq k \right\}$$

2. **Update step:** Re-estimate the cluster centroids $(\boldsymbol{\mu}_1^{(t+1)}, \dots, \boldsymbol{\mu}_k^{(t+1)})$ using the DBA algorithm.

The algorithm is said to converge when cluster assignments stop changing. Let us also note that one of the weaknesses of this algorithm is that it is not guaranteed to find the global optimum, since the resulting cluster assignments can be highly sensitive to the centroids initialisation. As such, a common strategy is to run this algorithm repeatedly from many different initialisation points.

k -Medoids clustering

Let us now also briefly mention a few things regarding the k -Medoids algorithm [Kaufmann, 1987], which like k -Means, belongs to the class of partitioning clustering methods. The concept of a medoid in a cluster is given by a point whose mean similarity between all other points in that cluster is maximal, and intuitively can be thought of as the cluster point which is more “centrally” found. Although quite similar conceptually, the main difference between the two methods is that the k -Medoids algorithm is more interpretable as it chooses actual data points for the cluster centres, instead of using an average data point as in k -Means. In addition, another important difference is that the k -Medoids algorithm can be used with any similarity measure, such as DTW, and so unlike the k -Means algorithm, no modifications are required when working with time-series data, so long as a well-defined similarity matrix is provided. Even though we will not go into the exact details of how the algorithm works, we state that the solutions to the k -Medoids problem are often heuristic, as the problem is NP-hard when it comes to finding an exact solution. One such heuristic solution example is the Partitioning Around Medoids (PAM) algorithm.

2.4.3 Hierarchical agglomerative clustering

The goal in Hierarchical clustering is to order clusters in a hierarchical manner, by sequentially merging similar clusters. Hierarchical clustering approaches can be separated in two types of methods, agglomerative and divisive. In this thesis, we only use the agglomerative type, in which every data point starts as its own cluster and pairs of clusters are sequentially merged as we move up the hierarchy. A key feature of hierarchical agglomerative clustering (HAC) is a dendrogram, which displays the hierarchy of the clusters, and can be used to form the final clusters. On the x -axis and y -axis of a dendrogram we can see the data point ID and the similarity between the clusters respectively. We highlight that the similarity between two clusters de-

depends on the *distance* function (e.g. Euclidean or DTW distance) and the *linkage* criterion used. A merge of two clusters is represented by the splitting of a vertical line into two vertical lines in the dendrogram. An example of a dendrogram is visualised in Figure 2.4. Due to the hierarchical nature of a dendrogram, the dataset can be split into any number of desired clusters according to the distance between them.

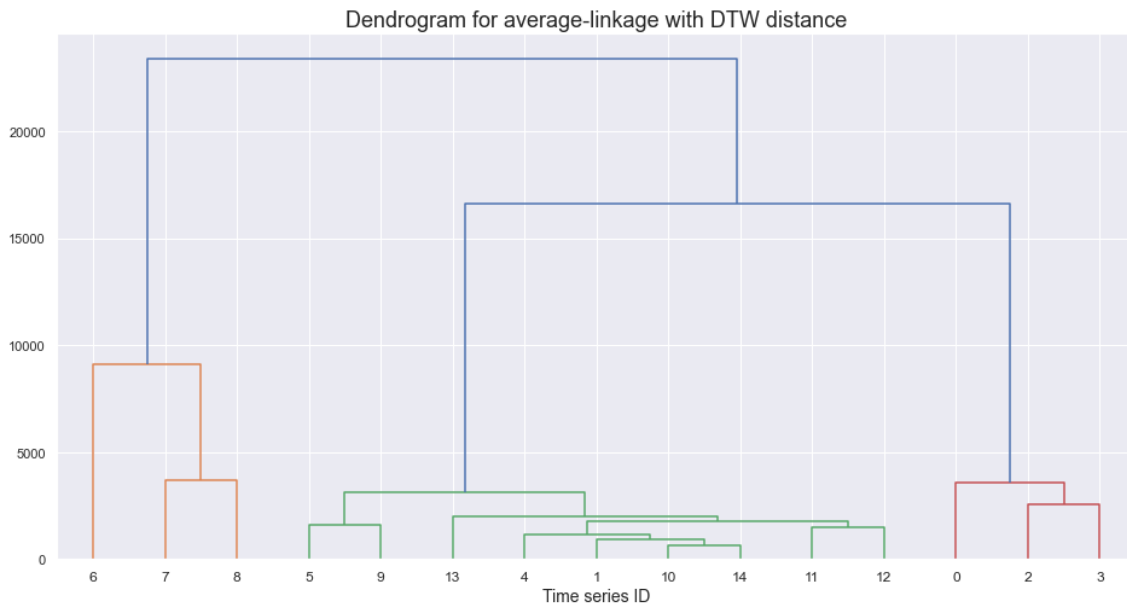


Figure 2.4: Dendrogram example in Hierarchical agglomerative clustering, where each color corresponds to a distinct cluster.

As already mentioned, in HAC all data points start as their own cluster, and then we iteratively merge the pair of clusters which are more “similar” to each other in a greedy manner. A fundamental concept in HAC is the idea of a linkage criterion, which measures the similarity of clusters as a function of the pairwise distances of the data points in those clusters. Some of the main types of linkage criteria commonly used include:

1. **Minimum/Single-linkage:** Using this type of linkage, cluster similarity is quantified as the minimum distance between points belonging to the two different clusters considered. In particular, the distance between two clusters A and B is given by $d_{SL}(A, B) := \min\{d(a, b) : a \in A, b \in B\}$, where the function $d(\cdot, \cdot)$ denotes our chosen distance metric. The rationale behind the single-linkage criterion is quite simple, data points that are close to each other must belong to the same cluster. A drawback of using this linkage criterion

is that clusters that are too big may be formed, which leads to loss of cluster interpretability.

2. **Maximum/Complete-linkage:** This type of linkage approach is based on the idea of quantifying cluster similarity using the least similar data points in two different clusters, and is expressed as $d_{CL}(A, B) := \max\{d(a, b) : a \in A, b \in B\}$. An issue with this linkage type though is that the clusters formed are sometimes too conservative, and do not unfold the structure in the data to a desired extent.
3. **Average-linkage:** Average-linkage can be seen as a middle ground measure of similarity of the two previous linkage functions, in which cluster similarity is quantified using the distance between the two cluster centres, $d_{AL}(A, B) := \frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$.
4. **Ward-linkage:** Finally, this linkage criterion aims at minimising the total within-cluster variance. To be exact, at each merging stage, we select to merge the pair of clusters that will lead to the smallest increase in the total within-cluster variance.

We end this section by noting that a discussion on clustering quality evaluation is included in Section 5.1.

2.5 Bayesian neural networks

Our last topic in this chapter is Bayesian neural networks (BNNs) [Goan and Fookes, 2020], a class of stochastic neural networks trained using Bayesian inference techniques. The widespread accessibility to computing power in the last decade has made the use of increasingly “deeper” neural network models more popular. However, these models yield predictions with unreasonably high confidence, even on data that is significantly different from the training set [Kristiadi et al., 2020]. BNNs are centred around the fundamental Bayesian paradigm, which is a useful framework for developing models with uncertainty quantification capabilities, and can possibly fix issues related to prediction overconfidence.

The main aim in deep learning is to learn arbitrary functional representations for mapping input vectors to their corresponding outputs, over a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, such that each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Equivalently, we can represent this dataset as $\mathcal{D} = \{X, \mathbf{y}\}$. Classical deep learning models, such as feed-forward

neural networks, consist of an input layer, consecutive hidden layers, and an output layer. It is often the case that each layer is specified as a linear mapping, followed by a non-linear activation function. Let us denote a neural network model by $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ represents the model parameters (which include the weights and biases of each layer). Usually, to find an estimate of the model parameters $\hat{\boldsymbol{\theta}}$, one defines an appropriately chosen loss function, which often corresponds to the negative log-likelihood function over the training set. The estimate $\hat{\boldsymbol{\theta}}$ is then obtained by minimising the specified loss function using gradient-based optimisation algorithms, which utilise the well-known backpropagation algorithm. In statistical jargon, this corresponds to the method of maximum likelihood estimation (MLE). A common pitfall of training neural networks using the MLE approach is that it can lead to overfitting, and result in models with bad generalisation properties on unseen data.

Regarding the specification of a BNN, one needs to choose a functional model, also known as the network architecture, which we will not be discussing here. In addition, a BNN also requires a stochastic component, which includes the model parameters' prior distribution and the model likelihood, denoted by $p(\boldsymbol{\theta})$ and $p(\mathbf{y}|X, \boldsymbol{\theta})$ respectively. Then, inference over the model parameters can be performed by applying Bayes' theorem as follows

$$p(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{\theta}|\mathbf{y}, X) = \frac{p(\mathbf{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y}|X)} = \frac{p(\mathbf{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathbf{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta}) \, d\boldsymbol{\theta}}.$$

The posterior distribution in deep learning models is frequently very complex, due to its high dimensionality. Additionally, the high-dimensional integral present in the denominator of the above expression is typically intractable, which renders exact posterior inference impossible. Hence, one addresses the task of posterior inference through either Markov chain Monte Carlo (MCMC) or Variational Inference techniques. Before going deeper in these techniques, let us briefly discuss how predictions are made in BNNs. Predictions on unseen data revolve around the posterior predictive distribution, which provides a way to incorporate the model's uncertainty from the posterior distribution in its predictions. In particular, and assuming we have access to samples from the posterior distribution $p(\boldsymbol{\theta}|\mathbf{y}, X)$, the posterior predictive distribution about a test point (\mathbf{x}^*, y^*) is given by

$$\begin{aligned}
p(y^*|\mathbf{y}, X, \mathbf{x}^*) &= \int p(y^*|\mathbf{x}^*, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y}, X) \, d\boldsymbol{\theta} \\
&\approx \frac{1}{N} \sum_{i=1}^N p(y^*|\mathbf{x}^*, \boldsymbol{\theta}^{(i)}),
\end{aligned} \tag{2.22}$$

where $\{\boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(N)}\}$ are samples from the posterior.

We end this section by mentioning that it is common practice to only use a small number of stochastic layers in a BNN architecture, especially towards the end of the network. This is because it has been noticed that the structure of deep learning models makes it possible to sufficiently capture uncertainty in the model parameters by only including a few stochastic layers. In this way, inference and prediction can be dramatically improved in terms of computational cost, while maintaining the attractive properties of BNNs with all layers being stochastic. Specific to this thesis, and the work of Chapter 5, we introduce the Bayesian last-layer neural network model, which only uses a single stochastic layer at the end of the network. Intuitively, for an L -layer network, one could view this as learning a fixed feature map using the first $L - 1$ layers, followed by a Bayesian linear classifier. Posterior inference in this setting would only be over the last-layer weights, which significantly reduces the dimensionality of the problem. Below, we discuss two posterior inference techniques commonly used in BNNs, which are employed in Chapter 5.

2.5.1 Laplace approximation

The Laplace approximation is a local approximation to the posterior distribution, in the form of a Gaussian distribution centred at the true posterior mode. Without getting into the details of the derivation of the Laplace approximation, we give the general idea. The log-posterior distribution of the model parameters is specified as

$$\log p(\boldsymbol{\theta}|\mathcal{D}) = \sum_{i=1}^n \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{constant}, \tag{2.23}$$

where the constant term includes any term that does not depend on $\boldsymbol{\theta}$. Define the Maximum a Posteriori estimate, $\boldsymbol{\mu} := \boldsymbol{\theta}_{\text{MAP}}$, and the inverse Hessian of the negative log-posterior evaluated at the mode, $\Sigma := (-\nabla^2 \log p(\boldsymbol{\theta}|\mathcal{D})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\text{MAP}}})^{-1}$. Then, one can obtain a Gaussian approximation to the posterior distribution, by setting $p(\boldsymbol{\theta}|\mathcal{D}) \approx$

$\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \Sigma)$. The advantage of this method is its conceptual simplicity, and that estimates for $\boldsymbol{\mu}$ and Σ can be obtained relatively easily. However, it can be a poor match to the true posterior if it is far from Gaussian. Another concern is that computing the Hessian matrix in high-dimensional problems can be unpleasant, and so fast approximations, such as the Kronecker-factorised Hessian approximation, are often used in deep learning applications.

2.5.2 Stochastic gradient Langevin dynamics

The second inference technique we explore is stochastic gradient Langevin dynamics (SGLD), which belongs to the class of stochastic Markov chain Monte Carlo techniques, and is used to obtain samples from the true posterior distribution. Traditional MCMC methods such as Hamiltonian Monte Carlo have bad scalability properties, as each MCMC iteration requires multiple gradient evaluations over the whole dataset. This makes them unattractive in modern deep learning applications, where the size of the datasets can be large. SGLD originates from the Langevin Monte Carlo (LMC) method [Grenander and Miller, 1994], and makes use of stochastic gradients obtained using mini-batches of the dataset. In essence, this method combines characteristics from the famous stochastic gradient descent optimisation algorithm, and the Langevin dynamics model, a stochastic differential equation used to sample from a distribution. In the notation we have been using so far, the Langevin dynamics model is specified by

$$d\boldsymbol{\theta}_t = \frac{1}{2} \nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{\theta}_t|\mathcal{D}) + d\mathbf{B}_t,$$

where \mathbf{B}_t is a Brownian motion. The key idea behind this model, is to encourage the dynamics to spend more time in regions of high probability density, while also exploring other regions in the parameter space.

As the Langevin dynamics model is a continuous-time model, the LMC algorithm makes use of the following first-order Euler discretised simulation

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \frac{\epsilon}{2} \nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{\theta}_t|\mathcal{D}) + \boldsymbol{\eta}_t,$$

where $\epsilon > 0$ is a suitably chosen step size, and $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \epsilon I)$. However, using finite step sizes introduces errors in the discretisation process, which are in turn accounted for using a Metropolis-Hastings acceptance-rejection step. We note that as

the step size tends to zero, the discretisation error vanishes and hence the acceptance rate goes to 1. This implies that for step sizes approaching zero, one can skip the acceptance-rejection step and still have a theoretically correct sampler.

Moving on, the SGLD method combines the Langevin dynamics model with stochastic gradients, computed over a mini-batch of the dataset. For a mini-batch of size m , where $m < n$, the resulting SGLD updating equation then becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \frac{\epsilon_t}{2} \left(\frac{n}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}_t} \log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{\theta}_t) \right) + \boldsymbol{\eta}_t, \quad (2.24)$$

where $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \epsilon_t I)$. The term in brackets follows from equation (2.23), and by noting that $\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{\theta}_t | \mathcal{D}) \approx \frac{n}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}_t} \log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{\theta}_t)$.

Taking a closer look at the above equation, we notice that this update corresponds to stochastic gradient ascent with an additive isotropic Gaussian noise component. We also mention that most of the computational cost in the Langevin dynamics model simulation comes from computing the gradients. Thus, if the mini-batch gradient approximation of (2.24) is close to the true gradient, then this algorithm will perform well. An important part of the algorithm is that for convergence to be guaranteed, we need to use a sequence of step sizes ϵ_t that slowly decrease over time. In this way, the Metropolis-Hastings acceptance probability will be approaching 1, and this allows us to skip the expensive acceptance-rejection step. Finally, we state that for large t , $\boldsymbol{\theta}_t$ sampled according to equation (2.24) can be regarded as samples from the true posterior distribution.

Chapter 3

Experiments on Synthetic Longitudinal Data

This chapter serves as a stepping-stone to tackling the CKD patient progression forecasting problem using our first approach, which corresponds to solving the eGFR regression problem. Our main aim is to evaluate the two types of mixed-effects models introduced in Chapter 2, namely, linear mixed-effects models, the GP Boost model, and a few different variants of them. We do this by analysing their performance on synthetic data, under a range of different data-generating settings. In turn, the insights drawn from these experiments provide meaningful information as to when might each model be useful, and what its potential flaws are. Based on our analysis, we propose a novel hypothesis on improving the learning process in the GP Boost algorithm by adopting a Bayesian approach, and in turn experimentally evaluate it on real data in Chapter 4.

3.1 Experimental setup

3.1.1 Data-generating process

We begin this section by describing the choice of the different data settings used throughout our experiments, and justify the rationale behind them. Firstly, let us mention that the experiments are performed in two distinct data regimes, the *full* and *sparse* data regime. The essential difference between these two settings is *the number of data points available per individual patient*. In particular, under these two settings we aim to investigate how the performance of the GP Boost model varies as a function of the per patient data points number, and compare it with the

parsimonious linear mixed-effects models. As already mentioned, one of the major challenges in accurate eGFR modelling is that in many cases, there are only a few data points for each individual patient. Hence, in our sparse data regime, we try to incorporate this aspect in our data-generating process. Now, and by keeping in mind that these two data regimes only differ in the per patient number of data points available (denoted by n below), we describe the data-generating process **for the i^{th} patient**, as follows

$$\mathbf{y}_i = F(X_i) + R_i(\mathbf{t}_i) + \boldsymbol{\epsilon}_i \quad (3.1)$$

where

- $n \in \mathbb{N}$ denotes the number of data points of the i^{th} patient, and is the same across all patients. In our experiments, we will be using $n = 10$ and $n = 40$, for the sparse and full data regimes respectively;
- $\mathbf{y}_i \in \mathbb{R}^n$ denotes the target vector of the i^{th} patient (this would correspond to the eGFR values in our experiments on the real dataset in Chapter 4);
- $\mathbf{t}_i \in \mathbb{R}^n$ denotes the random-effects covariate of the i^{th} patient, which in this case is just a vector containing the (unevenly sampled) *sorted times* of the patient measurements. Specifically, $\mathbf{t}_{ij} \stackrel{\text{iid}}{\sim} \text{Uniform}[0, 700]$, sampled without replacement, for $1 \leq j \leq n$. With this random-effects covariate choice, we aim to capture temporal correlations in the data, which makes sense as we are trying to model CKD progression over time. Note that the interval $[0, 700]$ is chosen specifically to make the data-generating process as close as possible to the actual dataset observation times;
- $X_i \in \mathbb{R}^{n \times 4}$ denotes the fixed-effects covariates matrix of the i^{th} patient. Note that for a fixed time-point t , $X_{it} := (x_{it,1}, x_{it,2}, x_{it,3}, x_{it,4})^\top \in \mathbb{R}^4$. In the real CKD dataset, these fixed-effects covariates could correspond to continuous variables such as body mass, height or blood pressure;
- $F(X_i) \in \mathbb{R}^n$ is the fixed-effects vector of the i^{th} patient, such that $F(X_i) := (F(X_{i1}), \dots, F(X_{in}))^\top \in \mathbb{R}^n$ is the row-wise evaluation of the fixed-effects function $F: \mathbb{R}^4 \rightarrow \mathbb{R}$. More details on the fixed-effects functions used will be provided below;
- $R_i(\mathbf{t}_i) := a_i \cdot \mathbf{1}_n + \mathbf{b}_i$, is the random-effects vector of the i^{th} patient, where $a_i \sim \mathcal{N}(0, 1)$ and $\mathbf{b}_i \sim \mathcal{N}(\mathbf{0}_n, K_{\text{sq-exp}}(\mathbf{t}_i, \mathbf{t}_i))$. In particular, each patient has its

own random intercept, but also a random vector \mathbf{b}_i generated from a zero-mean Gaussian Process with the squared-exponential kernel function. We provide more details on how \mathbf{b}_i is generated below;

- $\boldsymbol{\epsilon}_i \in \mathbb{R}^n$ denotes the random errors vector of the i^{th} patient, and is such that $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}_n, 0.25I_n)$.

Let us also provide more details on the fixed-effects function F and the underlying Gaussian Process which generates \mathbf{b}_i . Starting with the function F , we generate data under a linear and a non-linear function, in an attempt to investigate the performance of the models under different fixed-effects linearity assumptions. These fixed-effects functions have been extensively used in designing experiments for mixed-effects models [Hajjem et al., 2014].

The *linear case* is specified as

$$F_{\text{lin}}(X_{it}) := C \cdot (1 + x_{it,1} + x_{it,2} + x_{it,3} + x_{it,4}), \quad (3.2)$$

where $x_{it,j} \stackrel{\text{iid}}{\sim} \text{Uniform}(0, 1)$ for $1 \leq j \leq 4$. Moving on to the *non-linear case*, we have

$$F_{\text{non.lin}}(X_{it}) := C \cdot \tan^{-1} \left(\frac{x_{it,2} \cdot x_{it,3} - (x_{it,2} \cdot x_{it,4})^{-1} - 1}{x_{it,1}} \right), \quad (3.3)$$

where $x_{it,1} \sim \text{Uniform}(0, 100)$, $x_{it,2} \sim \text{Uniform}(40\pi, 560\pi)$, $x_{it,3} \sim \text{Uniform}(0, 1)$, $x_{it,4} \sim \text{Uniform}(1, 11)$.

The constant C is chosen such that the variance of the values of F are approximately one, which implies that the fixed and random effects have roughly the same signal strength.

As mentioned towards the end of Section 2.3.1, the random-effects vector \mathbf{b}_i can be generated in two different ways. Starting with the simpler case where the patient random-effects vectors come from a GP *common across all patients* in the data-generating process, we have

$$\mathbf{b}_i \sim \mathcal{N}(\mathbf{0}_n, K_{\boldsymbol{\theta}}(\mathbf{t}_i, \mathbf{t}_i)), \quad (3.4)$$

where $K_{\boldsymbol{\theta}}(\mathbf{t}_i, \mathbf{t}_i) \in \mathbb{R}^{n \times n}$ is the kernel matrix, with a squared-exponential kernel function, with hyperparameters $\boldsymbol{\theta}$ that are *shared across all patients*, whose $(i, j)^{th}$ element is specified as $[K_{\boldsymbol{\theta}}(\mathbf{t}_i, \mathbf{t}_i)]_{ij} := k(t_{ii}, t_{ij})$, and the kernel function $k(\cdot, \cdot)$ is as

expressed in equation (2.4). Furthermore, and using the notation from (2.4), in the *common GP* case we set $\boldsymbol{\theta} := (l, \sigma^2)^\top = (0.1, 1)^\top$.

Although conceptually very similar to the common GP case, the case where each patient's random-effects vector is generated from *an independent GP* is specified as

$$\mathbf{b}_i \sim \mathcal{N}(\mathbf{0}_n, K_{\boldsymbol{\theta}_i}(\mathbf{t}_i, \mathbf{t}_i)). \quad (3.5)$$

Notice that everything is the same as before, except that now *each GP has its own kernel hyperparameters*, $\boldsymbol{\theta}_i := (l_i, \sigma_i^2)^\top$. Finally, we note that for each individual GP, the kernel hyperparameters are sampled as follows, $l_i \stackrel{\text{iid}}{\sim} \text{Uniform}(0.01, 1000)$ and $\sigma_i^2 \stackrel{\text{iid}}{\sim} \text{Inverse Gamma}(1, 10)$.

In Figures 3.1 and 3.2, we can visualise how the random-effects vectors of our data-generating process using a *common GP* on the time interval $[0, 100]$ look, under different kernel hyperparameter settings.

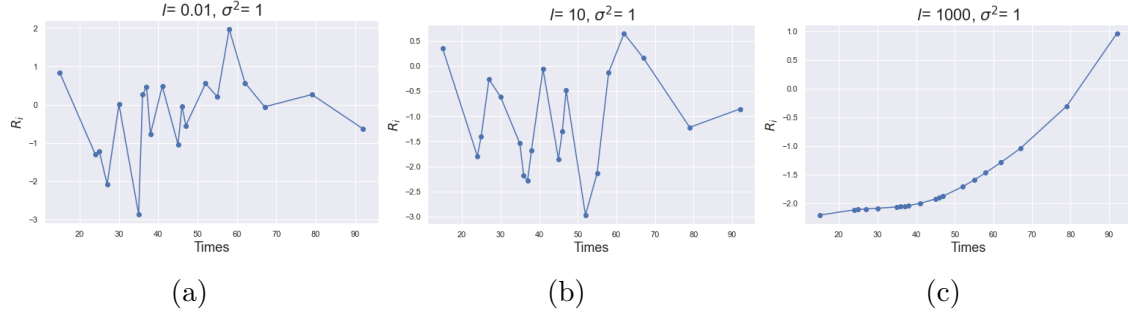


Figure 3.1: Random-effects vectors $R_i(\mathbf{t}_i)$ across different *kernel lengthscales* values under a common GP.

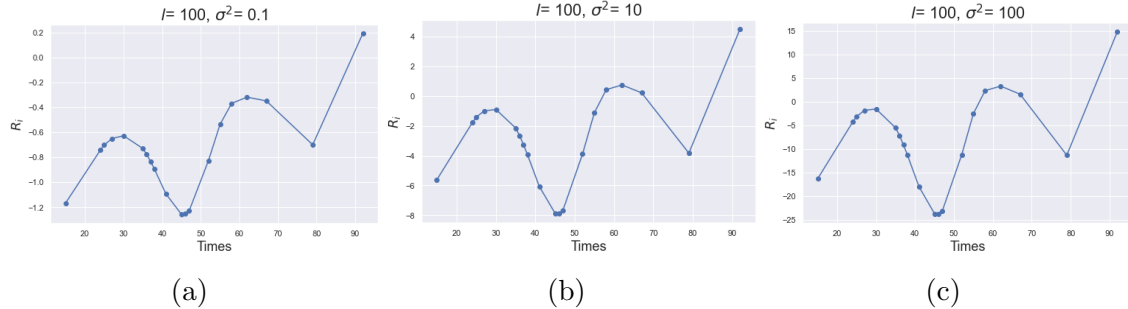


Figure 3.2: Random-effects vectors $R_i(\mathbf{t}_i)$ across different *kernel marginal variance* values under a common GP.

In the following experiments, we will consider models that either use a common or an independent GP across patients for capturing the random-effects part of the

data, in order to allow for greater flexibility in the modelling process. We hope that these models result in more accurate and tailored predictions according to the specific characteristics of each patient. Thus, we aim to compare their predictive performance in settings that reflect each patient’s idiosyncrasies, hence why we choose to generate the random-effects vectors using these two distinct ways.

To re-iterate, *for each data regime*, we have the following four different experiment settings:

1. **Exp1**: Linear fixed-effects function F as in (3.2), with a random-effects vector in which \mathbf{b}_i comes from a GP common to all patients, as in (3.4).
2. **Exp2**: Non-linear fixed-effects function F as in (3.3), with a random-effects vector in which \mathbf{b}_i comes from a GP common to all patients, as in (3.4).
3. **Exp3**: Linear fixed-effects function F as in (3.2), with a random-effects vector in which \mathbf{b}_i comes from a GP which is independent for each patient, as in (3.5).
4. **Exp4**: Non-linear fixed-effects function F as in (3.3), with a random-effects vector in which \mathbf{b}_i comes from a GP which is independent for each patient, as in (3.5).

We finish this section by mentioning again that in the sparse data regime, we generate a total of *10 data points per patient*, for a total of 50 distinct patients. Similarly, for the full data regime we generate a total of *40 data points per patient*, again for a total of 50 distinct patients.

3.1.2 Performance evaluation

For our experiments, using the procedure described in 3.1.1 we simulate twenty different train and test datasets, for which the whole model fitting and prediction pipeline is repeated in order to obtain mean and standard deviation test set scores. In addition, we also generate five more train and test datasets, which are used for the purpose of hyperparameter selection, as outlined in 3.1.3. An important thing to note is that due to the temporal nature of our longitudinal data setting, we evaluate our model performance across *two distinct types of non-overlapping test sets*, specified as:

1. **“Test set 1”**, which corresponds to test set data from patients that we have *observed in our training set*. The time-series aspect of the data requires that test set 1 data points belong to the “future”, and thus need to be further along in time compared to data points belonging to the training set. For example, consider the i^{th} patient with n_i data points in total, and denote the index at which we split this patient’s data between training and test set 1 by j , where $1 < j < n_i$ is chosen at random. Then, the first j data points $[1, 2, \dots, j]$ would belong to the training set, and data points $[j + 1, j + 2, \dots, n_i]$ would belong to test set 1. The idea behind this type of testing technique is to see how accurately a model can predict the future for a particular patient on which *the model has been trained on*.
2. **“Test set 2”**, which consists of data points from patients that we *did not observe in our training data*. Continuing from our above example, if patient i was selected to be part of test set 2, then all data points $[1, 2, \dots, n_i]$ would belong to test set 2, and no data points for this patient would be found in the training set. The rationale behind this type of testing technique is to check how well a model trained on data from other patients can *generalise to patients that it has not been trained on*.

For each dataset generated, we include 60% of the data in the training set, and 20% of the data to each of the two types of test sets described above. An obvious but important clarification is that for patient data belonging to test set 2 (and are not in the training set at all), and in cases where the model predicts the random-effects using patient-specific coefficients, test set 2 predictions are made *using just the fixed-effects part* of the model, since no estimates for those patient-specific coefficients exist. For evaluating the predictive performance of our models, we consider the root-mean-square error (RMSE) of the model predictions on *each of the two test sets individually*. More formally, the RMSE over a set of n predictions is defined as

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

3.1.3 Models specification

Let us start by mentioning that the mixed-effects models used in this thesis were developed using the GPBoost Python package, specifically version 0.7.6.2. At the time of the experiments, this package did not support random slope models, which

justifies why the mixed-effects models we consider below do not include random slopes for their random-effects parts. To avoid any confusion, we state again that for estimating parameters related to the fixed-effects parts of the models, we use the fixed-effects covariate data X . Similarly, for the random-effects parameters, we only use a single covariate, the times of the patient measurements \mathbf{t} .

Throughout our experiments, we compare the following eight models, where models 1-4 and 5-8 belong to the class of linear mixed-effects and GP Boost type models respectively:

1. **Intercept only model:** This is our simplest model and serves as a benchmark for performance. In essence, this model is a random slope model, where the prediction for the i^{th} patient at time t is specified as $\hat{y}_{it} = \bar{y} + \hat{a}_i$, where \bar{y} denotes the mean target value over the training set, and \hat{a}_i is the patient-specific intercept. Note that when predicting for patients in test set 2, the prediction is simply $\hat{y}_{it} = \bar{y}$, as we do not have estimates for patient-specific coefficients for those patients not observed during training time.
2. **Linear model with random intercept:** This model is specified as $\hat{y}_{it} = X_{it}^\top \hat{\beta} + \hat{a}_i$, where the estimated coefficients vector $\hat{\beta}$ is shared across all patients. Again, for test set 2 patient prediction, we have $\hat{y}_{it} = X_{it}^\top \hat{\beta}$.
3. **Linear model with common Gaussian Process:** Let $f(t|\hat{\theta})$ denote a GP posterior, where the GP kernel hyperparameters $\hat{\theta}$ are estimated using the random-effects covariate over *all patients*. Then, we specify this model as $\hat{y}_{it} = X_{it}^\top \hat{\beta} + f(t_i|\hat{\theta})$. Note that we can simply take the GP posterior mean at test time t_i as our point estimate for the random-effects part of the prediction.
4. **Linear model with independent Gaussian Process:** Let $f_i(t|\hat{\theta}_i)$ denote a GP posterior, where $\hat{\theta}_i$ is estimated using *the i^{th} patient* random-effects covariate. Then, we specify this model as $\hat{y}_{it} = X_{it}^\top \hat{\beta} + f_i(t_i|\hat{\theta}_i)$. Note that we can simply take the GP posterior mean at test time t_i as our point estimate for the random-effects part of the prediction.
5. **GBT with ID as categorical variable:** This model serves as an alternative approach to mixed-effects modelling, in which the model only consists of a fixed-effects part. The random effects are captured by including the grouping variable of the data, which is the ‘‘Patient ID’’ in our case, and treating it as

a categorical variable. This model is specified as $\hat{y}_{it} = \hat{F}(\tilde{X}_{it})$, where \tilde{X}_{it} is a modified fixed-effects covariates vector which also includes the “Patient ID” categorical variable. \hat{F} denotes the fitted gradient boosted tree (GBT) model.

6. **GBT with random intercept:** Extending the second model with a non-linear fixed-effects function, we specify this model as $\hat{y}_{it} = \hat{F}(X_{it}) + \hat{a}_i$, where model fitting and test set 2 patient prediction follow directly from the linear case.
7. **GBT with common Gaussian Process:** Extending the third model with a non-linear fixed-effects function, we specify this model as $\hat{y}_{it} = \hat{F}(X_{it}) + f(t_i|\hat{\theta})$, where model fitting and test set 2 patient prediction follow directly from above.
8. **GBT with independent Gaussian Process:** Finally, extending the fourth model with a non-linear fixed-effects function, we specify this model as $\hat{y}_{it} = \hat{F}(X_{it}) + f_i(t_i|\hat{\theta}_i)$, where model fitting and test set 2 patient prediction follow directly from above.

We end the model specification section by noting that for the dataset sizes we consider in our synthetic data experiments, we can perform exact GP inference without needing to resort to the Vecchia approximation. However, when we move on to our experiments on the real dataset, the prohibitively large dataset size will render the use of the Vecchia approximation necessary.

Hyperparameter selection

We now outline the details related to the tuning of the models introduced above. For the GP models, the covariance function used was the squared-exponential, and kernel hyperparameter optimisation is performed by maximising the marginal likelihood as described in 2.2.3. Also, it is clear that linear mixed-effects models do not have any hyperparameters, and so tuning is only performed for the GP Boost models. For the GP Boost algorithm hyperparameters, we consider tuning the number of boosting iterations M , and the learning rate ϵ as specified in Algorithm 1, where we try values in $\{1, \dots, 1000\}$ and $\{0.01, 0.10, 0.50, 1\}$ respectively. As for the gradient boosted tree part in GP Boost, we only consider tuning two of the most important hyperparameters for computational reasons. These are the minimum number of data points in the trees’ leaf nodes and the maximum depth of the trees, where we try values in $\{1, 10, 20, 50\}$ and $\{1, 2, 3, 4, 5, 10, 50\}$ respectively. Finally, the tuning is performed through exhaustive grid-search using five-fold cross-validation, with the

average five-fold RMSE over both test sets 1 and 2 as the evaluation metric for the optimal set of hyperparameters.

3.2 Results & analysis

We are now in a position to analyse the results of the experiments performed under the two different data regimes. The mean and standard deviation test set scores for the sparse and full data regimes are shown in Tables 3.1 and 3.2 respectively, where the models with the lowest mean RMSE scores are shown in bold. As a general comment, we can see that in both data regimes, the mean RMSE values on test set 1 are lower than those of test set 2. This makes sense since predicting for patients which have not been observed during training time is a significantly harder task.

Focusing on the *sparse data regime* results of Table 3.1, in experiments 1 and 2, it is pleasing to see that the models we should be expecting to perform best are indeed doing so on both test sets 1 and 2. However, for experiments 3 and 4, not only the mean RMSE values are a lot higher compared to experiments 1 and 2, but they also do not seem to produce the expected results. For example, in experiment 4, we can see that the GBT with a common GP performs best on test set 2, even though one might have expected the GBT with an independent GP to be superior, given that it better matches the true underlying data-generating process. A key observation that we need to take into account, though, is that for experiments 3 and 4, the results produced show a lot of variability which makes us question the extent of their validity. Given the specifics of the data-generating process under the independent GP random effects, as outlined in Section 3.1.1, it is natural to expect the data to have high variance, since our choice of the kernel hyperparameter sampling distributions put significant probability density to a large range of values. This means that each independent GP can be very different from the rest, which naturally introduces variation in the data. This feature of the data-generating process, emphasises the challenge of accurately forecasting individual patient eGFR time-series, which can be very different from one another.

Overall, the difference in performance between linear mixed-effects models and mixed-effects models with GBTs for their fixed part is less significant than what we anticipated. Nevertheless, we clearly conclude that an appropriate GP-based model (either with a linear or a non-linear fixed-effects part) is much better at capturing the random-effects compared to random intercept models. Also, judging by the RMSE scores on both test sets, it seems that the “GBT with ID as a categorical variable”

model stands somewhere in the middle of random intercept and GP-based random-effects models. This is actually surprising, since this model is rather simple and does not properly account for the random-effects present in the data. We believe that the reason for this is that there are only 50 different patient IDs, which make it a categorical variable of moderate cardinality. Had we used a larger number of patients in our data-generating process, we would expect this model to perform less well, as the cardinality of the categorical variable would be a lot higher.

Moving on to the *full data regime*, we observe that for the experiments involving independent GPs in the data-generating process, the increase in the number of per patient data points dramatically reduces the variance of the results, albeit still being relatively high. For experiments 1 and 2, performance on test set 1 slightly improves compared to the sparse data setting, but test set 2 performance stays roughly the same. The conclusions drawn from the sparse data regime in experiments 3 and 4 hold in the full data regime as well. An important conclusion we can draw from the experiments in these two regimes is that the number of per patient data points makes a big difference in the predictive performance of all the models considered.

Moreover, we can deduce that assuming the true underlying fixed-effects function is linear, then the GP Boost model does not produce results that are any superior to the simpler linear mixed-effects models. However, even in a linear fixed-effects function setting, the GP Boost model still performs very well. On the other hand, under settings where the true fixed-effects function is highly non-linear, such as in Exp2 of the full data regime, the GP Boost model performs significantly better than a linear mixed-effects model. Hence, we state that the GP Boost model is a safer choice in settings where we do not have much information about the true fixed-effects function. Of course, a drawback of this model compared to linear mixed-effects models is its computational cost of training, which is dramatically higher.

3.3 A Bayesian approach towards hyperparameter selection

3.3.1 Motivation and setup

Now that we have analysed the results of the experiments, we can clearly conclude that models which have a GP for their random-effects part have a higher predictive accuracy compared to random intercept models. Therefore, we have decided to dive deeper into the GP components of these models and develop a better understanding

Model	Test set 1	Test set 2
Exp1: Linear function F with common GP		
Intercept only model	2.158 ± 0.150	2.059 ± 0.136
Linear model with random intercept	1.913 ± 0.142	1.787 ± 0.149
Linear model with common Gaussian Process	1.309 ± 0.074	1.524 ± 0.185
Linear model with independent Gaussian Process	1.736 ± 0.122	1.816 ± 0.157
GBT with ID as categorical variable	1.662 ± 0.147	1.659 ± 0.174
GBT with random intercept	1.950 ± 0.145	1.818 ± 0.146
GBT with common Gaussian Process	1.371 ± 0.068	1.566 ± 0.187
GBT with independent Gaussian Process	1.771 ± 0.120	1.838 ± 0.159
Exp2: Non-linear function F with common GP		
Intercept only model	2.161 ± 0.144	2.020 ± 0.142
Linear model with random intercept	2.026 ± 0.172	1.903 ± 0.130
Linear model with common Gaussian Process	1.467 ± 0.102	1.654 ± 0.155
Linear model with independent Gaussian Process	1.866 ± 0.148	1.928 ± 0.128
GBT with ID as categorical variable	1.653 ± 0.128	1.665 ± 0.153
GBT with random intercept	2.001 ± 0.163	1.851 ± 0.118
GBT with common Gaussian Process	1.424 ± 0.083	1.592 ± 0.168
GBT with independent Gaussian Process	1.822 ± 0.138	1.863 ± 0.142
Exp3: Linear function F with independent GP		
Intercept only model	10.641 ± 10.018	8.363 ± 5.786
Linear model with random intercept	10.687 ± 10.128	8.358 ± 5.745
Linear model with common Gaussian Process	10.561 ± 10.082	7.715 ± 5.470
Linear model with independent Gaussian Process	10.226 ± 10.022	7.671 ± 5.465
GBT with ID as categorical variable	10.287 ± 10.092	7.924 ± 5.450
GBT with random intercept	10.626 ± 10.015	8.375 ± 5.748
GBT with common Gaussian Process	10.508 ± 9.945	7.712 ± 5.542
GBT with independent Gaussian Process	10.211 ± 9.936	7.706 ± 5.495
Exp4: Non-linear function F with independent GP		
Intercept only model	10.647 ± 10.035	8.351 ± 5.759
Linear model with random intercept	10.693 ± 10.126	8.421 ± 5.756
Linear model with common Gaussian Process	10.572 ± 10.080	7.781 ± 5.478
Linear model with independent Gaussian Process	10.240 ± 10.018	7.729 ± 5.470
GBT with ID as categorical variable	10.293 ± 10.028	7.741 ± 5.486
GBT with random intercept	10.641 ± 10.028	8.368 ± 5.734
GBT with common Gaussian Process	10.514 ± 9.970	7.711 ± 5.548
GBT with independent Gaussian Process	10.220 ± 9.963	7.735 ± 5.485

Table 3.1: Mean and standard deviation of RMSE scores of experiments on synthetic data in the *sparse data regime*.

Model	Test set 1	Test set 2
Exp1: Linear function F with common GP		
Intercept only model	2.018 ± 0.112	2.004 ± 0.123
Linear model with random intercept	1.656 ± 0.109	1.724 ± 0.145
Linear model with common Gaussian Process	1.186 ± 0.0650	1.541 ± 0.154
Linear model with independent Gaussian Process	1.822 ± 0.080	1.860 ± 0.160
GBT with ID as categorical variable	1.333 ± 0.125	1.582 ± 0.157
GBT with random intercept	1.677 ± 0.109	1.738 ± 0.141
GBT with common Gaussian Process	1.200 ± 0.071	1.554 ± 0.152
GBT with independent Gaussian Process	1.810 ± 0.086	1.857 ± 0.149
Exp2: Non-linear function F with common GP		
Intercept only model	2.028 ± 0.131	1.999 ± 0.121
Linear model with random intercept	1.826 ± 0.121	1.843 ± 0.134
Linear model with common Gaussian Process	1.354 ± 0.071	1.661 ± 0.130
Linear model with independent Gaussian Process	1.928 ± 0.106	1.954 ± 0.138
GBT with ID as categorical variable	1.387 ± 0.141	1.621 ± 0.150
GBT with random intercept	1.692 ± 0.104	1.754 ± 0.143
GBT with common Gaussian Process	1.218 ± 0.065	1.569 ± 0.149
GBT with independent Gaussian Process	1.819 ± 0.084	1.867 ± 0.144
Exp3: Linear function F with independent GP		
Intercept only model	2.810 ± 0.884	3.468 ± 2.521
Linear model with random intercept	2.615 ± 0.968	3.301 ± 2.564
Linear model with common Gaussian Process	2.205 ± 1.057	3.127 ± 2.603
Linear model with independent Gaussian Process	2.631 ± 1.111	3.379 ± 2.670
GBT with ID as categorical variable	2.318 ± 1.020	3.180 ± 2.590
GBT with random intercept	2.630 ± 0.956	3.312 ± 2.562
GBT with common Gaussian Process	2.248 ± 1.088	3.147 ± 2.596
GBT with independent Gaussian Process	2.650 ± 1.109	3.391 ± 2.664
Exp4: Non-linear function F with independent GP		
Intercept only model	2.830 ± 0.907	3.470 ± 2.519
Linear model with random intercept	2.722 ± 0.933	3.376 ± 2.540
Linear model with common Gaussian Process	2.306 ± 1.016	3.208 ± 2.578
Linear model with independent Gaussian Process	2.711 ± 1.071	3.449 ± 2.650
GBT with ID as categorical variable	2.433 ± 0.994	3.227 ± 2.576
GBT with random intercept	2.726 ± 0.971	3.369 ± 2.546
GBT with common Gaussian Process	2.310 ± 1.073	3.198 ± 2.580
GBT with independent Gaussian Process	2.702 ± 1.099	3.429 ± 2.656

Table 3.2: Mean and standard deviation of RMSE scores of experiments on synthetic data in the *full data regime*.

of how they work. A major drawback of the GP Boost algorithm is that it is relatively restrictive when it comes to GP kernel hyperparameter selection, since it only allows finding the optimal set of hyperparameters by maximising the marginal likelihood of the model. As we have already discussed in Section 2.2.3, the marginal likelihood objective function is often non-convex, and this can result in suboptimal hyperparameter values. A potential remedy is repeating the optimisation procedure from several initialisation points, however, given the high computational cost of the GP Boost fitting process this is not a practical solution. A better alternative to marginal likelihood optimisation is to take a Bayesian approach, place prior distributions on the GP kernel hyperparameters, and infer their posterior distributions using Markov chain Monte Carlo sampling techniques. This approach has the additional advantage that it allows for injecting domain-specific knowledge into the learning process through the use of prior distributions. In the remaining of this section, we investigate whether this approach can improve the learning of these GP hyperparameters.

For our analysis, we need to select a specific data regime and one of the four data-generating settings as outlined in Section 3.1.1. To this end, we investigate GP-based models under the sparse data setting, as we believe it more closely resembles the real CKD dataset, and therefore the insights we draw will be more relevant to future experiments. Regarding our data-generating process, we decided to perform our analysis under the Exp2 setting, that is, with a non-linear fixed-effects function and a common GP for the random effects. We choose a common GP for our random-effects generation in an attempt to avoid having unreasonably high variation in the data, as noticed in experiments under the Exp3 and 4 settings, to be able to draw robust conclusions. Note also that we set the kernel lengthscale and marginal variance hyperparameters in the data-generating process to $l = 5000$ and $\sigma^2 = 1$ respectively, while keeping everything else the same as in the experiments before. The choice of this lengthscale value was to make sure the random effects are highly correlated, which would then render a GP model an ideal choice for testing the extent to which these correlations can be captured. In Figure 3.3, we can see the generated random effects from the true underlying process, alongside draws from the respective GP prior. We note that the true underlying functions that are possible under this GP prior are smooth.

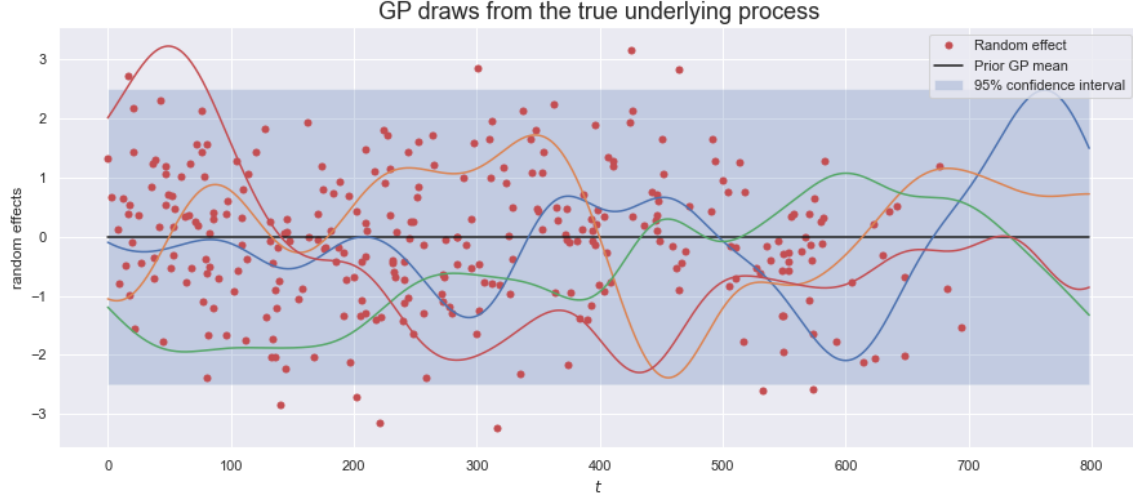


Figure 3.3: Simulated random effects and draws from the corresponding GP prior of the true data-generating process.

Let us make some important remarks regarding the subsequent analysis, that are key to justifying our approach:

- We focus on the *GBT with independent GP* model, since our ultimate goal in tackling CKD patient progression forecasting is through accurate personalised predictions for individual patients. It is therefore natural that we are interested in analysing each patient's GP obtained from this model;
- Additionally, we will be only considering model predictions on test set 1 data, as it is obvious that we cannot have a GP posterior for a patient who has no data in the training set;

3.3.2 Approach & findings

We now explain our approach towards estimating the GP kernel hyperparameters in a Bayesian way. Let us be clear that in this approach, we will be using the GP Boost algorithm to estimate *all* model parameter values *apart from the GP kernel hyperparameters l and σ^2* , whose estimation procedure is specified below. It is important to realise that we must use estimates for the random effects of the data, that can be obtained by subtracting the model's fixed-effects prediction part from the target value, i.e. the estimated random effect of the i^{th} patient at time t is $\hat{r}_{it} = y_{it} - \hat{F}(X_{it})$. Then, we can use a GP to model these random effects, and perform posterior inference over the model's kernel hyperparameters, as outlined in

(2.11). To be exact, we use the corresponding posterior means as point estimates, and this approach essentially corresponds to setting $\theta_{\text{post.mean}} = \hat{\theta} = \theta_M$ in Algorithm 1 (though the model likelihood noise variance in θ is still estimated using the standard GP Boost fitting procedure). For the Bayesian inference part of this task, we use the Stan probabilistic programming language, and more specifically its Python interface, PyStan.

Posterior inference can be done in a principled manner according to equation (2.11), where the posterior samples can be drawn using Hamiltonian Monte Carlo (HMC). Regarding our choice of priors, even though in theory they should be chosen in a way that reflects our beliefs about the true hyperparameter values, we also follow some useful guidelines from [Gelman et al., 1995]. To be precise, for the kernel lengthscale we use a $\mathcal{N}(5000, 1000)$ prior, where the large standard deviation reflects our uncertainty about what the true value is (assuming we are unaware of the true values of the hyperparameters used in the data-generating process). Moreover, for the kernel marginal variance, we use a Half-Normal² prior with a scale parameter of 1. This is to ensure we follow the positivity constraint of the marginal variance hyperparameter. For our sampling process, we run an HMC sampler across two Markov chains in parallel, where for each chain we have a total of 2000 posterior draws, in which the first 300 draws are discarded to make sure we only keep samples that have converged to the true posterior distribution. To monitor convergence of the Markov chains to the stationary distribution, we use the \hat{R} statistic [Moins et al., 2022], which is based on comparing between-and within-chain estimates of the kernel hyperparameters. In our experiments, all \hat{R} values are very close to 1, which indicates that the chains have mixed well, and we can use the obtained posterior sample draws for inference purposes.

To evaluate our method, we compare the estimated kernel hyperparameters *found using the GP Boost algorithm*, and the corresponding *posterior mean estimates* using our proposed approach. By observing Figure A.2, we can see that the thick-haired Markov chains are efficiently exploring all regions of the hyperparameter space, which makes us confident about their convergence to the true posterior distribution. The resulting hyperparameter estimates are summarised in Table 3.3, from which we clearly see that the posterior mean point estimates are much closer to the true values, which are $l = 5000$ and $\sigma^2 = 1$. Even though the marginal variance estimates using

²A Half-Normal distribution with a location parameter of zero is a Normal distribution which is truncated in a way such that only non-negative values have a non-zero probability density.

the two different techniques are relatively similar, we observe that the difference in the estimates of the lengthscale hyperparameter is huge.

To further highlight the differences between these two approaches, we compare individual patient predictions on test set 1 using the models’ corresponding GP posterior distributions. More precisely, we hand-pick a few different patients, with varying numbers of training and test set 1 data points, and analyse their resulting personalised predictions. We note that *only the random-effects* are shown, since both approaches use the same fixed-effects function. This analysis is only possible because we have access to the true random effects, as we are using simulated data.

Estimation method	Lengthscale l	Marginal variance σ^2
GP Boost algorithm	162	1.56
Posterior mean using HMC	5234	1.22

Table 3.3: GP kernel hyperparameter estimates using the GP Boost algorithm and the proposed Bayesian approach, calculated using patient 3 training data.

By analysing Figures 3.4, 3.5 and 3.6, we state that the resulting GP posteriors in which we use the posterior mean estimates as kernel hyperparameters are a lot closer to the true underlying GP of the data-generating process (shown in Figure 3.3), in terms of functional structure. That is, the GP posterior means are smoother, compared to the GP posteriors obtained using the GP Boost algorithm kernel hyperparameter estimates, which almost linearly interpolate between the training data. Additionally, we believe that the GP posterior using the posterior mean estimates, has more well-calibrated uncertainty estimates, something vital in medical applications. For example, by observing Figures 3.4 and 3.6, we notice that the GP posteriors using the GP Boost algorithm point estimates (subplots (b)) have unreasonably wide confidence intervals in regions where we have sufficient amounts of training data. Nevertheless, the corresponding subplots (a), have confidence intervals of more reasonable width given the data of the patients. Similarly, by observing subplot (b) in Figure 3.5, we see that half of the test points are outside the 95% prediction confidence interval. This implies that uncertainty in regions far from the training data should have been higher, as is the case in subplot (a) of the same figure, where only two of the test points are not contained in the prediction interval.

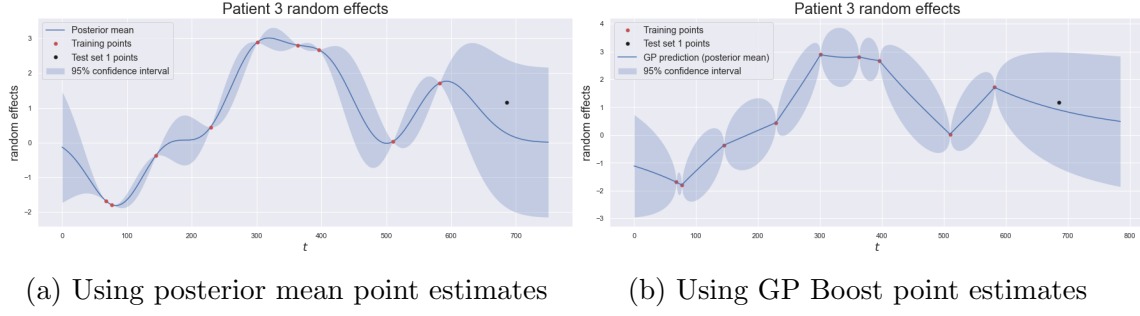


Figure 3.4: GP posteriors and the corresponding **true random effects** of patient 3 simulated data.

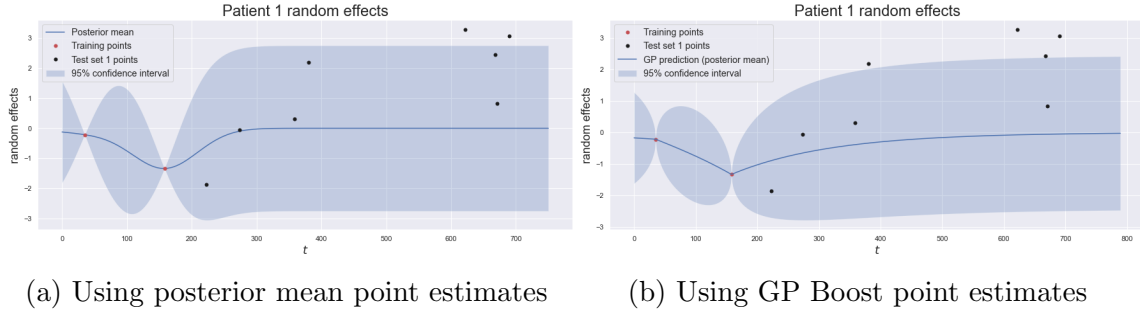


Figure 3.5: GP posteriors and the corresponding **true random effects** of patient 1 simulated data.

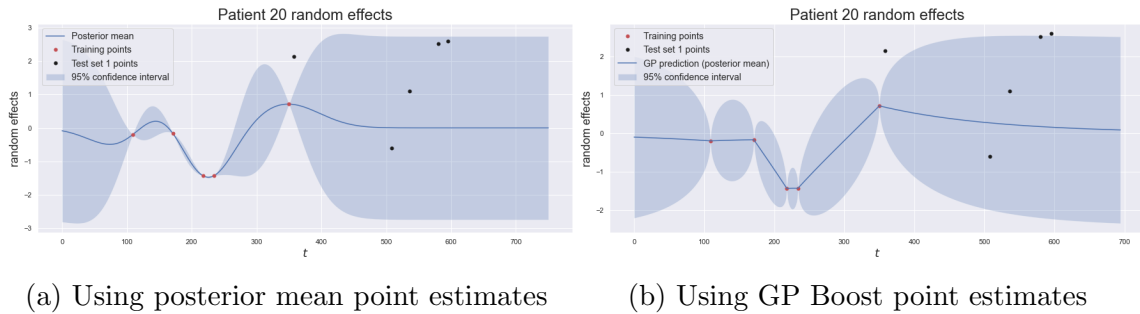


Figure 3.6: GP posteriors and the corresponding **true random effects** of patient 20 simulated data.

Based on the above analysis, we believe that the learning process of the kernel hyperparameters in the GP Boost algorithm can be made more robust by incorporating prior information. The use of prior distributions in Bayesian models can be particularly useful in settings where we do not have much available data for specific patients. Thus, using the insights we have gained from this section, we pose two interesting hypotheses for improving the GP Boost learning process regarding the estimation of its GP kernel hyperparameters:

1. Does using the posterior mean estimates *directly* in the GP Boost algorithm, lead to more accurate predictions by finding GP kernel hyperparameter values closer to the true ones? This hypothesis is equivalent to setting $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_M = \boldsymbol{\theta}_{\text{post.mean}}$ in Algorithm 1.
2. Does using the posterior mean estimates *as initialisation points* in the GP Boost algorithm lead to improved model performance? This basically corresponds to setting $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_{\text{post.mean}}$ in Algorithm 1.

Regarding our second hypothesis, our rationale is that since one of the key drawbacks in fitting GPs via the method of maximum marginal likelihood is being prone to getting stuck in local maxima, we hypothesise that a systematic approach towards selecting good initialisation points in the GP Boost optimisation procedure could make a significant difference to the estimated hyperparameter values. In the next chapter, alongside with the rest of our analysis on mixed-effects models, we will be also *experimentally investigating* these two hypotheses.

3.4 Conclusion

Through our experiments, we successfully compared GP Boost type models against linear mixed-effects models under a range of different data-generating settings, and gained valuable insights regarding their relative strengths and weaknesses. Notably, the GP Boost model can be viewed as a method that performs quite well under numerous settings, and in some cases offers a significant improvement in terms of predictive accuracy over traditional mixed-effects models. We have also delved deeper in the Gaussian Process component of this model, and proposed novel hypotheses on how to improve the learning process.

Chapter 4

Forecasting Patient eGFR Trajectories

Our goal in this chapter is to tackle the problem of predicting CKD patient progression *using our first approach*. Specifically, we use mixed-effects models to forecast patient eGFR trajectories, by predicting their future eGFR values using past data, similar to our methodology in Chapter 3. All the experiments in this chapter are performed on a real dataset, which consists of CKD patients from two medical sites.

4.1 Dataset description & pre-processing

The dataset we use for our experiments consists of a total of 10819 data points across 1554 distinct patients suffering from CKD. This dataset has patient information from two distinct medical sites, one is located in Patras, Greece, and the second one is in Sheffield, England. Amongst the 1554 patients, 560 of them come from the Patras site, and the remaining 994 come from the Sheffield medical site. A wide range of features are included in the dataset, such as date of birth, sex, ethnicity etc. Additionally, there are features which are more relevant to the CKD progression task, such as patient eGFR values, whether the patient underwent a kidney transplant operation, and many more.

Before proceeding to the training and prediction process, we had to do some necessary data cleaning and pre-processing steps in order to deal with missing values, and make sure the data was in good shape for the modelling part. Below, we present a summary of the most important data pre-processing steps we took:

- Firstly, as the dataset consisted of 77 distinct features for each patient, we followed expert advice from modellers at UCB Pharma to only select features that were relevant to our task. In this way, we can avoid overfitting noise present in unimportant features;
- Secondly, since each patient’s time-series data are at different dates, we had to find a way to incorporate this temporal component in a way that is uniform across all patients. Hence, to do this, we created a new feature “Times”, for which we mapped each date to the number of days since the date of the first time point. Using this approach, all patient time-series have a time value of 0 at their first data point;
- Moreover, for certain categorical features that had few observations in some of their levels, we decided to merge them to avoid having categorical features of unnecessarily high cardinality. For example, certain ethnic groups in the “Ethnicity” feature had very few members, and so we merged them to create four main ethnic groups (Caucasian, Black, Asian and Others);
- Data points with missing eGFR values were dropped since eGFR is our target variable, and is thus important to have accurate measurements. Since, the number of data points with missing eGFR values was not large, we preferred to drop them rather than impute them with possibly inaccurate sample statistics;
- For features that are affected by factors such as gender or ethnicity, for example a patient’s height, we imputed missing values according to their group-specific mean value. For example, if the height value of a Caucasian male was missing, we would impute it with the mean height value of all Caucasian males in the training set. Similarly, for features that are uncorrelated to factors like gender or ethnicity, such as systolic blood pressure, we imputed missing values using the mean value across all patient training set data;
- We emphasise that all missing value imputations were performed using statistics calculated only using the training set, to avoid leaking information from the test set in the modelling pipeline;
- Finally, all categorical features were one-hot encoded to create dummy variables for each distinct category of such features.

After our data-cleaning process, the final dataset (snapshot in Figure A.3) consisted of 10397 data points, across 1367 distinct patients, of which 492 and 875 were

from the Patras and Sheffield medical sites respectively. The final features included in the dataset are shown in Table 4.1.

Feature name	Description	Data type
ID	The patient ID.	Integer
times	Number of days since the first time point of the patient.	Integer
bun	Blood Urea Nitrogen level measurement. (in mmol/L)	Float
bp.sys	Systolic blood pressure measurement. (in mm Hg)	Float
disease	The disease the patient is suffering from (if any).	Categorical
patient_died	Whether this patient has died.	Boolean
kidney_transplant	Whether this patient underwent a kidney transplant operation.	Boolean
smoker	Patient's smoking status.	Categorical
weight	Patient's weight (in kg).	Float
height	Patient's height (in m).	Float
ethnicity	Patient's ethnicity.	Categorical
gender	Patient's gender.	Categorical
age	Patient's age (in years).	Integer
egfr	Patient's eGFR measurement (in mL/min).	Float

Table 4.1: CKD dataset features, their descriptions and data types.

Let us be clear in that we do not include the medical site location of each patient as a feature in our dataset because we decided to perform our analysis separately on the patients of each of the two medical sites. That is, we split our dataset into two, one dataset containing patients from the Patras medical site, and the other containing the patients from Sheffield's medical site. This approach is motivated by the fact that the eGFR values amongst patients between the two medical sites show substantial differences, as visualised using the boxplots in Figure 4.1. In particular, the median eGFR value across patients from Patras is roughly twice as high as the median eGFR value in Sheffield. The substantially lower eGFR values of patients in the Sheffield medical site imply that these patients are in more advanced CKD stages compared to those at Patras. Hence, since our ultimate aim is to provide precise personalised predictions with models that can incorporate patients' idiosyncrasies, it makes sense to build separate predictive models for each medical site.

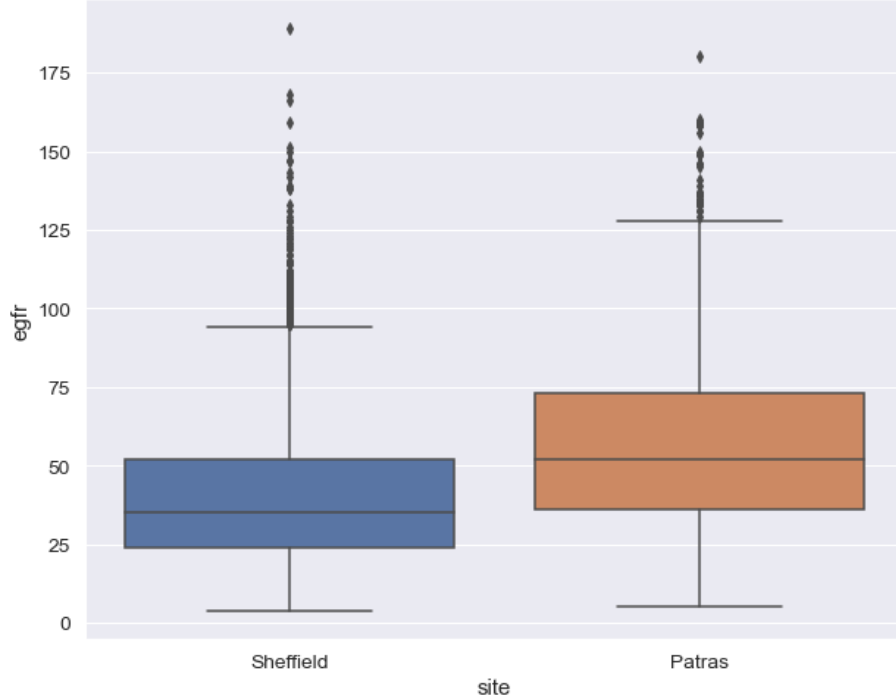


Figure 4.1: Boxplots for the eGFR values of patients in the two medical sites.

4.2 Experimental setup

We are now in a position to provide more details regarding our experimental setup. Let us start by describing our modelling pipeline for the experiments performed *on each of the two datasets*. We begin by splitting our data into 60% training set, and 20% for each of test sets 1 and 2, as previously done in Section 3.1.2. Model performance is quantified using the RMSE scores of the models on each of the two test sets. Furthermore, we repeat *the whole* train & test set splitting, data pre-processing, hyperparameter tuning, and model evaluation procedure for a total of five times using different train & test splits, to calculate the standard deviation of our test set RMSE scores and draw robust conclusions.

4.2.1 Models specification

Similar to our methodology in Section 3.1.1, we mention that for the fixed-effects covariates, we use all the features listed in Table 4.1 apart from the times of the data points, which we use as *our only random-effects covariate*. Regarding the mixed-effects models used, we use the same eight models introduced in Chapter 3, except for our “GBT with common Gaussian Process” model. The Patras and Sheffield

datasets contain 4267 and 6121 data points respectively. By also reminding ourselves that exact GP inference is of cubic time complexity in the number of training data points, we can easily see that repeating the model fitting and hyperparameter tuning process for a total of five times is prohibitively computationally expensive, and clearly infeasible. Therefore, for inference and prediction using that model, we resort to the use of the Vecchia approximation, as discussed in Section 2.3.4. From now on, we refer to this model as “Vecchia GBT with common Gaussian Process”. Following up from the hypotheses proposed towards the end of Section 3.3, we introduce two additional models:

1. **GBT with independent Gaussian Process using $\hat{\theta} = \theta_{\text{post_mean}}$** : This model corresponds to a GP Boost model with an independent GP for the random effects of each patient, where the kernel hyperparameters used for prediction are the posterior mean estimates obtained using our approach in 3.3.2;
2. **GBT with independent Gaussian Process using $\theta_0 = \theta_{\text{post_mean}}$** : Similarly, this model corresponds to our second hypothesis, as outlined in 3.3.2, and uses the posterior mean estimates as initialisation points in the GP Boost optimisation process.

This gives *a total of ten different models* for our experiments.

Let us now describe our modelling choices regarding the last two models we introduced. For the GP kernel lengthscale hyperparameter, l , we chose to place a $\mathcal{N}(5000, 1000)$ prior on it, since this specific choice gave good results in the synthetic data setting, and the scale of the “times” feature in the real and synthetic datasets is similar. Considering the GP kernel marginal variance, σ^2 , we used a $\mathcal{N}(150, 100)$ prior, where the mean value of 150 was selected by analysing the amount of variation in the eGFR training set values. Notice that in both priors, we use a large standard deviation value to reflect our lack of certainty on what the true values of these hyperparameters should be. For the posterior inference part, we run a Hamiltonian Monte Carlo sampler across two chains in parallel, each consisting of 1500 sample draws, in which we use the first 300 draws as burn-in period. Just as in the synthetic data experiments, we monitor the convergence of the Markov chains to their stationary distribution using the \hat{R} statistic. In our experiments, all \hat{R} values are very close to 1, which indicate that the chains mix well, and we can use the obtained posterior samples to reliably calculate the corresponding quantities of interest.

4.2.2 Hyperparameter selection

We start by mentioning that for the models involving GPs, the covariance function used was the squared-exponential. Also, we remind ourselves that linear mixed-effects models do not have any hyperparameters, and so tuning is only performed for GP Boost type models. For the GP Boost algorithm hyperparameters, we consider tuning the number of boosting iterations M , and the learning rate ϵ , as specified in Algorithm 1, where we try values in $\{1, \dots, 5000\}$ and $\{0.10, 0.30, 0.50, 0.80, 1.0, 1.5, 2.0\}$ respectively. These choices were made by observing that having more iterations in the GP Boost algorithm produced better results on average. In addition, the learning rate plays an important role in the optimisation process, and so we decided to try a larger range of values compared to our synthetic data experiments. As for the gradient boosted tree hyperparameters, we only consider tuning two of the most important hyperparameters for computational reasons. These are the minimum number of data points in the tree leaf nodes and the maximum depth of the trees, where we try values in $\{5, 10, 50, 100\}$ and $\{3, 10, 50, 100\}$ respectively. Lastly, regarding the number of nearest neighbors m_ν used in the Vecchia approximation, we try values in $\{10, 30\}$. To justify this choice, we state that using nearest neighbor values significantly larger than 30 resulted in a big increase in training time, which defeats the whole purpose of using approximate techniques over exact ones. On a final note, we mention that the hyperparameter selection process is performed through exhaustive grid-search using five-fold cross-validation on the training set, with the average five-fold RMSE score over both test sets 1 and 2 as the evaluation metric for the optimal set of hyperparameters.

4.3 Results & analysis

In this section, we present our results on the two datasets considered, critically evaluate the models used, and draw some important conclusions.

4.3.1 Sheffield medical site

In Table 4.2 we can see our Sheffield experiment results for the ten models considered, with the scores of the top-performing models in terms of mean RMSE displayed in bold. As a general comment, we mention that the standard deviations on both test set 1 and 2 results are very reasonable, which is a sign of consistency in the model

fitting process. In other words, fitting the models on different train & test splits of the same dataset produces results that are not too different from each other. This in turn allows us to draw reliable conclusions by analysing their performance.

Starting with the test set 1 scores, it is apparent that GP Boost type models provide superior predictive performance over linear mixed-effects models. This potentially implies that the fixed-effects function of the true underlying data-generating process of the patients at Sheffield is non-linear. Additionally, we note that the two last models that use information from the posterior distributions of the GP kernel hyperparameters are the top-performing ones, even if the difference with the “GBT with independent Gaussian Process” model is not too big. This is a rather significant result, as it suggests that the incorporation of prior knowledge in a principled manner indeed improves the learning process in the GP Boost algorithm.

Moving on, the “GBT with ID as categorical variable” and “Vecchia GBT with common Gaussian Process” models have the worst test set 1 performance. As we hypothesised in the results analysis of Chapter 3, the significantly increased cardinality of the “Patient ID” feature compared to the synthetic data setting (875 vs 50 distinct patients respectively), results in poor estimates of individual patient coefficients and fails to adequately capture the random effects. Regarding the Vecchia approximation GP Boost model, it is clear that on this dataset, the quality of this approximate inference technique is not sufficiently good. The Vecchia approximation is known to produce very good results in spacial data settings [Zilber and Katzfuss, 2021]. However, it might be the case that approximating the likelihood function using a fixed number of nearest neighbors in a temporal setting does not yield accurate results. Unfortunately, the `GPBoost` package does not currently support other types of GP approximations, and we believe that this is an important limitation of the model. Finally, similar to the synthetic data setting, models which include a GP component for modelling the random effects in the data, produce better results compared to random intercept models. This is made particularly apparent in models that utilise an independent GP for each patient.

Observing the test set 2 RMSE scores, we state that models that use a gradient boosted tree as their fixed-effects function perform significantly better compared to their linear counterparts. In fact, this confirms our supposition that the true underlying fixed-effects function is non-linear. To see this, remember that test set 2 only consists of patients that have not been observed in the training set, and hence for models which involve estimating patient-specific coefficients, *predictions are based solely on the models’ estimated fixed-effects function*. This is highly suggestive that

a GBT provides a much better fit to the data compared to a linear model.

Model	Test set 1	Test set 2
Intercept only model	12.627 ± 0.328	21.381 ± 0.804
Linear model with random intercept	12.791 ± 0.647	20.284 ± 1.268
Linear model with common Gaussian Process	12.714 ± 0.629	20.184 ± 1.285
Linear model with independent Gaussian Process	11.540 ± 0.660	19.939 ± 1.112
GBT with ID as categorical variable	14.135 ± 0.526	16.189 ± 1.111
GBT with random intercept	10.578 ± 0.188	16.232 ± 1.075
Vecchia GBT with common Gaussian Process	14.064 ± 0.351	16.891 ± 0.868
GBT with independent Gaussian Process	9.914 ± 0.208	16.118 ± 0.931
GBT with independent Gaussian Process using $\hat{\theta} = \theta_{\text{post_mean}}$	9.886 ± 0.246	16.404 ± 0.905
GBT with independent Gaussian Process using $\theta_0 = \theta_{\text{post_mean}}$	9.849 ± 0.156	16.125 ± 0.937

Table 4.2: Mean and standard deviation of RMSE scores on the *Sheffield data*.

It is widely known that gradient boosted trees offer feature importance measures, which we use to gain a better understanding of how eGFR values depend on the features shown in Table 4.1. Using our top-performing model, we obtain these feature importance estimates, which are shown in Figure 4.2.

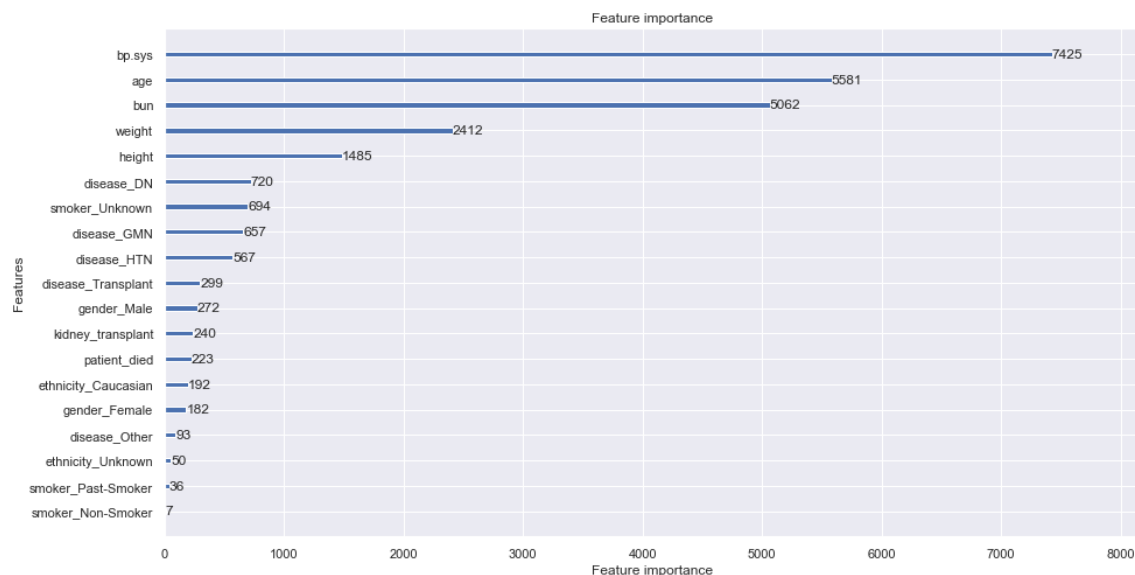


Figure 4.2: GP Boost feature importance estimates on the Sheffield data.

The above figure shows that the three most important features for predicting eGFR values are systolic blood pressure, age and blood urea nitrogen levels respectively. This feature importance plot makes sense, since all three of these features directly affect kidney functionality, which is highly correlated with eGFR values.

For example, in the case of blood urea nitrogen, one could say that the higher the concentration levels in patients’ blood, the less effective their kidneys are, indicated by the inefficient filtration of the urea nitrogen substance. Though, it is somewhat surprising that the systolic blood pressure feature is by far the most important one in predicting eGFR values. In Figure A.4 of the appendix, we show partial dependence plots (PDPs) to summarise the marginal effect each of the three features has on eGFR values. As we would expect, blood urea nitrogen levels and age are negatively correlated with eGFR values. In contrast, systolic blood pressure is positively correlated with eGFR values.

4.3.2 Patras medical site

We proceed with a similar analysis for our results on the Patras dataset. As a general comment, we observe that the RMSE scores of the top-performing models are slightly higher compared to the Sheffield dataset. We think that this is due to the fact that the Sheffield dataset is bigger (6121 data points vs 4267 in Patras), since one of our main findings in the synthetic data experiments was that model performance heavily depends on the dataset size. Notably, the two models that use the posterior mean estimates in the GP Boost algorithm are the top-performing ones in terms of test set 1 scores. A very interesting observation related to this is that the gap in predictive performance between these two models and the remaining ones has grown. Once again, this highlights the benefit of introducing prior information in the learning process of GP Boost, something particularly helpful in sparse data settings like this one.

Continuing with our observations about test set 1 model performance, we see that linear mixed-effects models have very similar performance to GP Boost type models, which points us towards the direction that the true underlying fixed-effects function must be close to linear. The bad performance of the “GBT with ID as categorical variable” and “Vecchia GBT with common Gaussian Process” models seems to be consistent with our reasoning in the Sheffield results. Nonetheless, these models are clear winners when it comes to predicting on test set 2, which proves that in smaller data settings, the two models have better generalisation properties compared to the rest. Our last comment regarding performance on test set 2 is that the results for certain linear models have unreasonably high mean RMSE scores, and exhibit high variability. This leads us to the conclusion that these models potentially overfit the training set, and could benefit from regularisation approaches.

Model	Test set 1	Test set 2
Intercept only model	13.773 ± 0.411	26.007 ± 1.353
Linear model with random intercept	10.946 ± 0.191	26.016 ± 10.421
Linear model with common Gaussian Process	10.857 ± 0.320	20.506 ± 0.399
Linear model with independent Gaussian Process	11.074 ± 0.485	35.557 ± 23.594
GBT with ID as categorical variable	14.311 ± 0.547	17.021 ± 0.948
GBT with random intercept	10.819 ± 0.315	19.688 ± 1.066
Vecchia GBT with common Gaussian Process	13.694 ± 0.162	16.920 ± 0.688
GBT with independent Gaussian Process	10.875 ± 0.265	19.601 ± 0.983
GBT with independent Gaussian Process using $\hat{\theta} = \theta_{\text{post_mean}}$	10.628 ± 0.261	19.726 ± 0.634
GBT with independent Gaussian Process using $\theta_0 = \theta_{\text{post_mean}}$	10.572 ± 0.247	19.752 ± 0.758

Table 4.3: Mean and standard deviation of RMSE scores on the *Patras data*.

By analysing Figure 4.3, we note that the three most important features are the same as in the Sheffield dataset, but with the order of the first and third most important features swapped. Even though we believe that the feature importance order that we observe here is what we should expect, we also think that perhaps there is some complex interaction between systolic blood pressure measurements and eGFR levels in patients at the Sheffield medical site, that the models are able to successfully capture. In figure A.5, we can see the underlying marginal effect relationships between the three aforementioned most important features and eGFR values, along with their mean functions and 95% confidence intervals.

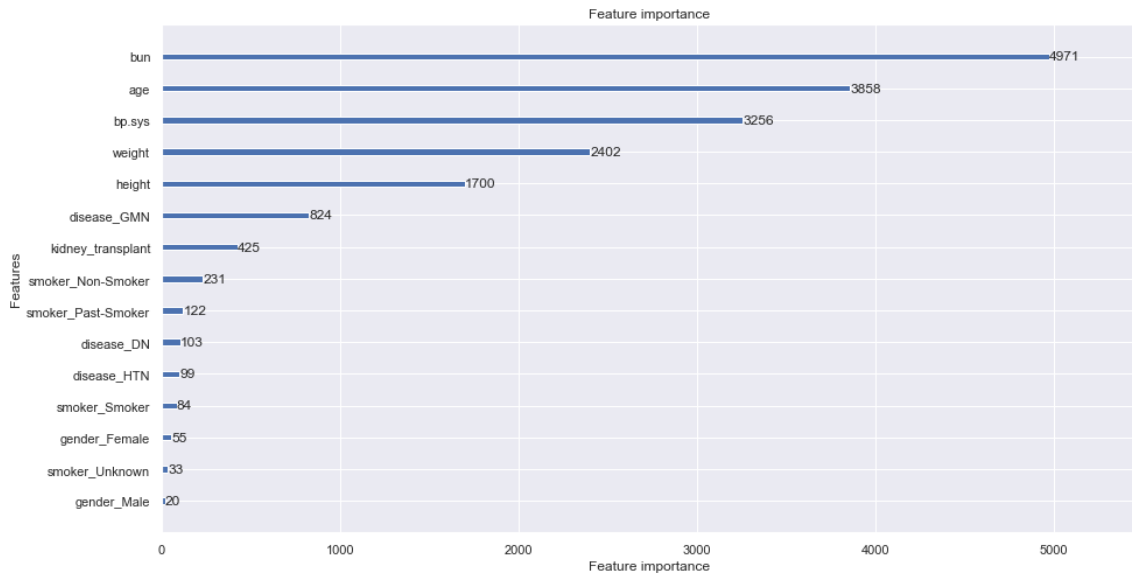


Figure 4.3: GP Boost feature importance estimates on the Patras data.

To further examine the extent to which the random effects are captured by independent GP models, we have decided to analyse the random effects predictions of *the test set 1 top-performing model* of Table 4.3. To make sure we get this straight, the y -values of the points in Figure 4.4 represent the *true eGFR values minus the fixed-effects prediction part* of the specified model. We can visually confirm that the individual GP components model the random effects to a very satisfactory level of accuracy. However, as a result of the data being noisy, we see that in certain regions our model is overly conservative in its predictions, indicated by the large width of the confidence intervals (e.g. for patient 406).

As a final remark, let us tie the work of this chapter back to the main goal of this thesis, forecasting CKD patient progression. Using precise predictions of a patient’s eGFR trajectory, healthcare professionals can draw conclusions regarding the future CKD stage of that patient, using a CKD stage and eGFR value conversion table, as seen in Table A.6. Thus, one could view the CKD patient progression forecasting task as equivalent to tackling the patient eGFR regression problem, that was investigated throughout this chapter.

4.4 Conclusion

We end this chapter with a concise summary of the most noteworthy findings of our experiments. We have shown that GP Boost type models provide significant improvements over linear mixed-effects models in terms of *both test set 1 and 2 accuracy*. Furthermore, we experimentally validated the hypotheses proposed in Chapter 3, and came up with *a novel method* to make the learning process in GP Boost more robust by incorporating prior information. Through this method, experienced healthcare professionals can directly inject domain-specific knowledge in the modelling process, and have a substantial impact in settings of limited data availability. Regarding precise personalised patient predictions based on past data, we conclude that mixed-effects models with an independent Gaussian Process for each patient are unparalleled when it comes to modelling random effects of temporal nature. Finally, their well-calibrated uncertainty estimates can prove to be a very handy tool for guiding medical professionals in their decision-making in critical situations.

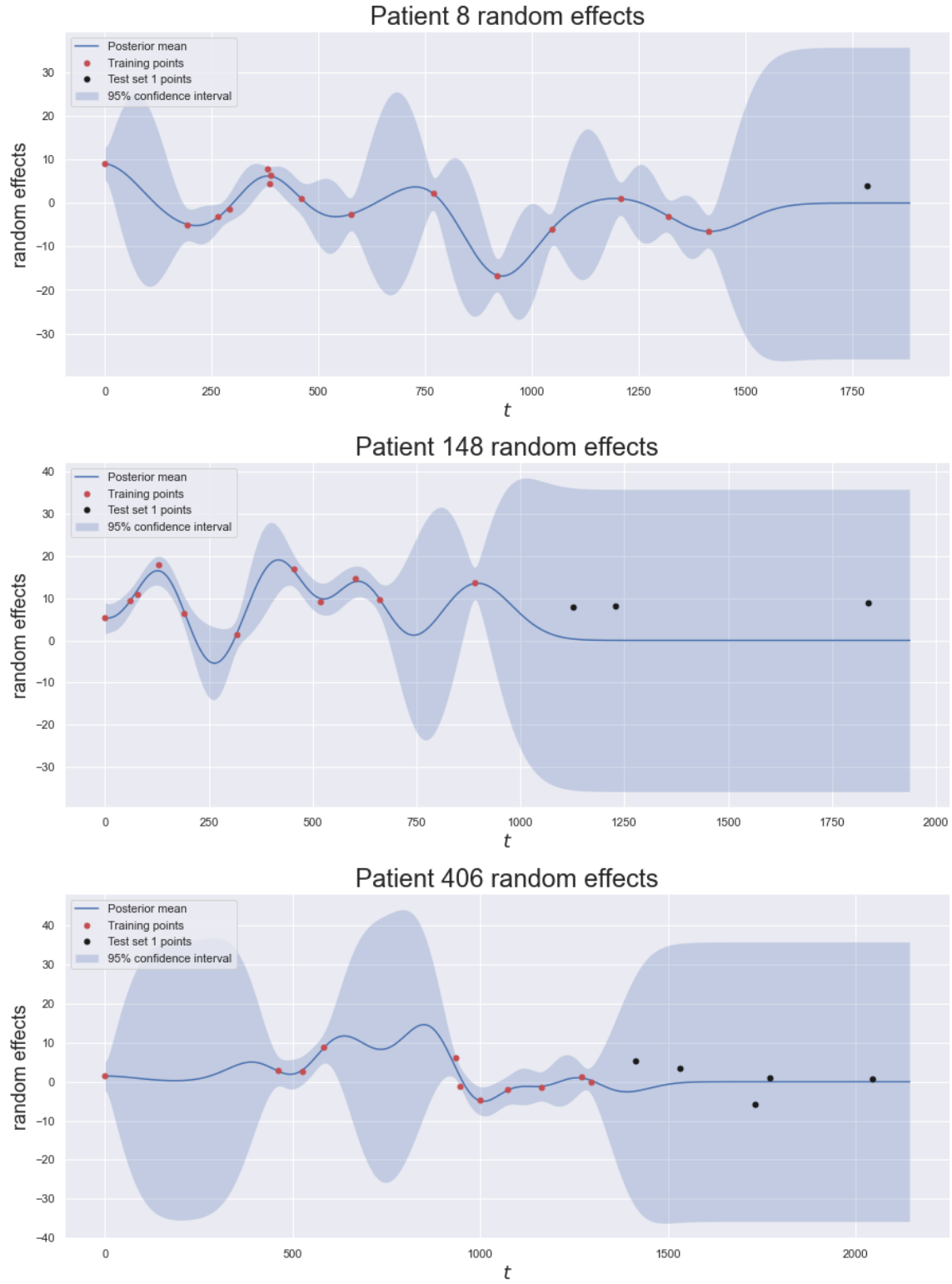


Figure 4.4: Individual patient GP posteriors using the top-performing model of Table 4.3 and the corresponding **predicted random effects** across three different patients.

Chapter 5

Forecasting CKD Patient Progression Status

In this chapter, we explore *the second proposed approach to forecasting CKD patient progression*. To motivate this problem, we note that in many cases, there is limited patient data availability to medical professionals, and making important decisions for an individual patient can be a very challenging task. Hence, predictive models which only utilise patient data at a *single* point in time could be invaluable for CKD progression modelling. For this task, we begin by investigating how time-series clustering techniques can be used to assign data-driven labels to individual patients, according to their eGFR trajectory trends. We then convert our longitudinal dataset to a cross-sectional one, by only keeping the first data point for each patient, and assigning categorical target values according to the patients' cluster assignments. Finally, using our cross-sectional dataset, we compare and evaluate the performance of a range of classification models, including state of the art Bayesian neural networks.

5.1 Time-series clustering for eGFR trajectories

In this section, we describe the approach taken to assign CKD progression status labels to individual patients. We begin by stating that after careful discussions and expert advice from specialists in the field of CKD at UCB Pharma, patients can be broadly categorised into three types according to their eGFR trajectory time-series. The first type of patients, are those who show an increase in their eGFR values over time, which signals that there is an *improvement in their kidney functionality and*

CKD progression. Secondly, patients who display a decrease in their eGFR values over time, can be interpreted as those whose kidney functionality is further reduced and *their CKD progression is worsening*. Lastly, patients with no apparent patterns or trends in their eGFR trajectories, can be interpreted as those with a *relatively stable CKD progression*. Hence, even though the optimal number of clusters (which is chosen a priori) is addressed properly through the process of model selection, we state that domain-specific expert advice can be invaluable in making such modelling choices.

Before we dive into the experiments performed and the results obtained on the dataset, we provide a short discussion regarding the choice of the specific clustering techniques chosen. As we can see in Figure 5.1, individual patient eGFR trajectories can be highly irregular, with different number of data points at unevenly spaced time intervals, and with complex trends in some cases. Therefore, because of these particularities of the eGFR trajectories, traditional clustering techniques which use common distance metrics such as the Euclidean distance are inappropriate for this task, and we have to resort to alternatives. As explained in Section 2.4.1, Dynamic Time Warping (DTW) is a similarity pseudo-metric that has the ability to properly align two time-series of different lengths. We emphasise that *all distance calculations* throughout this section are performed using the DTW pseudo-metric.

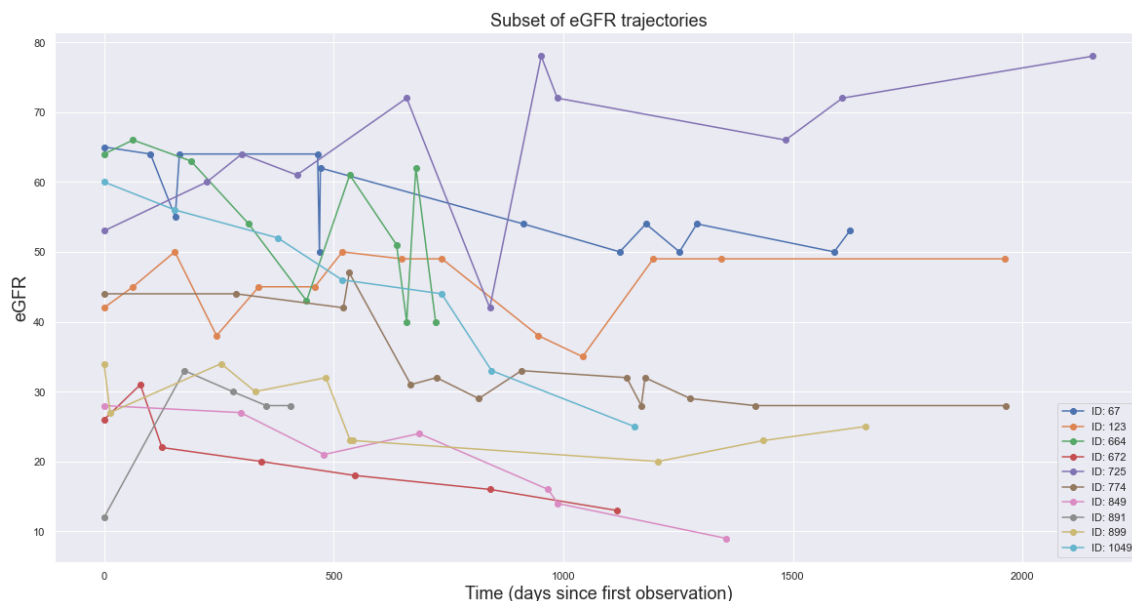


Figure 5.1: A subset of the patients' eGFR time-series trajectories.

5.1.1 Synthetic data experiments

Experimental setup

To assess the suitability of the three clustering techniques we consider, for the task of grouping eGFR trajectories with similar trends, we decided to perform some experiments on synthetic data, where the true underlying trends of the time-series were known in advance. This allowed us to assess the quality of the resulting clusters using *extrinsic measures*, which require ground truth labels. This is in contrast to our model evaluation performed on the real data, where we assessed the clustering quality using *intrinsic measures*, as no ground truth labels were available. The data-generating process was designed in a way such that the generated time-series could be as representative as possible of the real patient eGFR trajectories. The motivation for choosing this specific data-generating process is to replicate three key characteristics that we observe in eGFR trajectories:

- Each eGFR trajectory time-series can be of different length;
- Time-points across an individual eGFR trajectory time-series are often unevenly spaced;
- The strength of the underlying trend and the amount of noise present in different eGFR trajectory time-series can be very different from one another.

We generate the data according to three underlying types of trends, where the data-generating process *for each* univariate time-series is specified as follows:

1. Sample the number of data points of the individual time-series, $n \in \mathbb{N}$, from a discrete uniform distribution, such that $n \sim \text{Uniform}[3, 20]$.
2. Sample the time values of the time-series, $\mathbf{t} \in \mathbb{R}^n$, where $\mathbf{t}_i \sim \text{Uniform}[0, 1000]$, is sampled without replacement, for $2 \leq i \leq n$ and set $\mathbf{t}_1 = 0$. Then, sort the elements of the vector \mathbf{t} from smallest to largest.
3. Pick a trend and function type at random, with a choice from $\{\text{Increasing, Decreasing, Random}\}$ and $\{\text{Linear, Quadratic}\}$ respectively.
4. If the chosen function type is *increasing* and *linear*, generate the target values, $\mathbf{y} \in \mathbb{R}^n$, according to:

$$\mathbf{y} = c \cdot \mathbf{1} + \beta \cdot \mathbf{t} + \mathbf{e},$$

where $\beta \sim \text{Uniform}(0, 20)$, $c \sim \text{Uniform}(0, 5)$, $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$ is an isotropic Gaussian noise term, and $\sigma^2 \sim \text{Uniform}(0, 500)$.

5. If the chosen function type is *increasing* and *quadratic*, generate the target values, $\mathbf{y} \in \mathbb{R}^n$, according to:

$$\mathbf{y} = \beta \cdot \mathbf{t}^2 + \mathbf{e},$$

where $\beta \sim \text{Uniform}(0, 0.01)$ and \mathbf{e} is generated as above. Note that $\mathbf{t}^2 := \mathbf{t} \odot \mathbf{t}$, denotes the Hadamard ³ (or element-wise) product of the vector \mathbf{t} with itself.

6. When the trend is of type *decreasing*, we simply *flip the sign* of the sampled coefficient β , while keeping everything else unchanged.
7. Finally, for the *random* type trend, we simply set $\mathbf{y} = \mathbf{e}$, where the noise term \mathbf{e} is again sampled as above.

Samples of the generated time-series can be seen in Figure 5.2. For our experiments, we generate five different sets, each consisting of 500 time-series, where each time-series is sampled according to the above procedure.

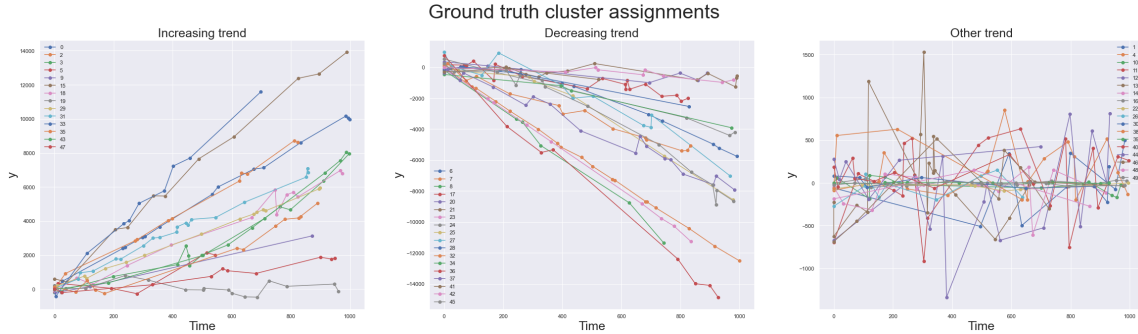


Figure 5.2: Samples from the data-generating process described in Section 5.1.1.

Data pre-processing & transformations

We now describe the pre-processing steps we took in an attempt to improve the quality of the clustering. Firstly, all time-series were normalised so that they had a mean value of zero across time. This step was crucial in ensuring that the resulting

³For the definition of the Hadamard product, visit [https://en.wikipedia.org/wiki/Hadamard_product_\(matrices\)](https://en.wikipedia.org/wiki/Hadamard_product_(matrices))

clusters formed were based on the different *trends* present in the time-series and not based on any other features (such as the magnitude of their y -axis values). Moreover, each time-series was smoothed using the exponential smoothing technique, a common time-series analysis tool used to average out noise. The rationale for using the exponential smoothing technique is that, the overall trend of the time-series becomes clearer after it has been smoothed. More formally, consider a time-series consisting of T time points, $\{x_t\}_{t=1}^T$, and denote the corresponding exponentially smoothed series by $\{s_t\}_{t=1}^T$. Then, the observations of the smoothed series $\{s_t\}$ are defined as:

$$\begin{aligned} s_1 &= x_1 \\ s_t &= \alpha x_t + (1 - \alpha)s_{t-1}, \quad 2 \leq t \leq T \end{aligned}$$

where $\alpha \in (0, 1)$ is the smoothing factor, which is pre-specified by the user. In our subsequent experiments, we use a smoothing factor of 0.2.

Let us mention that for the clustering of the real patient eGFR trajectories, we repeat the above pre-processing steps, and additionally we only select the subset of eGFR trajectories with at least three data points. Otherwise, any meaningful clustering according to the eGFR trajectory trends would not be possible for time-series of shorter length. Of course, choosing this threshold for the number of data points is a trade-off between having enough data points in each time-series, and keeping as many time-series as possible in the dataset.

Models considered & evaluation metric

In the synthetic data experiments performed, we compare three types of time-series clustering models:

1. k -Means clustering, with $k = 3$:

We specify that for the barycenter estimation procedure as discussed in Section 2.4.2, we set the number of DBA iterations to 100. Also, we use a maximum number of 50 iterations for each run of the k -Means algorithm, with a tolerance value of 10^{-6} . That is, if the decrease in the Within-Cluster Sum of Squares is less than 10^{-6} between two consecutive iterations of the k -Means algorithm, then the algorithm is said to converge. Finally, we state that the algorithm

was run from three different centroid initialisation points, and the best results were kept.

2. k -Medoids clustering, with $k = 3$:

Similar to the k -Means algorithm, we use a maximum number of 200 iterations, and a tolerance value of 10^{-4} . In addition, the initial medoids are randomly chosen.

3. Hierarchical Agglomerative Clustering (HAC):

For this algorithm we choose to have three resulting clusters, and consider the complete, single, average and ward linkage functions.

The hyperparameters chosen in the above algorithms' specification settings were found by experimentation on the data, and by following good-practice standards suggested by experts. Since we know that there are three types of clusters in the true data-generating process, we decided to avoid treating the number of clusters k as a tuning parameter, in an attempt to simplify things and focus on comparing and interpreting the resulting clusters formed. However, when applying these methods to the actual dataset, we take care to compare the models using different cluster numbers k , and choose the optimal number in a principled manner through model selection. Also, for the HAC method, the results we present are for the linkage function that performed best.

To evaluate the performance of the three models, and given that we have access to the true label of each of the generated time-series, we decided to use the *Rand index* metric [Rand, 1971]. Intuitively, the Rand index quantifies the similarity between two different data clusterings by counting pairs of samples that are allocated to the same or different clusters in the predicted and actual clusterings. More formally, given a set of n data points $S = \{s_1, \dots, s_n\}$, the Rand index between two clusterings is defined as:

$$\text{Rand Index} = \frac{\text{number of agreeing pairs between the two clusterings}}{{}^nC_2},$$

where nC_2 represents the total number of pairs that can be formed from the elements of the set S .

Note that the Rand index takes values between 0 and 1, with 1 indicating that the resulting data clusterings match perfectly, and with 0 indicating that the data clusterings agree on none of the possible pairs of points that can be formed.

Experimental results

In Table 5.1 we can see the resulting mean and standard deviation Rand index scores produced by the three models, over the five different time-series sets. We conclude that k -Means clearly has the best performance, closely followed by k -Medoids and then by the Hierarchical Agglomerative clustering algorithm, which has the worst performance. Notably, the performance of the three models is quite satisfactory, which renders them an appropriate choice for clustering eGFR trajectories later on. Let us also state for completeness that we decided to visually inspect the time-series which were assigned to the wrong clusters by the k -Means algorithm. We found out that in most wrongly clustered time-series, the underlying trend was either severely distorted by large amounts of noise or the time-series consisted of few data points. Such time-series could have been very well mislabelled by humans as well, which further highlights the challenge in accurately clustering eGFR trajectories.

Model	Rand index score
k -Means (with $k = 3$)	0.78 ± 0.036
k -Medoids (with $k = 3$)	0.76 ± 0.033
HAC with average-linkage	0.68 ± 0.045

Table 5.1: Mean and standard deviation of Rand index scores for the experiments on the synthetic data.

5.1.2 Clustering patient eGFR trajectories

Evaluation metric

In contrast to our synthetic data experiments evaluation approach, we now need to resort to the use of intrinsic metrics to assess the clustering quality since ground truth labels are not available. In brief, intrinsic metrics are based on the concepts of cluster separation and compactness, which are key ideas in clustering techniques. Ideally, and assuming the data comes from a distribution with different subgroups present, a successful clustering algorithm will produce clusters which are non-overlapping and where the different objects within the same cluster are close to each other. To this end, we have decided to use the Silhouette score metric as proposed in [Rousseeuw, 1987], which is a particular choice of intrinsic metric. Choosing this metric was due to the fact that we measure the distance between two time-series

using DTW instead of Euclidean distance, and hence our choice of metrics is reduced to the Silhouette score and the Calinski-Harabasz index. However, using the Calinski-Harabasz index requires the time-series to be of equal length. Thus, the only appropriate metric in our case is the Silhouette score, obtained as follows:

Assume our data have been clustered into κ distinct clusters. Then, for the i^{th} data point in the i^{th} cluster, $i \in C_I$, let $a(i) := \frac{1}{|C_I| - 1} \sum_{j \in C_I \setminus \{i\}} d(i, j)$ denote the average intra-cluster distance of the i^{th} data point and the remaining data points in that cluster. Also, define $b(i) := \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i, j)$ to be the average nearest-cluster distance of the i^{th} data point, where this nearest cluster cannot be the cluster in which data point i belongs to. We then define the *Silhouette coefficient* of the i^{th} data point as $s(i) := \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$.

Finally, the Silhouette score is defined as *the average Silhouette coefficient of all data points*. Note that the range of possible values the Silhouette score can take is $[-1, 1]$, with 1 being the best value and -1 the worst.

Model comparison and selection

We are now in a position to use the three clustering algorithms we considered in Section 5.1.1 on the actual dataset and cluster individual patient eGFR trajectories according to their trends. As we have already discussed at the beginning of this chapter, by utilising domain-specific knowledge we can expect to see three different clusters, corresponding to the three main types of patient progression patterns. Nevertheless, we properly address this model selection problem by re-running the clustering algorithms on our dataset with $k \in \{2, 3, 4, 5\}$. Regarding the Hierarchical Agglomerative clustering algorithm, even though results were similar with the four different linkage functions introduced before, we decided to only present the results for the ward-linkage function which produced marginally better results in terms of Silhouette scores. By analysing the results in Table 5.2, we can reach two conclusions. Firstly, we notice that for almost all values of k , the k -Means and Hierarchical Agglomerative clustering algorithms perform better compared to the k -Medoids algorithm, with the k -Means algorithm having slightly superior performance. Secondly, for the two top-performing algorithms, the results with $k = 3$ clusters are clearly better compared to the results for other values of k . Certainly, it is pleasing to see that the optimal number of clusters suggested by the model selection

process is in agreement with what we should expect based on domain knowledge.

Model	$k = 2$	$k = 3$	$k = 4$	$k = 5$
k -Means	0.440	0.516	0.453	0.430
k -Medoids	0.441	0.388	0.407	0.381
HAC with ward-linkage	0.481	0.507	0.405	0.405

Table 5.2: Silhouette scores for different number of clusters, k , for the three algorithms considered.

We therefore decide that the optimal model choice for clustering the eGFR patient trajectories is the k -Means algorithm with $k = 3$.

Interpreting the k -Means clusters

To conclude this section, we will now critically evaluate the resulting clusterings produced by the k -Means model. Specifically, we refer to Figure A.1, which displays a histogram of the Silhouette coefficient values in our dataset. We can see that the majority of the data points have Silhouette coefficient values of at least 0.5, which indicates that the clustering quality is sufficiently good. However, we also require that the clusters formed are also of practical medical significance for the specific task of assigning data-driven disease progression status labels to individual patients. Therefore, in Figure 5.3 we inspect subsets of randomly chosen patient eGFR trajectories from each cluster, in order to confirm that the clustering outputs are meaningful. Indeed, these plots confirm the presence of the three different types of patient eGFR trajectories. Notably, in cluster 1, we clearly see the decreasing trend in the eGFR values of patients over time. Patients belonging to this cluster correspond to those whose CKD condition is becoming more severe over time. Similarly, by analysing cluster 2, we conclude that patients belonging to this cluster correspond to those who seem to show an improvement in their kidney functionality. Finally, patients in cluster 0 can be viewed as those with no clear pattern in their eGFR trajectories, and whose CKD progression status is neither improving nor worsening.

In conclusion, we believe that the time-series clustering techniques have produced meaningful cluster labels from a medical perspective. These cluster labels will be now used in Section 5.2 as target labels for the patients' CKD patient progression statuses. With this, we are in a position to treat the CKD patient progression forecasting problem as a three-class classification task, whose importance will be highlighted in the next section.

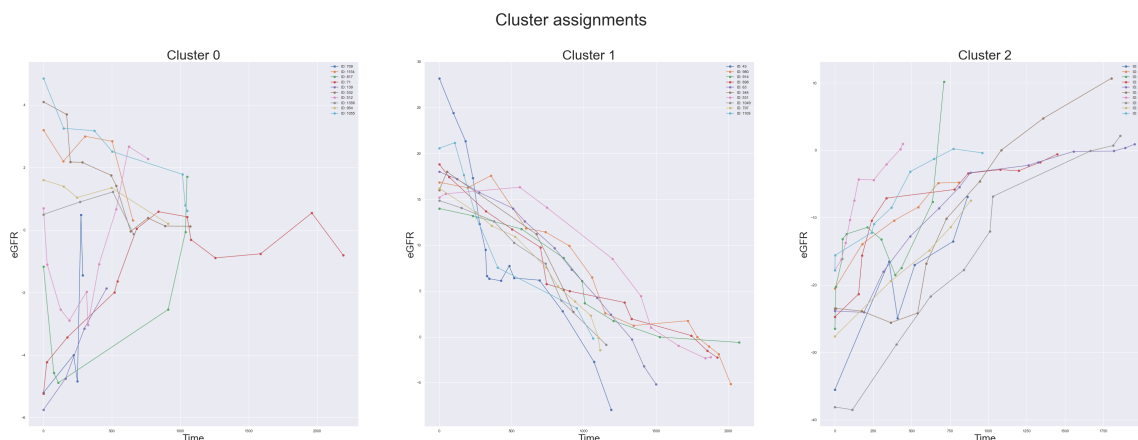


Figure 5.3: Subset of eGFR trajectories from the clusters formed using k -Means with $k = 3$.

5.2 CKD progression forecasting as a classification task

In this section, we explore the usefulness of classification models in very sparse data settings, and in particular, the case where only one data point is available for each patient. Our overall goal is to accurately predict how patients' CKD progression will evolve, where the three progression types are explained at the beginning of Section 5.1. To avoid any ambiguity, we state that the three patient classes in this classification problem correspond to patients whose CKD condition improves, worsens, or remains stable over time.

Having access to the progression status early on in the patient's CKD diagnosis stage can be very helpful in determining an appropriate choice of medical treatment. As an example, a patient whose CKD condition is predicted to worsen should be monitored more frequently compared to one whose condition is predicted to improve. Consequently, this can make a significant impact in optimising resource allocation and reducing costs for governments and healthcare organisations.

5.2.1 Dataset description

For this task, we convert our longitudinal dataset to a cross-sectional one, in which the independence assumption between distinct data points will now hold. This allows us to use various classical supervised learning algorithms, that are designed under such data assumptions. Our procedure for obtaining the cross-sectional dataset is simple. We firstly, merge the datasets from the Patras and Sheffield medical sites

into one, and add a medical site feature to each patient. Then, for each patient, we only keep the first data point (i.e. the time 0 data point) and drop the “Times” feature. Finally, to each patient, we assign a progression status target label according to its resulting cluster assignment produced by the time-series k -Means algorithm. Figure A.7 should give a good idea of how the resulting cross-sectional dataset looks like.

Let us clarify that ideally we would have wanted to build separate predictive models for each medical site as in Chapter 4, but this was not possible. The cross-sectional dataset consists of 1207 data points, of which 398 and 809 come from the Patras and Sheffield medical sites respectively. It is widely known that deep learning approaches tend to perform badly on small datasets, and since a significant portion of our work includes neural network models, we decided it was best to merge the data from the two medical sites. Another important feature in our dataset is *class imbalance*. We note that the dataset consists of 854 patients with a “stable” progression status, 249 patients with a “worsens” status, and 104 patients with an “imrpoves” status. These proportions are in par with what we usually observe in CKD progression in reality. The class imbalance issue in the dataset is a further complication in CKD progression forecasting, and renders this task even more challenging compared to the eGFR forecasting problem of Chapter 4.

Finally, no data pre-processing was required since we use the already clean longitudinal data from our previous work to get the cross-sectional dataset. The only extra step we took was to standardise numerical features, which is important when using certain models (e.g. for support vector machines).

5.2.2 Evaluation metrics

Aside from using the test set accuracy for model evaluation, we have decided to also use the Matthews Correlation Coefficient (MCC), a reliable statistical measure which is high only if predictions are sufficiently good *across all classes* in the data. The MCC metric has been shown to avoid some of the common pitfalls of using the accuracy and F1 scores in imbalanced data settings [Chicco and Jurman, 2020], such as yielding inflated results that appear to be too optimistic. We present the MCC score formula in the binary class setting for simplicity, and note that the idea extends naturally to the multi-class case. The MCC score, which takes values in $[-1, 1]$, is specified as

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} ,$$

where TP, TN, FP , and FN denote the number of true positive, true negative, false positive and false negative predictions respectively. We believe that using the test set MCC scores will allow us to draw more reliable conclusions regarding each model's actual performance.

Relevant to our Section 2.5 discussion around overconfidence in modern deep neural networks, we additionally evaluate our deep learning models using the concept of confidence calibration. The main idea behind confidence calibration is rather intuitive. Put simply, if a model predicts one of the classes with an 80% probability, then for a well-calibrated model that class should be the true one 80% of the time. A way to measure the confidence calibration error in a model is through the Expected Calibration Error (ECE). Intuitively, the ECE works by discretising the $[0, 1]$ probability interval into a fixed number of bins, and subsequently assigns each predicted probability to the bin that encompasses it. Then, the ECE is the difference between the proportion of bin predictions that are correct (this is the accuracy part) and the average bin probability (this is the confidence part).

To make this more concrete, consider M consecutive intervals partitioning $[0, 1]$, such that the m^{th} interval is specified as $I_m = \left(\frac{m-1}{M}, \frac{m}{M}\right]$. Then, define the bin of indices whose probabilities fall in the interval I_m , as $B_m = \{i: \hat{p}_i \in I_m\}$, where \hat{p}_i denotes the predicted probability at index i . Then:

- The *accuracy*, denoted by $acc(B_m)$, is given by $\frac{1}{|B_m|} \cdot \sum_{i \in B_m} \mathbb{I}[\hat{y}_i = y_i]$, where \hat{y}_i is the predicted class label at index i .
- The average *confidence*, denoted by $conf(B_m)$, is specified as $\frac{1}{|B_m|} \cdot \sum_{i \in B_m} \hat{p}_i$.

A well-calibrated model should have $acc(B_m) = conf(B_m)$, for each interval m . Finally, the ECE, which takes values in $[0, 1]$, is specified as

$$ECE = \sum_{m=1}^M \frac{|B_m|}{n} \cdot |acc(B_m) - conf(B_m)| ,$$

where n is the total number of data points.

5.2.3 Models specification

For our experiments, we use 80% of the data for training, and the remaining 20% for testing, while also using stratified sampling in our splitting process. In stratified splitting, both the training and test sets contain equal proportions of data points from each of the three classes. This is vital for ensuring that the models can generalise well, and they are not simply predicting the class label with the majority of members. Regarding the hyperparameter selection process, we perform exhaustive grid-search using five-fold cross-validation on the training set, with the average five-fold test set accuracy score being our evaluation criterion. In addition, results are obtained for *five different training & test set splits*, to calculate the mean accuracy, MCC and ECE (for the neural network models) scores, alongside their standard deviations. The models we consider, with their corresponding hyperparameters are:

1. **Logistic Regression:** The only hyperparameter here is the L2-regularisation strength parameter.
2. **Support Vector Machine:** The hyperparameters in this model include the regularisation strength parameter, and the kernel lengthscale parameter. Note that we only use a radial basis function as our kernel, since this choice often gives the best results.
3. **Random Forest:** We consider tuning the number of estimators used, maximum tree depth, maximum number of features considered in splitting and minimum number of data points in leaf nodes hyperparameters.
4. **Feed-forward neural network:** Due to the computational cost of neural network models, we only consider the number of hidden layers, and their respective number of nodes. We specify that for all the network layers, we use a rectified linear unit (ReLU) activation function, and learning and momentum rates of 0.001 and 0.9 respectively in the stochastic gradient descent optimisation routine.
5. **Bayesian neural network (LLLA):** This model is a Bayesian neural network with a last-layer Laplace approximation (LLLA) to the weights posterior distribution, as explained in Section 2.5.1.
6. **Bayesian neural network (SGLD):** This method is similar to the above one, with the only difference being that inference over the last-layer weights is

performed using the stochastic gradient Langevin dynamics (SGLD) sampling technique, as mentioned in Section 2.5.2.

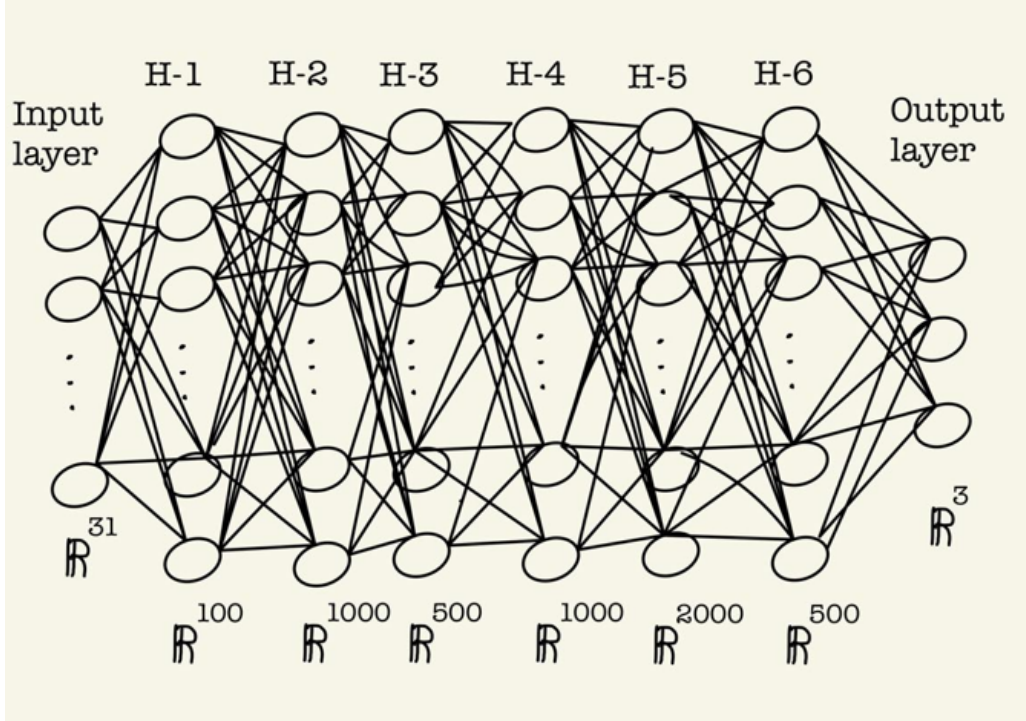


Figure 5.4: The architecture of the feed-forward neural network models 4-6, with the number of hidden units in each layer displayed at the bottom. For example, a layer labelled as \mathbb{R}^{500} indicates that there are 500 hidden units in that layer.

Let us mention that for the Bayesian neural network models, we use the same final network architecture as in the feed-forward network case to allow for a fair comparison between the two approaches. The network architecture used for these three models is shown in Figure 5.4, where we highlight that hidden layer 6 (H6) denotes the last layer, whose weights we treat as random variables in models 5 and 6. Bayesian inference is only performed over the last layer weights for computational reasons, since this neural network model contains more than four million parameters in total, and so using either of the two posterior inference techniques is clearly infeasible.

Regarding the prior used for the Bayesian neural network models, we place an independent $\mathcal{N}(0, 1)$ prior on each of the 1500 last-layer weights. In turn, predictions are made using the posterior predictive distribution, which is approximated using Monte Carlo integration, as explained in (2.22). For the Monte Carlo integration part, we use 4,000 and 100,000 posterior samples for the Bayesian neural network

SGLD and LLLA models respectively. Ideally, it would have been better to have more posterior samples for the SGLD case, to obtain a more accurate approximation to the predictive distribution. However, the computational overhead of MCMC sampling methods in such high-dimensional settings is massive, which explains why scalable sampling techniques still remain a very active area of research.

Model	Test set accuracy	Test set MCC	ECE
Logistic Regression	0.704 ± 0.0130	0.128 ± 0.0661	-
Support Vector Machine	0.717 ± 0.00496	0.187 ± 0.0510	-
Random Forest	0.719 ± 0.00978	0.189 ± 0.0457	-
Feed-forward neural network	0.734 ± 0.00413	0.224 ± 0.0259	0.405 ± 0.0241
Bayesian neural network (LLLA)	0.721 ± 0.0193	0.275 ± 0.0486	0.327 ± 0.0352
Bayesian neural network (SGLD)	0.724 ± 0.0122	0.245 ± 0.0432	0.405 ± 0.0276

Table 5.3: Mean and standard deviation of test set accuracy, MCC and ECE scores for the CKD patient progression status classification task.

5.2.4 Results & analysis

In Table 5.3, we display the average accuracy and MCC test set scores over our five different repeats of the experiment, along with their standard deviations. For the neural network models, we also include the average ECE, with its standard deviation. Let us state that the standard deviation values are very reasonable, which indicates that the models have stable performance over different train & test splits. At first, one might naively say that based on the accuracy scores, most models have very similar predictive capabilities. However, as we mentioned before, in imbalanced-data settings, focusing only on accuracy can lead us to false conclusions. This is in fact highlighted in the MCC scores column, where the difference amongst models is significantly bigger. Keeping in mind that 70.8% of the patients belong to the stable progression category, we believe that this number should serve as our benchmark accuracy score for this specific task.

Starting with the logistic regression model, we clearly observe unsatisfactory performance, indicated by both an accuracy score below the aforementioned benchmark but also by the significantly lower MCC score compared to other models. Moving to the support vector machine and random forest models, we see that the models' accuracy score beats the benchmark, and their MCC scores imply that they have learned something useful, even if not to a great extent. Finally, the neural network

models appear to be the best class amongst the models considered, with superior performance in terms of both accuracy and MCC. An interesting observation is that even though our feed-forward neural network performs best in terms of accuracy, it has the lowest average MCC score amongst the three deep learning models. This perhaps suggests that the model performs very well on predicting one or two of the classes present, but does poorly on the remaining. In contrast, the Bayesian neural network (LLLA) model has by far the best MCC score, which is suggestive that it has the best performance when considering predictions over all three classes. To further add to this, the substantially lower ECE of this model also confirms that its predictions are more well-calibrated, and the model seems to generalise better. Instead, the other two neural network models seem to perform really well on certain classes but do badly on the remaining class. The receiver operating characteristic (ROC) curves for the two top-performing models are displayed in Figure 5.5, and gives us a sense of how the models perform across different classification thresholds.

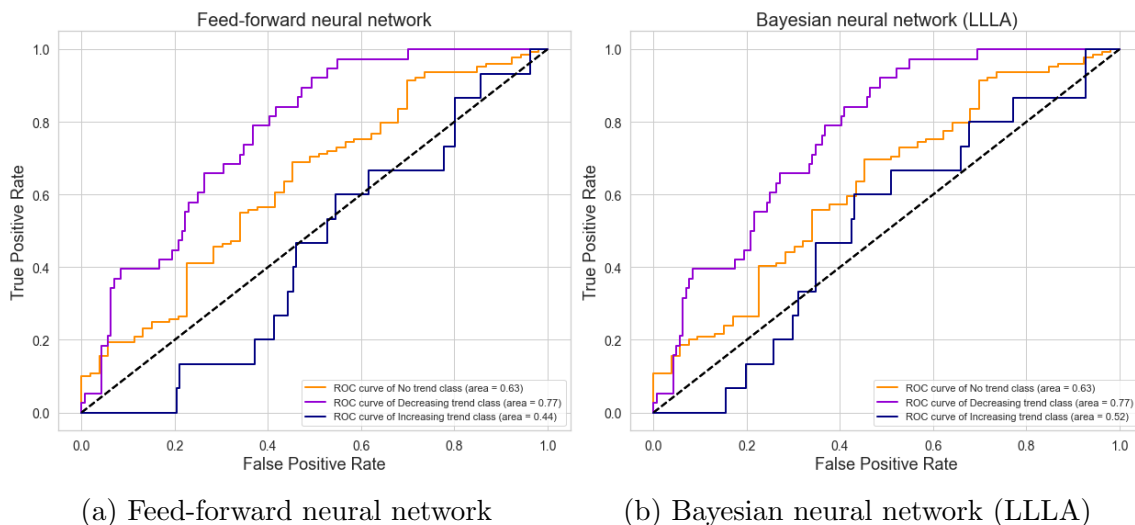


Figure 5.5: ROC curves for test set predictions, at a specific run of the experiments.

Finally, we speculate on the MCC performance difference between the two Bayesian neural network models, which only differ in their respective posterior inference techniques. As discussed in the models specification section, due to computational costs, for the SGLD model we were only able to use 4,000 posterior draws for our predictive distribution approximation. Perhaps, this number is not enough to yield a sufficiently accurate approximation, even though our convergence diagnostics suggest that these samples are from the true weight posterior distribution. Judging by

the impressive Bayesian neural network (LLLA) results, one could conclude that the true weight posterior is approximately Gaussian, and hence the conceptually simple Laplace approximation does the job. The difference in prediction overconfidence is further highlighted by observing Figure 5.6, where we see that the distribution of the predicted label probabilities for the Bayesian neural network (LLLA) model places less density on large probability values. This indicates that being Bayesian about the model weights, even if just with respect to the last layer, can help in tackling prediction overconfidence in deep learning models.

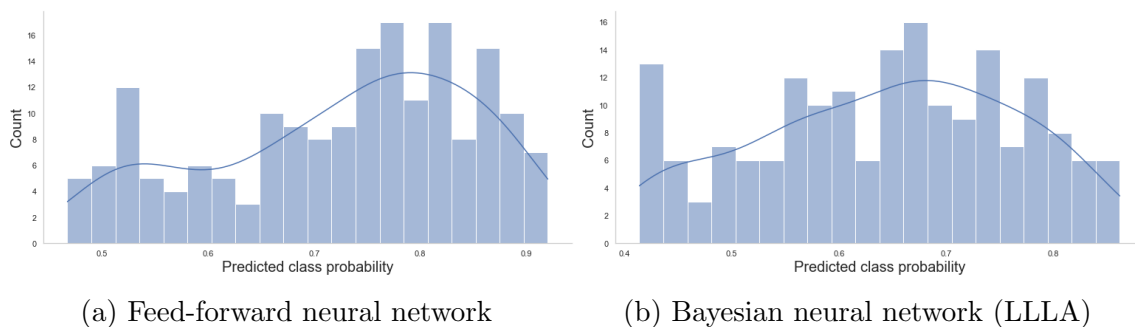


Figure 5.6: Predicted label probabilities on the test set, at a specific run of the experiments.

5.3 Conclusion

We finish this chapter with some thoughts on the practical usefulness of the classification models developed. Even though model performance on the CKD progression status classification task is far from perfect, we also recognise the difficulty this task entails, as we restrict ourselves to making predictions using only the first data point from each patient’s history. Nonetheless, the majority of the models were still able to find meaningful patterns in the data, as proven by their ability to beat the benchmark accuracy score. An important finding in this chapter is that, our Bayesian neural network (LLLA) model not only managed to outperform all other models in terms of more balanced predictions across the different classes, but also produced better calibrated predictive uncertainty, the latter of which is of utmost importance to healthcare professionals.

Chapter 6

Conclusion and Future Directions

6.1 Summary

We conclude this thesis with a concise summary of the key findings from our experiments, along with potential areas to further advance this work. Our experiments suggest that Gaussian Process boosting has the potential to significantly outperform linear mixed-effects models in eGFR forecasting, and could have a big impact on the healthcare industry. Moreover, and specific to the GP Boost algorithm, we experimentally validated a method to improve the learning process of the Gaussian Process kernel hyperparameters, through the use of prior distributions. In fact, we have shown that our proposed method not only provides an improvement in terms of test set RMSE scores, but also leads to more reliable predictive uncertainty in the Gaussian Process component of the model. We believe that this will be of particular interest to healthcare professionals, especially in settings of limited patient data availability. Regarding our second approach to the CKD progression task, we have successfully proposed a novel methodology to forecast patient progression status using just a single observation per patient. In turn, we show that state of the art Bayesian neural network models offer superior performance in terms of MCC on imbalanced data, and yield better-calibrated predictions over their non-probabilistic counterparts.

6.2 Future directions

Let us now discuss some future directions to further improve this work, which were not explored due to time constraints. Firstly, we mention that our method for im-

proving the learning process in GP Boost is, strictly speaking, not fully Bayesian since predictions are made using point estimates for the hyperparameters (e.g. the posterior distribution mean). Instead, a fully Bayesian approach would involve using all posterior samples to obtain an approximation to the posterior predictive distribution, as exhibited in equation (2.22). In this way, one can fully exploit the benefits of Bayesian modelling, by incorporating model uncertainty in the predictions in a principled and systematic manner. However, doing this would involve modifying the complex `GPBoost` library source code, which is a rather challenging task given our timeframe. Secondly, we observe that the GP Boost model using approximate Gaussian Process inference techniques was hindered by the quality of the Vecchia approximation. Given the model’s impressive performance on predicting unseen patients, we believe that a useful extension would be to investigate whether using alternative approximations, such as the method of inducing variables, could improve prediction on patients observed in the training data. Unfortunately, the `GPBoost` library does not yet support other approximation techniques. Lastly, in our CKD progression status classification task, we only consider the case where a single data point is available per patient. As we saw, obtaining precise forecasts in this setting can be a difficult task. If we had more time, it would be interesting to repeat the experiments using the first two or three data points of each patient, and see how performance varies as a function of the per patient number of data points. This would provide helpful insights on the methodology of our second approach to CKD progression forecasting, under various limited patient data settings.

Bibliography

- [Anh Luong and Chandola, 2017] Anh Luong, D. T. and Chandola, V. (2017). A K-Means Approach to Clustering Disease Progressions. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 268–274.
- [Bhatt et al., 2017] Bhatt, S., Cameron, E., Flaxman, S. R., Weiss, D. J., Smith, D. L., and Gething, P. W. (2017). Improved Prediction Accuracy for Disease Risk Mapping using Gaussian Process Stacked Generalization. *Journal of The Royal Society Interface*, 14(134):20170520.
- [Chen et al., 2020] Chen, I. Y., Joshi, S., Ghassemi, M., and Ranganath, R. (2020). Probabilistic Machine Learning for Healthcare. *arXiv preprint arXiv:2009.11087*.
- [Cheng et al., 2018] Cheng, L., Ramchandran, S., Vatanen, T., Lietzen, N., Lahesmaa, R., Vehtari, A., and Lähdesmäki, H. (2018). LonGP: an Additive Gaussian Process Regression Model for Longitudinal Study Designs. *bioRxiv*, page 259564.
- [Chicco and Jurman, 2020] Chicco, D. and Jurman, G. (2020). The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation. *BMC genomics*, 21(1):1–13.
- [Dürichen et al., 2014] Dürichen, R., Pimentel, M. A., Clifton, L., Schweikard, A., and Clifton, D. A. (2014). Multitask Gaussian Processes for Multivariate Physiological Time-Series Analysis. *IEEE Transactions on Biomedical Engineering*, 62(1):314–322.
- [Freund and Schapire, 1999] Freund, Y. and Schapire, R. (1999). A Short Introduction to Boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.
- [Freund et al., 1996] Freund, Y., Schapire, R. E., et al. (1996). Experiments with a New Boosting Algorithm. Citeseer.

- [Futoma et al.,] Futoma, J., Sendak, M. P., Cameron, B., and Heller, K. A. Scalable Joint Modeling of Longitudinal and Point Process Data for Disease Trajectory Prediction and Improving Management of Chronic Kidney Disease. Citeseer.
- [Gad and El Kholy, 2012] Gad, A. M. and El Kholy, R. B. (2012). Generalized Linear Mixed Models for Longitudinal Data.
- [Gelman et al., 1995] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian Data Analysis*. Chapman and Hall/CRC.
- [Go et al., 2004] Go, A. S., Chertow, G. M., Fan, D., McCulloch, C. E., and Hsu, C.-y. (2004). Chronic Kidney Disease and the Risks of Death, Cardiovascular Events, and Hospitalization. *New England Journal of Medicine*, 351(13):1296–1305.
- [Goan and Fookes, 2020] Goan, E. and Fookes, C. (2020). Bayesian Neural Networks: An Introduction and Survey. In *Case Studies in Applied Bayesian Data Science*, pages 45–87. Springer.
- [Grenander and Miller, 1994] Grenander, U. and Miller, M. I. (1994). Representations of Knowledge in Complex Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 56(4):549–603.
- [Groll and Tutz, 2012] Groll, A. and Tutz, G. (2012). Regularization for Generalized Additive Mixed Models by Likelihood-Based Boosting. *Methods of Information in Medicine*, 51(02):168–177.
- [Grün, 2019] Grün, B. (2019). Model-Based Clustering. In *Handbook of mixture analysis*, pages 157–192. Chapman and Hall/CRC.
- [Hafeez AR, 2016] Hafeez AR, Idrees MK, A. S. (2016). Accuracy of GFR Estimation Formula in Determination of Glomerular Filtration Rate in Kidney Donors: Comparison with 24 h Urine Creatinine Clearance. *Saudi J Kidney Dis Transpl.* 2016;27(2):320-325. doi:10.4103/1319-2442.178551.
- [Hajjem et al., 2011] Hajjem, A., Bellavance, F., and Larocque, D. (2011). Mixed Effects Regression Trees for Clustered Data. *Statistics Probability Letters*, 81(4):451–459.
- [Hajjem et al., 2014] Hajjem, A., Bellavance, F., and Larocque, D. (2014). Mixed-Effects Random Forest for Clustered Data. *Journal of Statistical Computation and Simulation*, 84(6):1313–1328.

- [Jain et al., 1999] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data Clustering: A Review. *ACM computing surveys (CSUR)*, 31(3):264–323.
- [Katzfuss and Guinness, 2021] Katzfuss, M. and Guinness, J. (2021). A General Framework for Vecchia Approximations of Gaussian Processes. *Statistical Science*, 36(1).
- [Kaufmann, 1987] Kaufmann, L. (1987). Clustering by Means of Medoids. In *Proc. Statistical Data Analysis Based on the L1 Norm Conference, Neuchatel, 1987*, pages 405–416.
- [Kristiadi et al., 2020] Kristiadi, A., Hein, M., and Hennig, P. (2020). Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5436–5446. PMLR.
- [Li and Biswas, 1999] Li, C. and Biswas, G. (1999). Temporal Pattern Generation Using Hidden Markov Model Based Unsupervised Classification. In Hand, D. J., Kok, J. N., and Berthold, M. R., editors, *Advances in Intelligent Data Analysis*, pages 245–256, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Liao, 2005] Liao, T. W. (2005). Clustering of Time Series Data—a Survey. *Pattern recognition*, 38(11):1857–1874.
- [Lintu et al., 2022] Lintu, M., Shreyas, K., and Kamath, A. (2022). A Multi-State Model for Kidney Disease Progression. *Clinical Epidemiology and Global Health*, 13:100946.
- [Luong et al., 2017] Luong, D. T. A., Tran, D., Pace, W. D., Dickinson, M., Vassalotti, J., Carroll, J., Withiam-Leitch, M., Yang, M., Satchidanand, N., Staton, E., et al. (2017). Extracting Deep Phenotypes for Chronic Kidney Disease using Electronic Health Records. *eGEMs*, 5(1).
- [Moins et al., 2022] Moins, T., Arbel, J., Dutfoy, A., and Girard, S. (2022). On the Use of a Local R-Hat to Improve MCMC Convergence Diagnostic.
- [Neal et al.,] Neal, R. M. et al. MCMC using Hamiltonian Dynamics.

- [Petitjean et al., 2011] Petitjean, F., Ketterlin, A., and Gançarski, P. (2011). A Global Averaging Method for Dynamic Time Warping, with Applications to Clustering. *Pattern Recognition*, 44(3):678–693.
- [Rand, 1971] Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850.
- [Rasmussen and Williams, 2017] Rasmussen, C. E. and Williams, C. K. (2017). Gaussian Processes for Machine Learning. 2006. *Cited on*, 95.
- [Rizopoulos, 2011] Rizopoulos, D. (2011). Dynamic Predictions and Prospective Accuracy in Joint Models for Longitudinal and Time-to-Event Data. *Biometrics*, 67(3):819–829.
- [Roberts et al., 2013] Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. (2013). Gaussian Processes for Time-Series Modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20110550.
- [Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.
- [Sakoe and Chiba, 1978] Sakoe, H. and Chiba, S. (1978). Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, 26:43–49.
- [Schulam and Saria, 2016] Schulam, P. and Saria, S. (2016). A Framework for Individualizing Predictions of Disease Trajectories by Exploiting Multi-Resolution Structure. *arXiv preprint arXiv:1601.04674*.
- [Schulam et al., 2015] Schulam, P., Wigley, F., and Saria, S. (2015). Clustering Longitudinal Clinical Marker Trajectories from Electronic Health Data: Applications to Phenotyping and Endotype Discovery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.
- [Sigrist, 2018] Sigrist, F. (2018). Gradient and Newton Boosting for Classification and Regression. *arXiv preprint arXiv:1808.03064*.
- [Sigrist, 2020] Sigrist, F. (2020). Gaussian Process Boosting.

- [Snelson and Ghahramani, 2005] Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian Processes using Pseudo-Inputs. *Advances in neural information processing systems*, 18.
- [Soleimani et al., 2018] Soleimani, H., Hensman, J., and Saria, S. (2018). Scalable Joint Models for Reliable Uncertainty-Aware Event Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(8):1948–1963.
- [Stroup, 2012] Stroup, W. (2012). *Generalized Linear Mixed Models: Modern Concepts, Methods and Applications*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis.
- [Tangri et al., 2011] Tangri, N., Stevens, L. A., Griffith, J., Tighiouart, H., Djurdjev, O., Naimark, D., Levin, A., and Levey, A. S. (2011). A Predictive Model for Progression of Chronic Kidney Disease to Kidney Failure. *Jama*, 305(15):1553–1559.
- [USRDS, 2003] USRDS, U. (2003). Annual Data Report: Atlas of End-Stage Renal Disease in the United States. *Bethesda, MD: National Institutes of Health, National Institute of Diabetes and Digestive and Kidney Diseases*.
- [van Dijk et al., 2008] van Dijk, P. C., Jager, K. J., Zwinderman, A. H., Zoccali, C., and Dekker, F. W. (2008). The Analysis of Survival Data in Nephrology: Basic Concepts and Methods of Cox Regression. *Kidney International*, 74(6):705–709.
- [Vantini et al., 2022] Vantini, M., Mannerström, H., Rautio, S., Ahlfors, H., Stockinger, B., and Lähdesmäki, H. (2022). PairGP: Gaussian Process Modeling of Longitudinal Data from Paired Multi-Condition Studies. *Computers in Biology and Medicine*, 143:105268.
- [Varin et al., 2011] Varin, C., Reid, N., and Firth, D. (2011). An Overview of Composite Likelihood Methods. *Statistica Sinica*, 21(1):5–42.
- [Verbeke, 1997] Verbeke, G. (1997). Linear Mixed Models for Longitudinal Data. In *Linear mixed models in practice*, pages 63–153. Springer.
- [Xiong and Yeung, 2002] Xiong, Y. and Yeung, D.-Y. (2002). Mixtures of ARMA Models for Model-Based Time Series Clustering. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 717–720.

[Zilber and Katzfuss, 2021] Zilber, D. and Katzfuss, M. (2021). Vecchia-Laplace Approximations of Generalized Gaussian Processes for Big Non-Gaussian Spatial Data. *Computational Statistics & Data Analysis*, 153:107081.

Appendix A

Additional figures



Figure A.1: Histogram of the Silhouette coefficient values for the k -Means algorithm.

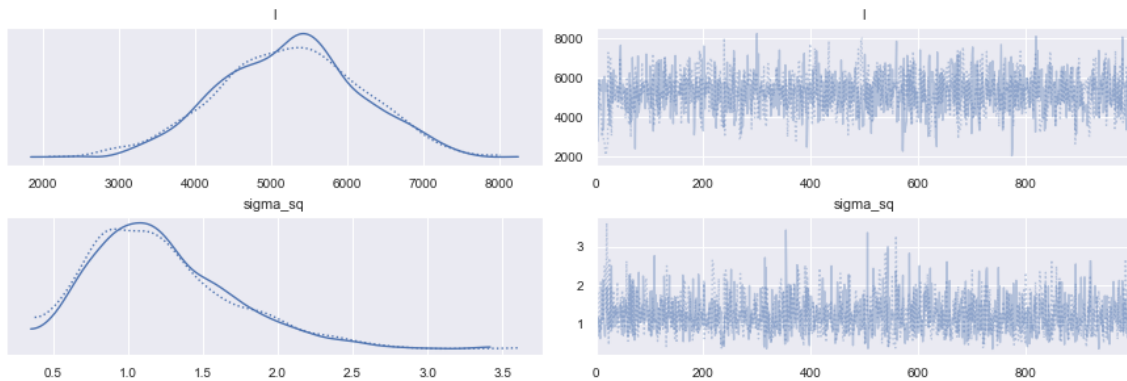
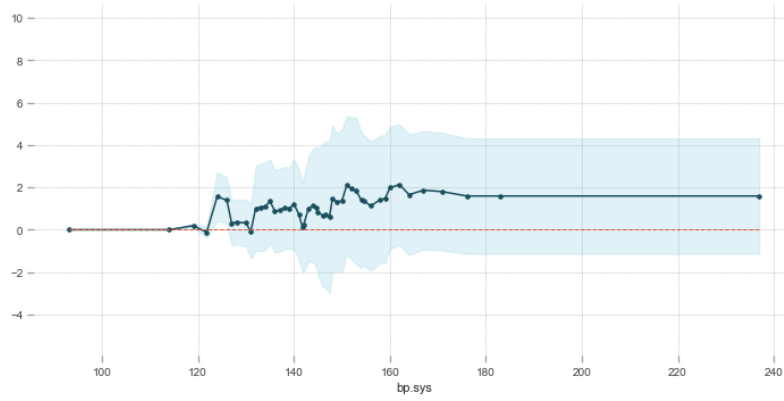


Figure A.2: Posterior samples (right column) and their respective approximate distributions (left column) shown for two distinct Markov chains, computed using data from patient 3.

	ID	age	gender	ethnicity	height	weight	smoker	kidney_transplant	patient_died	disease	bp.sys	bun	egfr	times
0	0	59.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	150.600000	8.0	73.0	0
1	0	59.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	143.000000	8.0	62.0	168
2	0	59.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	150.000000	6.0	74.0	294
3	0	60.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	138.000000	7.0	77.0	492
4	0	60.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	144.000000	7.0	70.0	608
5	0	61.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	150.600000	6.0	74.0	874
6	0	61.0	Male	Unknown	1.71	132.200000	Unknown	False	False	GMN	178.000000	6.7	81.0	1011
7	1	55.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	131.000000	8.0	42.0	0
8	1	55.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	118.000000	15.0	4.0	175
9	1	55.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	164.000000	7.0	49.0	313
10	1	56.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	144.833333	8.0	41.0	502
11	1	57.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	147.000000	8.0	44.0	754
12	1	57.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	163.000000	11.0	32.0	883
13	1	57.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	144.833333	8.5	37.0	1037
14	1	58.0	Female	Caucasian	1.61	73.300000	Unknown	True	True	Transplant	146.000000	9.1	30.0	1060
15	2	56.0	Male	Caucasian	1.74	67.666667	Unknown	True	False	Transplant	108.000000	12.0	9.0	0
16	2	56.0	Male	Caucasian	1.74	67.666667	Unknown	True	False	Transplant	96.000000	23.0	4.0	158
17	2	56.0	Male	Caucasian	1.74	67.666667	Unknown	True	False	Transplant	153.000000	10.0	75.0	182
18	2	57.0	Male	Caucasian	1.74	67.666667	Unknown	True	False	Transplant	125.750000	10.0	69.0	371
19	2	57.0	Male	Caucasian	1.74	67.666667	Unknown	True	False	Transplant	117.000000	8.0	71.0	532

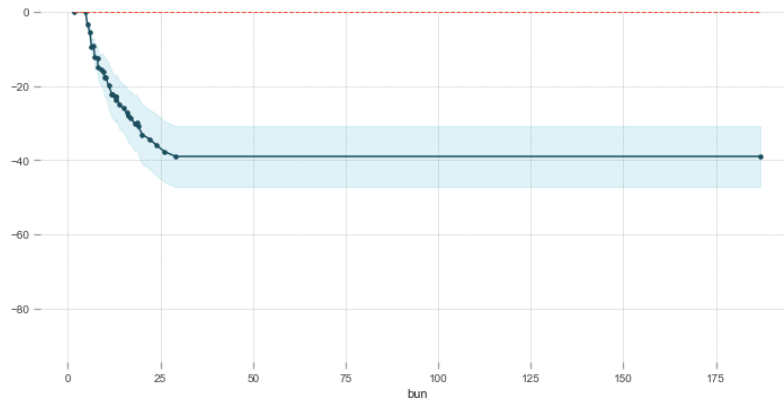
Figure A.3: Snapshot of our longitudinal CKD patient dataset.

PDP for feature "bp.sys"
Number of unique grid points: 48



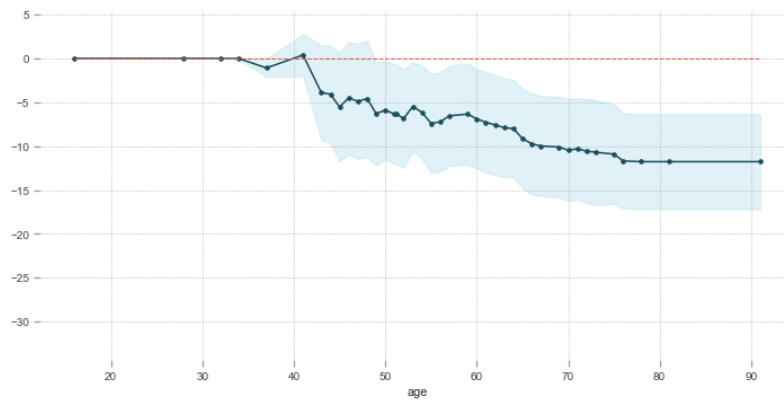
(a) Systolic blood pressure

PDP for feature "bun"
Number of unique grid points: 33



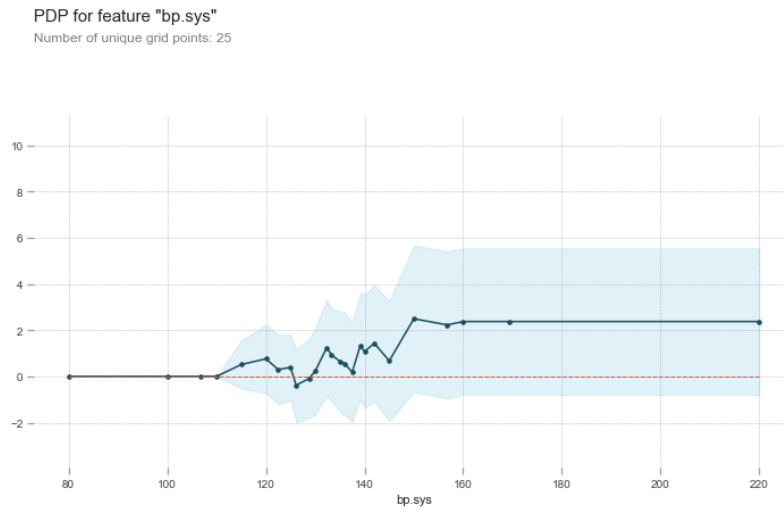
(b) Blood urea nitrogen levels

PDP for feature "age"
Number of unique grid points: 41

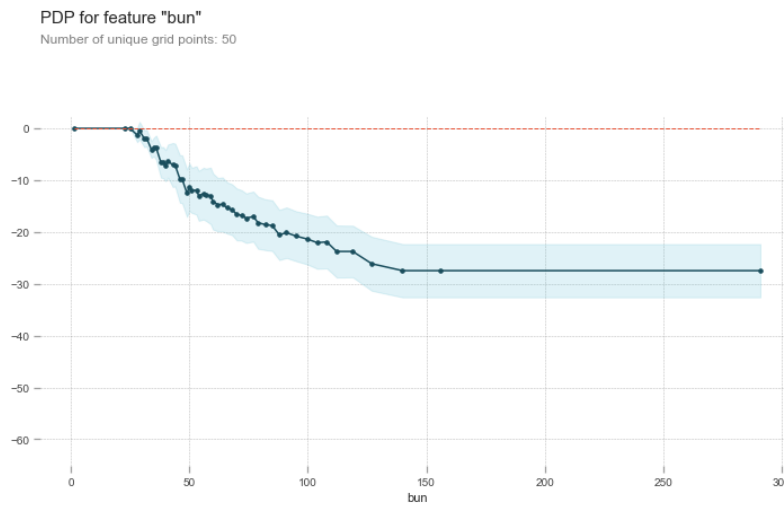


(c) Age

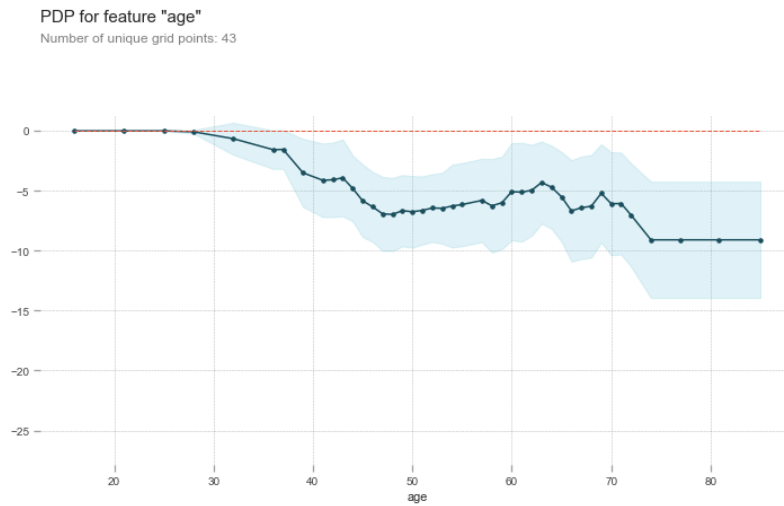
Figure A.4: Partial dependence plot for the three most important features *in the Sheffield data*. The vertical and horizontal axes denote eGFR and the corresponding feature values respectively, the line shows the conditional mean function, and the shaded area is a 95% confidence interval centred at the mean function.



(a) Systolic blood pressure



(b) Blood urea nitrogen levels



(c) Age

Figure A.5: Partial dependence plot for the three most important features *in the Patras data*. The vertical and horizontal axes denote eGFR and the corresponding feature values respectively, the line shows the conditional mean function, and the shaded area is a 95% confidence interval centred at the mean function.







Stage	Description	eGFR	Kidney Function
1	Possible kidney damage (e.g., protein in the urine) with normal kidney function	90 or above	 90-100%
2	Kidney damage with mild loss of kidney function	60-89	 60-89%
3a	Mild to moderate loss of kidney function	45-59	 45-59%
3b	Moderate to severe loss of kidney function	30-44	 30-44%
4	Severe loss of kidney function	15-29	 15-29%
5	Kidney failure	Less than 15	 Less than 15%

Figure A.6: Conversion table relating a patient's eGFR value and Chronic Kidney Disease stage.

	ID	site	age	gender	ethnicity	height	weight	smoker	kidney_transplant	patient_died	disease	bp.sys	bun	egfr	Labels
0	0	Sheffield	51.0	Male	Caucasian	1.720	80.225	Past-Smoker	False	False	DN	132.0	12.000	31.0	0
1	1	Sheffield	68.0	Male	Unknown	1.690	80.225	Unknown	False	False	GMN	172.0	9.000	45.0	0
2	2	Sheffield	50.0	Female	Black	1.630	80.225	Unknown	False	False	GMN	159.5	8.700	39.0	1
3	3	Sheffield	51.0	Male	Caucasian	1.750	80.225	Past-Smoker	False	False	Vascular	151.0	10.000	36.0	0
4	4	Patras	61.0	Female	Caucasian	1.670	74.333	Smoker	False	False	HTN	130.0	88.000	32.0	0
...
1202	1202	Patras	77.0	Female	Caucasian	1.590	82.250	Non-Smoker	False	False	DN	144.0	68.000	31.0	0
1203	1203	Sheffield	53.0	Male	Caucasian	1.725	80.225	Unknown	False	False	HTN	135.0	37.954	22.0	0
1204	1204	Patras	52.0	Female	Caucasian	1.640	71.200	Smoker	False	False	GMN	132.0	31.000	62.0	2
1205	1205	Sheffield	70.0	Male	Caucasian	1.725	80.700	Unknown	False	False	GMN	152.0	6.000	59.0	0
1206	1206	Patras	56.0	Male	Caucasian	1.730	80.000	Non-Smoker	True	False	Transplant	130.0	36.000	82.0	0

Figure A.7: Snapshot of our cross-sectional CKD patient dataset.