UNIVERSITY OF CAMBRIDGE
Department of Engineering

Interpretation of Multivariate Machine Learning Models Based on Stochastic Processes

Author Name: Luke Peart
Supervisor: Prof. Carl Rasmussen
Date: 31/05/23

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed _L Peart_____ *date* _31/05/23_____

Final Project Report
Interpretation of Multivariate Machine Learning Models Based on Stochastic Processes

Luke Peart (lp551@cam.ac.uk)
St. Catharine's College

Technical Abstract

This project aims to have developed a software tool for Gaussian Process modelling and analysis, aimed at providing users with an intuitive and powerful platform for interpretation and predictive modelling. The theory for the software tool is predicated on previous work, which allows the tool to incorporate various functionalities, including model training, interpretation of results, and visualization of data. It implements efficient algorithms to handle datasets of different sizes and dimensions. Theoretical results demonstrate the accuracy and utility of the software tool by evaluating the quality of predictions and visualizing component plots. Comparisons with existing models and analyses validate the tool's effectiveness in capturing data patterns and improving predictive performance. The software results showcase the efficiency and performance gains achieved through optimization efforts. Runtime measurements demonstrate improvements over alternative solutions. This provides users with an accessible platform to leverage the power of Gaussian Processes in data analysis, visualization, and prediction. The tool and future work suggestions provide a solid basis for future advancements and improvements in this domain, contributing to the broader field of machine learning and data analysis.

This technical report presents a comprehensive analysis of the problem space, the theoretical framework, software development process, and the evaluation of a novel software tool. The introduction provides an overview of the problem domain, setting the context for the subsequent sections. The theoretical framework section explores the fundamental concepts of Gaussian Process and discusses insights gleaned from previous papers. Moving forward, the report elucidates the software development process, covering the architectural design, the adopted methodology, and the implementation details. Subsequently, the report presents the results obtained from both theoretical investigations and software performance evaluations. The conclusion summarizes the findings and offers suggestions for further research. Lastly, an appendix is appended to the report, providing additional mathematical details to supplement the discussion. Overall, this report serves as a comprehensive resource for understanding the problem space, theoretical foundations, software development process, and results pertaining to the software tool under investigation.

1 Introduction

In the rapidly evolving field of machine learning, understanding and interpreting the complex relationships between input features and model outputs is crucial. As machine learning models become increasingly sophisticated and powerful, there is a growing demand for tools that facilitate human evaluation and comprehension of these interactions. This project aims to address this need by developing an interactive tool that enables in-depth exploration and analysis of the relationships between multivariate input features and the output generated by a machine learning model.

Previous research has made significant progress in the development of machine learning algorithms and models, resulting in remarkable achievements across various domains. However, as models become more complex and datasets grow larger, the interpretability of these models becomes a critical challenge. Many machine learning algorithms excel at making accurate predictions and classifications, but often lack transparency in regard to understanding the underlying computations. This limitation hinders their application in domains where interpretability and explainability are of utmost importance, such as healthcare, finance, and environmental monitoring.

Several approaches have been proposed to tackle the interpretability challenge, ranging from post-hoc explainability techniques to inherently interpretable models. This project uses the latter, specifically a model considered to be inherently interpretable that was proposed in a recent (2022) paper [2]. The proposed model suggests that it is able to provide an intuitive understanding of the complex interactions between input features and model outputs. By providing a user-friendly interface and visualizations, the tool will use the model to facilitate human analysis and exploration of the intricate connections that may exist within datasets.

2 Theoretical Framework and methodology

The theoretical framework of this project relies upon the paper "*Additive Gaussian Processes Revisited*" [2]. This paper serves as a key reference and forms the basis of understanding for the scope and principles underlying the project. In order to ensure a comprehensive grasp of the subject matter, it was imperative to establish a solid foundation in the field. This was achieved by first delving into the fundamentals of Gaussian Processes and subsequently examining the seminal work developed in the paper "*Additive Gaussian Processes"* [1].

2.1 Gaussian Processes

A Gaussian Process (GP) can be described as a collection of random variables (RVs) that may potentially be infinite. The defining characteristic of a GP is that the joint distribution of any finite subset of these RVs follows a multivariate Gaussian distribution. This can be combined with Bayesian theory, using a GP prior conditioned on training data allows for modelling of the joint distribution of training observations and prediction of test points. These concepts are illustrated in the figure below.

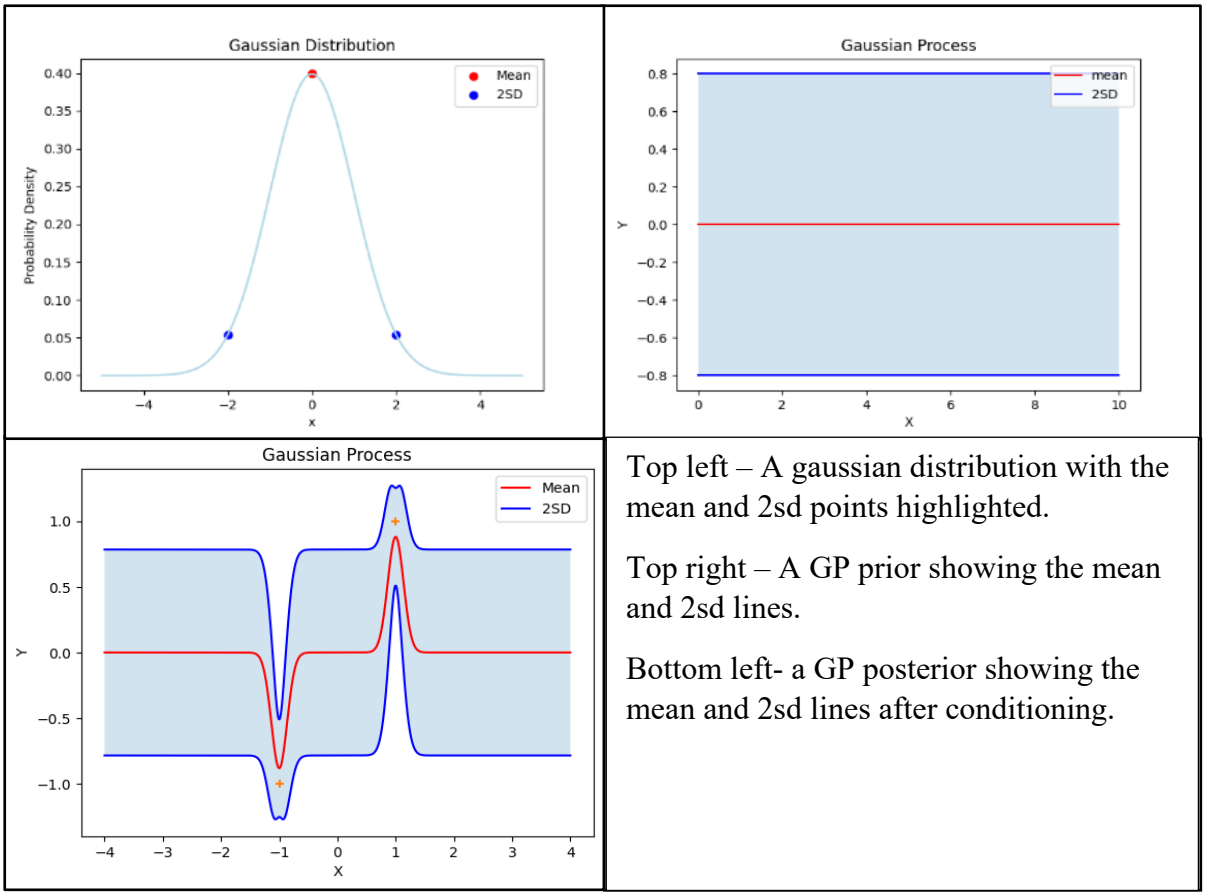

Figure 1 – graphics showing the concepts discussed in the first paragraph of section 2.1. A random variable (RV) can be characterised as having a gaussian distribution, giving the probability density function shown in the top left. A potentially infinite collection of these RVs is a Gaussian Process (GP) prior shown in the top right, with the same colours for the mean and 2sd points being used to illustrate the relation to the gaussian distribution. The bottom left shoes a GP posterior and demonstrates how the prior changes to accommodate observed datapoints (datapoints shown with orange plus symbols)

A GP can be completely characterized by its mean function and covariance function. The mean function represents the average or expected value of the process at any given input point. The covariance function captures the relationships between different input points in the process. For the theory behind this project, the GP models do not directly use a mean function, thus only the covariance function needs be considered.

The particular kernel that is used for the covariance function in this project is the squared exponential kernel (also known as the RBF kernel). This kernel takes the form:

$$k(x, x') = \sigma_f \exp\left(-\frac{(x - x')^2}{2l^2}\right)$$

The σ_f (output variance) and l (lengthscale) in this formula are free parameters that are able optimised by the model to achieve better performance.

2.2 Additive Kernels

Two differing styles of regression models commonly used are Generalized Additive Models (GAMs) and kernel-based models, specifically Gaussian process models with a squared-exponential (SE-GP) kernel. The advantage of GAMs is that they are easy to interpret and fit, however the disadvantage is that they become challenging with an increasing number of terms.

Alternatively, SE-GPs allow dependence on all input variables simultaneously, allowing them to deal well with a higher number of terms, but these models struggle with generalization.

The paper *"Additive Gaussian Processes"* [1] introduces an additive Gaussian process model that aims to address the limitations of the above models by introducing a kernel that combines the strengths of both approaches. The Additive kernel considers every possible combination of features from high dimensional data.

A graphical representation showing the difference and similarities of these approaches is shown below. From this figure it can be seen that the Additive kernel manages to incorporate interactions of various orders, from 1st order (GAM-like) to Dth order (SE-GP-like).

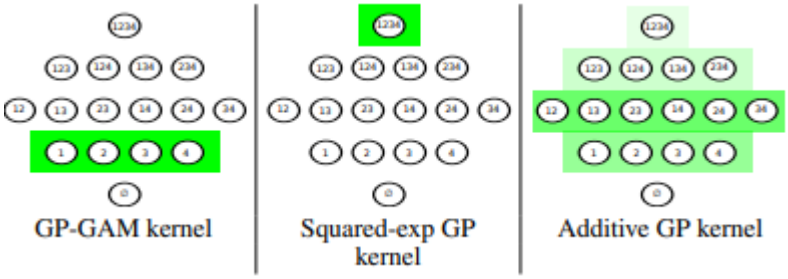

Figure 2 – A figure taken from original paper [1] demonstrating a comparison of components used by different models. Left is a GP-GAM that only uses the 1st order components. Middle is an SE-GP that only uses the Dth order component. Right is the Additive kernel that make use of all components.

The construction of the full Additive kernel involves a straightforward process. First, the base kernels are computed, which serve as fundamental building blocks. Then, each order of interaction is evaluated separately, considering different levels of interaction between elements in the system. Within each order, all combinations of elements are computed, and these combinations are then summed, resulting in a single kernel for each order. The summed values are multiplied by a corresponding order parameter, which represent the relative importance of each order. Finally, the contributions from all of the different orders are added together to form the complete Additive kernel. A more detailed process is included in the appendix section, which provides additional mathematical details and algorithms.

The result of this is a kernel that contains only marginally more hyperparameters than a GP-GAM (2D+1 for GP-GAM, 3D+1 for Additive). The extra hyperparameters are not only useful for modelling, but they can also be used in identifying which orders contribute most to the model, adding extra interpretability to the model.

Standard computation of this kernel can be proved to increase exponentially as the dimension of the dataset increases. Using basic theory from combinatorics shows that the cost is $O(2^D)$, where D is the number of features/dimensions of the dataset. To address this problem, the paper presents a much more efficient alternative method of computation. Using polynomial theory and the Newton-Girard formulae to calculate each order of the kernel reduces the cost to $O(D^2)$.

2.3 Orthogonal Additive Kernels

Although the Additive kernel proves to be effective for numerous datasets, it encounters challenges when dealing with certain datasets due to the reliance on high-order components that are difficult to visualize. The paper "*Additive Gaussian Processes Revisited*" [2] identified

this issue as an identifiability problem. The identifiability issue refers to the occurrence of the higher-order terms absorbing the effects of lower-order terms and the lower -order terms absorbing the effects of higher -order terms. To illustrate this concept visually, the diagram below is presented. While the drawing may not be mathematically accurate, it effectively conveys the underlying concept. The first row depicts the true values of the components, where all the information can be captured by the low-dimensional terms. The second row demonstrates how these lower-order effects can be absorbed by the higher-dimensional terms.

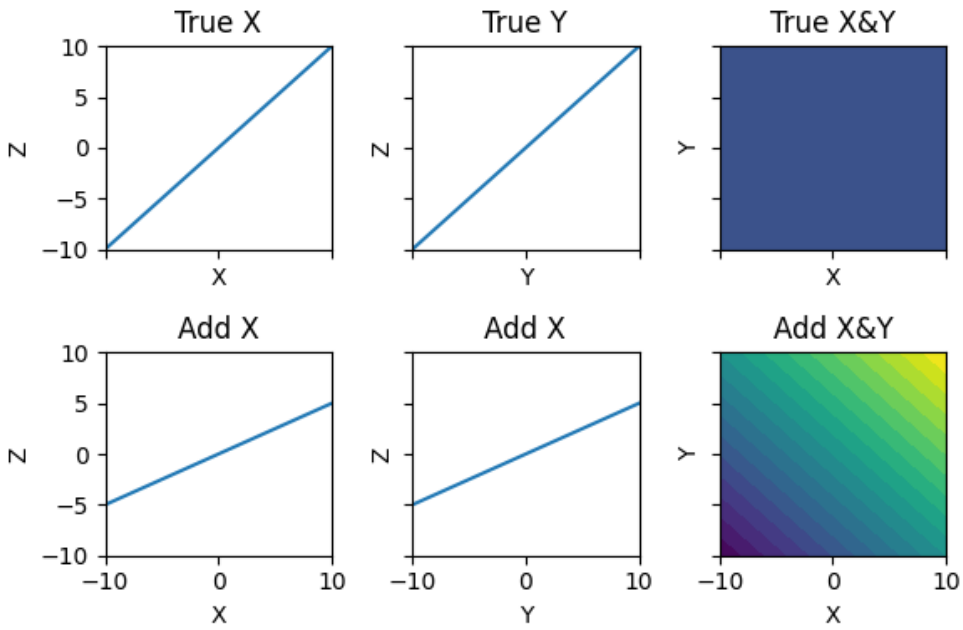

Figure 3 – A graphic demonstrating the concepts discussed in section 2.3. The top row shows the true values of the components of a function, with no relation present in the component representing the interaction between x and y. The bottom row provides a provisional example of the identifiability issue by showing how the pattern that should only be present in the x component and the y component has been captured by the x and y component.

A solution was proposed in the revisited paper to address this issue by orthogonalizing the components of the model. This is achieved by incorporating an orthogonality constraint into the base kernels. The orthogonality constraint is a specific constraint applied to the base kernel that enforces the integral of the kernel's function to be zero with respect to the input measure. By using this constraint, the mixing of orders is prevented which minimizes the presence of higher-order terms unless they are essential. As a result, the model becomes more understandable, providing clearer insights into its underlying mechanisms.

Incorporating this mechanism into the Additive kernel relies on two important results. The first is that the orthogonality constraint is preserved for higher orders through multiplication of constrained lower order terms. This eliminates the need for a different process in forming the higher order components compared to the standard Additive kernel. The second is that the implementation of the orthogonality constraint into the squared exponential kernel has an analytical solution. Therefore, the Additive kernel can be constructed in exactly the same fashion, with only a switching of the base kernels. The construction of the kernel in this manner is referred to as the Orthogonal Additive Kernel (OAK).

Another issue posited with the original model is that of extracting the importance of each component. The original proposed method of determining importance relied on examination of the hyperparameters. The revisited paper challenges the efficacy of this methodology suggesting an alternative approach that yields superior results. This alternative approach is the use of Sobol indices.

Sobol indices are a statistical measure used to assess the contribution of individual input features to the overall variability of an output. They decompose the total variability into components attributed to each input variable and their interactions. Sobol indices are valuable in analysis to identify the most influential components and consequently provide interpretability to the model.

The paper showed that calculation of the Sobol index for the posterior mean function of a component was tractable (in the case of using SE base kernels). This allows for direct computation of these values. The indices are normalised after being calculated to allow for easier understanding.

3. Software Development

Software development encompasses an array of interrelated components that are integral to the successful creation of software tools. Software architecture forms the structural foundation upon which software systems are built. It entails the high-level design decisions that determine the overall organization, communication, and interaction between various software components. Development processes employed in software development play a crucial role in achieving successful outcomes. These processes encompass a range of methodologies and frameworks that guide the entire software development lifecycle. The details that underlie software development is essential for comprehending the inner workings of software systems. This encompasses various aspects, such as programming languages, algorithms, data structures, and libraries, which are the building blocks used to create functional software solutions. By utilising each of these concepts an effective and reliable software tool can be developed.

3.1 Software Architecture

The architecture of the software tool is designed with a focus on efficiency, user-friendliness, and interactivity. The tool is implemented using the Python programming language, leveraging its versatility and extensive ecosystem of libraries. The software tool employs a modular architecture that comprises three main components: the frontend, the backend repository, and the interfaces. A graphical representation of the architecture can be seen in the figure below.

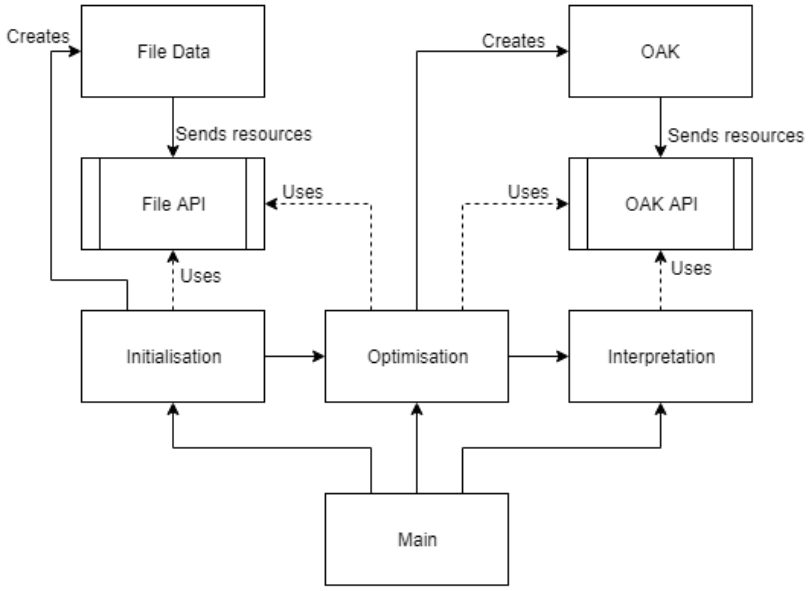

Figure 4 – A diagram representing the architecture of the software tool. The regular boxes represent standard code procedures. The boxes with double lines on their edges represent code that will be cached.

The frontend of the tool is built using Streamlit [3], a popular open-source Python framework for creating interactive web applications. Streamlit provides a straightforward and intuitive way to design and develop the user interface, making it easy for users to interact with the tool and explore its functionalities.

The backend of the software tool is based on a previously developed repository [4] from the revisited paper [2], which serves as the foundation for the tool's data processing and analysis capabilities. This repository encompasses a set of functions and modules that handle data manipulation, model training, and evaluation. Leveraging the existing repository allows for reusability and integration of well-tested and optimized components, saving development time and ensuring a robust backend infrastructure.

For communication between frontend and backend, there are multiple interfaces that are used throughout the software. These interfaces are used to allow for efficient and robust passing of data and resources. Most of the interface calls contain the ability to cache the relevant data/resources, thus allowing the interface to operate smoothly.

3.2 Development Processes

The development process involved the utilization of iterative software development methodologies. Each iteration was gradually improved until the desired outcome was fully achieved. Throughout the development of each iteration, design methodologies were employed to ensure efficient and effective coding practices were implemented.

3.2.1 Development Iterations

During the development process of the software tool, a methodical and iterative approach was employed to improve its functionality and user experience. Throughout this process, three significant iterations were carried out, each dedicated to addressing specific challenges and enhancing the tool's performance. These iterative stages played a crucial role as milestones, enabling identification for areas for improvement, and allowing the refining of the tool's

features and functionality. The table below compares the different versions of the software over the course of development.

Table 1 – table showing the focus of each iterative version of the software tool.

Software Version	1	2	3	4
Speed	✗	✓	✗	✓
Functionality	✗	✗	✓	✓

The first form of the tool did not include a user interface and was intended to solidify understanding of the background material and theory that would be needed in the later versions. In this initial iteration, the implementation was built from the scratch and only used basic modules such as NumPy and SciPy. By restricting the code to only basic components, a comprehensive grasp of the underlying principles was gained.

The second version implemented extra modules for both the frontend and backend of the software, expanding its capabilities and enhancing its performance. With the inclusion of GPflow [5] and TensorFlow [6] in the backend, the computational processes were significantly optimized, resulting in a much faster and simpler implementation of the theory. However, despite these advancements, the second iteration encountered a notable limitation in terms of richness of features and customization options. While the tool excelled in computational efficiency, it fell short in providing users with a diverse range of functionalities and the ability to tailor the tool to their specific needs.

Recognizing this drawback, the third iteration aimed to address the issue of limited features and customization options by leveraging the repository developed by the team who wrote the second paper, harnessing its resources and insights. By integrating this repository, a more straightforward verification of results was enabled, ensuring alignment with the established research. However, as these additional features were implemented, the length of time taken for the code to run started to become a complication. This drawback became particularly evident when dealing with larger datasets, causing usability problems. The tool's performance suffered as the number of extra features increased, impeding its effectiveness in handling substantial data sets.

The final iteration of the software tool aimed to maintain all of the extra features while also being fast enough that usability was not dependent on the time taken to run the code. Caching techniques were strategically employed to optimize the tool's performance and enhance its user-friendliness. Leveraging caching mechanisms enabled the tool to store and retrieve frequently accessed data, models and plots, reducing processing time and improving overall efficiency. This final iteration marked a significant milestone in the development process, culminating in a software tool that balanced performance and functionality.

3.2.2 Development Methodology

The development of the final iteration of the software followed a principled approach to achieve the software goals. A continuous improvement methodology was employed, ensuring robust ongoing enhancements and refinements to the software.

GitHub Action workflows were leveraged to automate code checks, offering a seamless process for validating any code alterations. This workflow encompassed both linting and testing, which played vital roles in optimizing code quality and reliability.

Linting is an essential practice in code development that offers several benefits to improve code quality and efficiency. By utilizing linting tools, errors and inefficiencies can be detected and corrected early on, enforce coding standards, and promoting best practices. Using an industry standard for linting ensures code uniformity, facilitating better readability, comprehension, and maintainability for other developers.

The testing was implemented using a behaviour-driven development (BDD) approach. This methodology involves first identifying the expected behaviours of the specific code section. Subsequently, tests are written before writing the actual code, ensuring that the expected behaviours are met. This approach is advantageous because it enables comprehensive test coverage, ensuring that the software functions as intended. By prioritizing tests early in the development process, a more robust and reliable software solution was able to be developed. Specific details on testing are included in a later section (3.3.6).

3.3 Software Details

The particular way in which Streamlit operates as a user interface presents an interesting subset of possibilities that brings a number of benefits and challenges. Whenever an interaction occurs in the interface webpage, the entirety of the code in ran from the start, with any alterations made in the interaction included in the subsequent running. This results in any unoptimized code being particularly noticeable. It also presents issues for the style of classes as instances are not by default preserved between runs.

Attempting to integrate this style of program with the repository from the revisited paper raised a number of difficulties. The repository uses class instances for the model, and with some significant amount of computation being done in the setting up of these classes, attempting to use this in a conventional manner was not an option. Careful consideration was needed to successfully combine the benefits of interactive Streamlit interface with the repository's class-based models.

3.3.1 Interfaces

An interface in software refers to a formal contract or specification that defines a set of methods, properties, and behaviours that a class or component must adhere to in order to interact with other software entities.

The role of interfaces within the application is crucial as they serve to simplify and optimize the software. Each interface is implemented as a class with multiple "get" and "set" calls, enabling efficient interaction with the underlying functionality. The specific structure of these calls is outlined using the example code below.

```python
class ExampleInterface:

    @staticmethod
    @st.cache_data
    def get_example(_example_arg: Any = None) -> Any:
        if _example_arg is None:
            raise NotCachedError
        return _example_arg

    @staticmethod
    def set_example(example_arg: Any) -> None:
        # perform any necessary computing on example_arg
        _ = ExampleInterface.get_example(example_arg)


def function_for_setting():
    # perform any necessary computing on example_arg
    ExampleInterface.set_example(example_arg)


def function_for_getting():
    try:
        return_value = ExampleInterface.get_example()
    except NotCachedError:
        pass  # Take necessary action
    # do next step
```

The structure of the "get" and "set" calls is influenced by the particular caching mechanism employed by Streamlit [3]. To inform Streamlit that a particular function is being cached, the "@st.cache_data" decorator is utilized. Modifying the arguments passed to the cached function results in each call being treated as a separate entity. However, for most cases in this application this behaviour is not desirable, only being used in certain areas such as varying plotting.

To address this issue, an underscore is added to the argument name, effectively preventing it from being hashed. Consequently, the function can be invoked later in the code without the need to provide any arguments explicitly. This approach allows for the discarding of data that would have otherwise been used as arguments, thereby reducing code-bloat and improving efficiency.

Additionally, the presence of static methods within the class signifies that it serves as a namespace rather than being instantiated as an object. This means that the class is utilized solely to group related functions and variables together, without the need for creating instances of the class.

By employing interfaces structured in this manner, the application achieves enhanced simplicity, optimization, and improved code organization. These interfaces provide a streamlined interface for accessing the underlying functionality while ensuring efficient caching and reducing unnecessary code duplication.

3.3.2 Initialisation Page

This page of the application serves as the initial interface for users upon launching the software tool. It is intended to allow users to choose a dataset to be used by the model. A flowchart showing the intended behaviour of the software for this page is presented below.

The simple behaviour for the intended process facilitated the development of a streamlined code implementation for this section. By utilizing try/except blocks for the caching functions, the software was able to perform straightforward file checks. Additionally, a modular system was employed to integrate the file selection and details, resulting in a well-organized design that enhanced the overall functionality of the application.

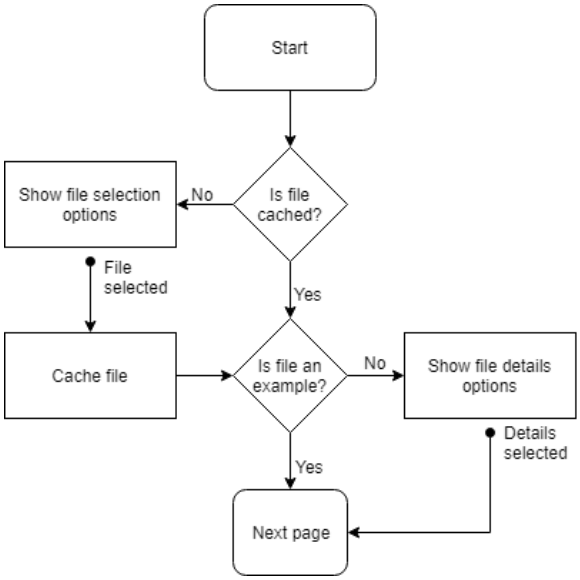

Figure 5 – A flowchart showing the intended behaviour of the initialisation page. The round boxes represent the start and end points of the page. The diamond boxes represent decisions, with the yes and no arrows indicated the consequence of the decision path. The rectangular boxes represent modular section of code. The arrows that begin with dots indicate the procedure that occurs based on a user interaction.

The interface for this page is depicted in the figure below. It incorporates a tab system that presents users with two alternatives for data input. They can either upload their own files or choose from a collection of pre-selected example files provided with the tool.

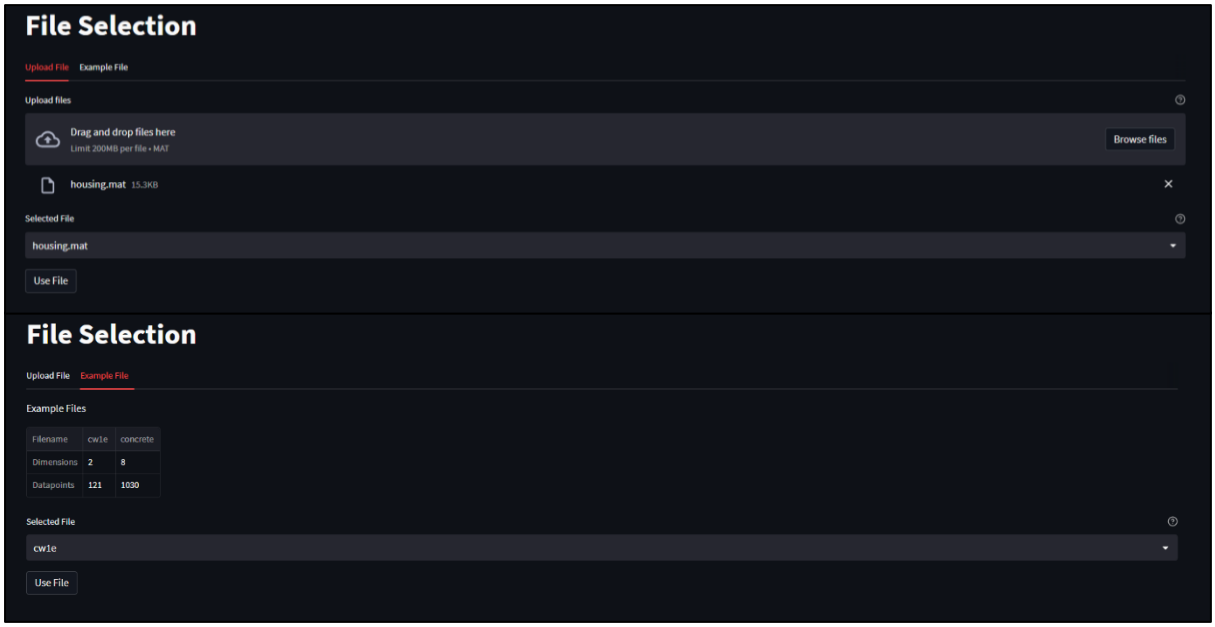

Figure 6 – A snapshot taken of the user interface. The top image shows the tab for uploading files. The bottom image shows the tab for selected a provided example file.

Opting to use an uploaded file leads the user to another page, shown in the image below, where they can interactively view a dataframe of the data and have the opportunity to name the features and output value. This design provides a user-friendly system, with the flexibility to work with their custom datasets or explore the capabilities of the tool using preloaded data.

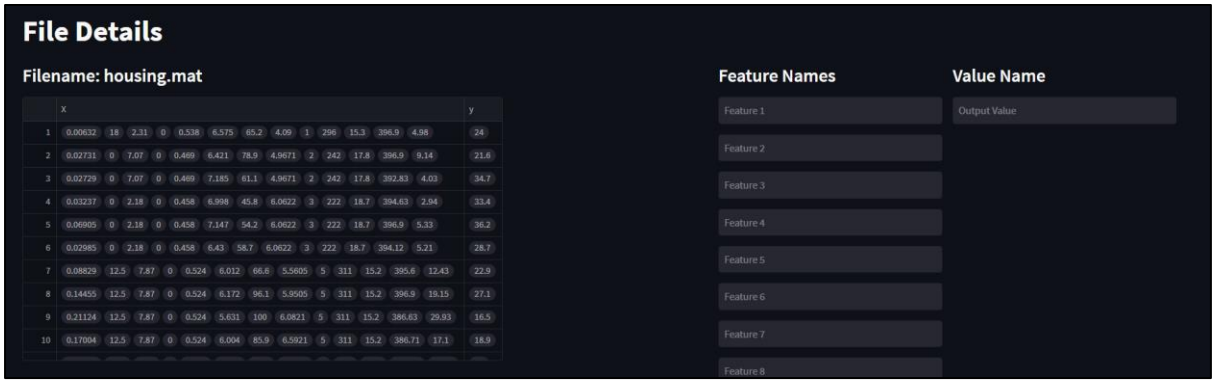

Figure 7 – A snapshot of the user interface showing the page for selecting the details of a file. An interactive dataframe of the dataset is present on the left and textboxes to enter feature name is displayed on the right.

3.3.3 Setup and Optimisation Page

Upon defining the dataset, users are directed to the optimization page, which serves the purpose of fine-tuning the model setup to meet specific requirements. Once the desired options have been chosen, the model is optimized accordingly.

The interface for this page is illustrated in the figure below. It offers a range of options and parameters to customize both the dataset and the model. The dataset options include the ability to select a specific train/test split. On the other hand, the model options encompass parameters such as maximum interaction depth and length scale bounds. Additionally, if the threshold for the number of data points is exceeded, users have the option to select the number of inducing points for a sparse model. By providing this level of configurability, the tool accommodates different user preferences and facilitates experimentation with various modelling approaches.

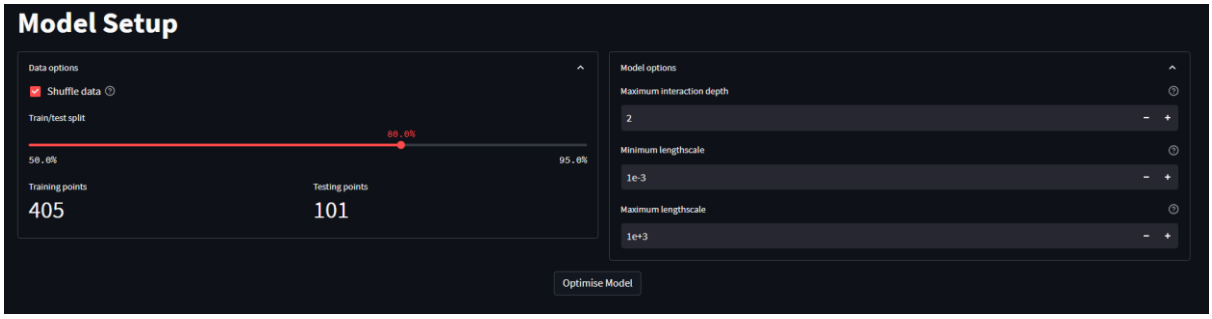

Figure 8 – A snapshot of the user interface showing the optimisation options. Dataset options are present in an expander on the left and model options are present in an expander on the right.

Once the desired options have been selected, the model undergoes setup and optimization. The optimization process incorporates callbacks to relay the progress of optimization to the user. The form of this progress display is depicted in the figure below. It showcases the progress for normalizing the orders of the model and provides real-time updates on the hyperparameter values. It is worth noting that while it would be more desirable to include a real-time plot of the function being minimized instead of solely focusing on the hyperparameters, the current

system for optimization does not support this functionality. Despite this limitation, the implemented system effectively communicates the progress of optimization to the user.

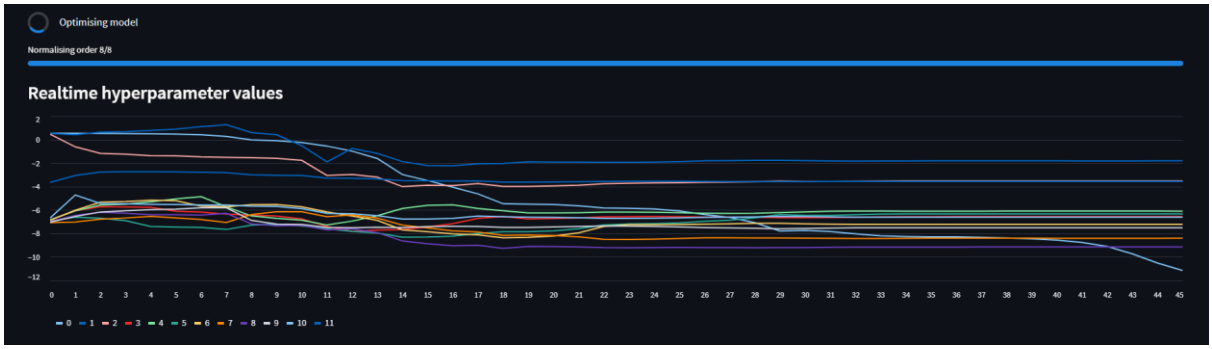

Figure 9 – A snapshot of the user interface showing the progress display during optimisation. At the very top is a spinner that states the current process, below that is a progress bar that indicates which order is being normalised during setup. At the bottom is a graph showing the real-time hyperparameter values.

Upon completion of the optimization process, the optimized model is cached, enabling efficient execution in subsequent runs. Caching the model allows for faster and more streamlined operations, as the optimized version is readily available for future use.

3.3.4 Interpretation Page

Once the model has been optimized, the final page focusing on interpretation is displayed. This page showcases the results obtained from the model through interactive graphs and visualizations. These visual representations provide valuable insights into the relationships between the input features and the output generated by the machine learning model. This facilitates effective interpretation and decision-making from a deeper understanding of the underlying patterns and correlations.

The figure below shows an example of this page. Metrics are given for the performance of the dataset allowing for comparison to other models. The order contributions show which level of interaction is important. The component contribution shows how the performance of the model changes based on how many components are included in the model. This is useful because it shows how computational cost could be traded for accuracy. The last plots are the component plots. The amount generated can be altered by a widget. These plots give insight into exact relations between features.

The figure below demonstrates this page. Performance metrics for the dataset are provided, allowing users to compare the model's performance with that of other models. The order contributions graph illustrates the importance of each level of interaction in the model. The component contribution graph showcases how the model's performance varies based on the number of components included, offering insights into the trade-off between complexity and accuracy. The component plots provide a precise visualisation of the underlying function for each component. The amount of these plots being generated can be adjusted using a widget, enabling a detailed exploration of specific feature relations.

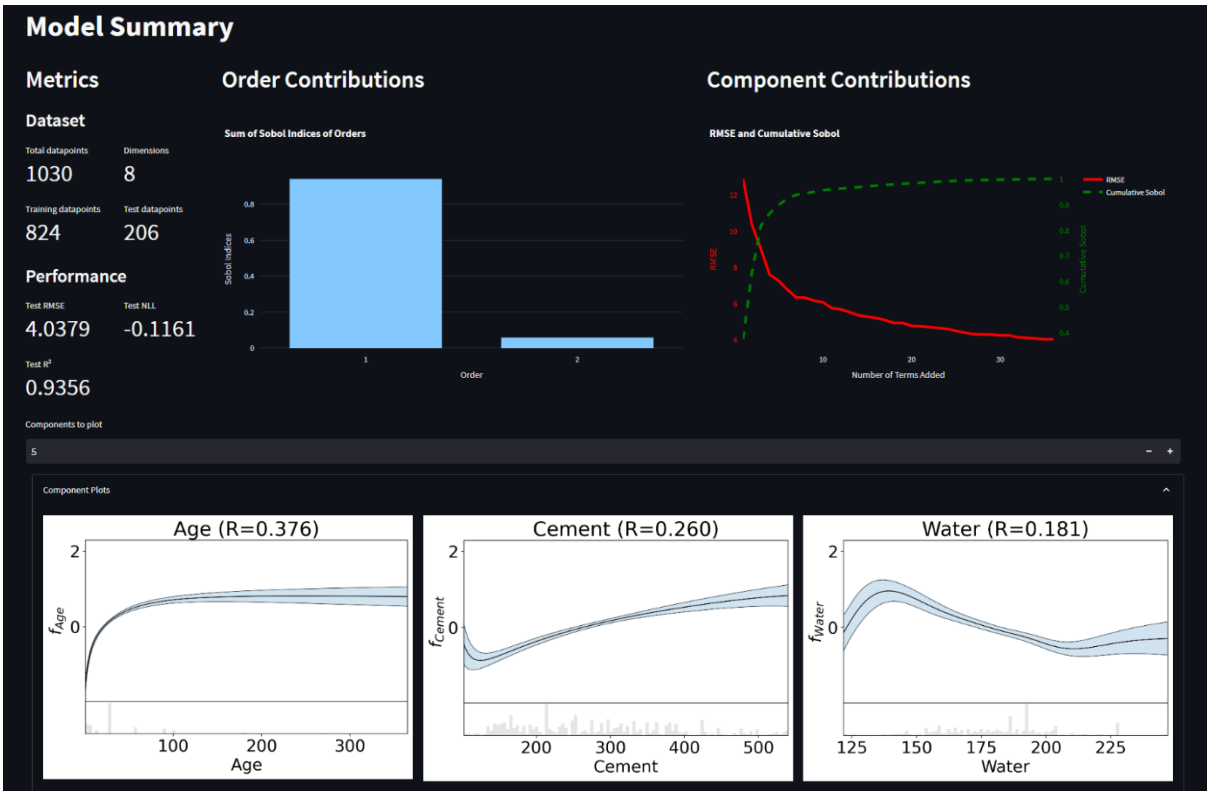

Figure 10 – A snapshot of the user interface showing the final page provided details of the model. Top left shows metrics of the model. Top middle shoes the order contribution plot. Top right shows the component contributions plot. Bottom shows the individual component plots.

The software implementation of this page employed a modular system to optimize the code. Adopting a modular approach benefited both the quality and performance of the page. Additionally, caching mechanism are used to store and retrieve results, ensuring quick access for subsequent runs. This caching feature significantly improved the system's performance by eliminating the need to recompute data that had been previously processed.

The functionality of this page provides a comprehensive and visual representation of the model's results, empowering users to gain valuable insights and make informed decisions based on the model's predictions.

3.3.5 Repository changes

Care was taken to avoid modifying the existing repository unless the alteration would provide significant benefit. When utilizing a pre-existing repository, it is imperative to minimize changes to the existing codebase. This approach ensures compatibility with the software tools built on top of the repository and maintains stability. Extensive modifications can introduce compatibility issues and compromise the functionality of the tools. Altering the codebase without thorough testing can result in bugs and undermine the reliability of the software. Additionally, minimizing changes reduces the maintenance burden associated with future updates and bug fixes. It allows for easier integration of new versions and enhancements from the original repository, promoting upgradability and long-term sustainability. Ultimately, minimizing modifications to the pre-existing repository fosters a robust and reliable software ecosystem that can evolve and adapt to future needs. Only three changes were deemed to be of sufficient value to warrant implementation.

The first change involved the introduction of callback functions to track computation progress. Callback functions for tracking progress are useful because they provide a mechanism to track and communicate the progress of long-running operations or tasks. These functions are typically implemented as parameters in asynchronous or iterative processes, allowing them to be called at specific intervals or checkpoints during the execution. The callbacks were specifically integrated as optional parameters in the model classes and functions. These callbacks are invoked either at the start of loops, or in the case of the optimisation callback, it is instead passed to the SciPy minimize which returns specific results to the callback.

The Sobol indices for a model are stored within a class attribute, however this attribute is not instantiated at initialisation of the class. As a result, the function generating these indices had to be called each time they were required. Generating these indices is a computationally expensive process for high dimensional dataset and performing this multiple times hinders the usability of the software to a great degree. To address this issue, a check was implemented using the "hasattr" built-in function before calling the Sobol function. If the attribute was present, indicating that the indices were already generated, the function was not called again.

Lastly, a modification was made to avoid an error that occurred when attempting to generate component plots with three or more dimensions. The original repository code raised an error in these cases. The alteration involved returning None instead of a figure, allowing for the display of an appropriate message instead of halting the program.

3.3.6 Testing

During the development phase, unit tests were utilized to ensure the correct functionality of the software tool. However, when working with Streamlit, certain tests posed challenges in terms of implementation. This was primarily because many of the interface functions required webpage input to execute their intended functionality. To address this issue, the concept of patching was employed.

Patching a function offers a solution by temporarily substituting external dependencies with mock objects or alternative implementations. This approach facilitates the simulation of specific scenarios and allows for controlled operation of these dependencies. Consequently, it becomes easier to test the desired functionality in isolation.

To aid in the process of mocking the widgets, an existing module called "Streamlit-mock" was utilized. However, this module was not updated to the latest version of Streamlit, which led to certain areas of the code being rendered incompatible. One workaround for this issue involved directly patching features, such as the session state with an empty dictionary. Where solutions like this were not possible, adjustments were made to rewrite specific portions of the code.

Testing the caching functionality posed significant challenges due to the introduction of non-determinism into the code. When caching is involved, the output of a function or code block becomes dependent on the cache state, making it difficult to obtain predictable test results. Moreover, caching can create dependencies between test cases, impeding the creation of isolated and independent tests.

To address these issues, the caching decorator was patched with a blank mock. This approach proved viable for the majority of tests, considering the carefully crafted form of the cached interface functions. In cases where blank mocks were not suitable, non-blank mocks were utilized to simulate the interface calls. This approach allowed for the management of caching

complexities and facilitated the implementation of more robust and reliable testing methodologies.

4. Results and discussion

The theoretical results encompass an evaluation of essential aspects derived from the paper. As the paper this software tool is based on provides numerous results, focus will be placed only on highlighting key findings to avoid unnecessary redundancy. By carefully examining select components, the validity and accuracy of the findings can be ensured. This approach allows consolidation of the most significant findings, thereby providing a concise yet comprehensive overview of the theoretical implications.

In addition to the theoretical outcomes, software timing results are included. These serve as a crucial indicator of improvement. These measurements demonstrate the effectiveness of the implemented software and its impact on performance. By showcasing the advancements achieved through the software, the tangible benefits resulting from its implementation can be highlighted.

4.1 Theoretical Results

The primary objective of the theoretical analysis was to validate the accuracy of the paper's findings and ascertain their reproducibility through the utilization of software. To accomplish this, the study concentrated on two specific areas. Firstly, the quality of a predictions made by the model on test data was examined. Secondly, component plots were employed to visually evaluate the accuracy of the model.

In the context of Gaussian Processes (GPs), the Negative Log-Likelihood (NLL) is a commonly used metric to assess the performance of the model. The NLL measures the quality of predictions made by the GP model by quantifying how well it captures the data.

The NLL is derived from the likelihood function, which describes the probability of observing the given data given the model's parameters. In the case of GPs, the likelihood function considers the distribution of the target variable conditioned on the observed inputs and the model's hyperparameters.

When evaluating a GP model on a test fold, the NLL is typically computed by comparing the model's predicted mean and variance (or covariance matrix) to the true target values. The NLL quantifies the discrepancy between the model's predictions and the actual observations, with lower values indicating better performance.

The importance of the NLL on a test fold lies in its ability to provide an objective measure of how well the GP model generalizes to unseen data. By evaluating the model's performance on a test fold, which contains data that it hasn't been trained on, we can gauge its ability to make accurate predictions in real-world scenarios.

The following table presents the negative log-likelihoods (NLLs) of various models evaluated on the "servo.mat" dataset. This dataset consists of 4 dimensions and 167 datapoints, with an 80/20 train/test split. The models assessed include the squared exponential Automatic Relevance Determination (SE-ARD), an extension of the SE-GP for higher dimensions, the

Additive kernel as described in the original paper, a "from-scratch" implementation of the OAK, and the OAK utilized in the final software.

Based on the results displayed in the table, it is evident that both OAK models outperform the SE-ARD and Additive kernel, as they achieve lower NLL values. Notably, the OAK model employed in the final software is expected to exhibit an even lower NLL due to the implementation of additional optimization tactics, such as placing priors on hyperparameters and normalization. These techniques further enhance the model's performance and overall accuracy.

Table 2 – Table showing the test NLL for different models on the same dataset

Model	SE-ARD	Additive	OAK (Custom implementation)	OAK (Final Software)
Test NLL	15.72	15.66	13.23	12.14

Examination of component plots is a crucial aspect of evaluation as they are what lead to the interpretability of the model. By utilizing a dataset sampled on a known function, effective analysis of the component plots can occur, as there is a clear understanding of what should be represented in each plot. The revisited paper uses the following function for the dataset: $f(x_1, x_2) = x_1^2 - 2x_2 + \cos(3x_1)\sin(5x_2)$. This function is used due to its utilization of all three potential components, thereby facilitating the clear visualization of their distinctiveness and ensuring that each component maintains its expected appearance.

The figure below, taken from the revisited paper [2], illustrates that the components of the OAK conform to the true values of the function in a much greater degree than the components from Additive kernel.

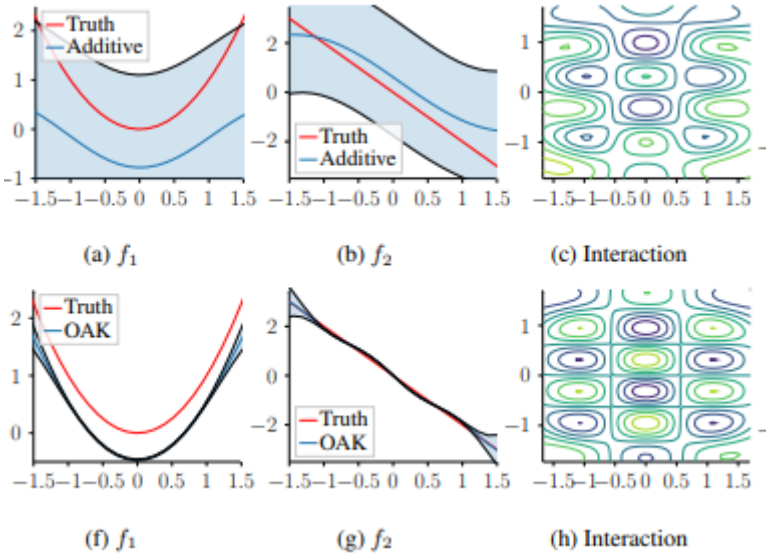

Figure 11 – A figure taken from the revisited paper [2] showing the component plots for data sampled on a known function.

The second figure illustrates the component plots produced by the final software. A comparison between the generated plots and the expected ones demonstrates that the final software successfully reproduces the anticipated plots, thereby showcasing its functionality and accuracy. It also serves to validate the results obtained in the paper. It should be noted

that different axes are employed for the software plots, resulting in apparent differences in shape. However, upon closer examination, it becomes evident that the accurate reproduction of the expected plots has indeed taken place.

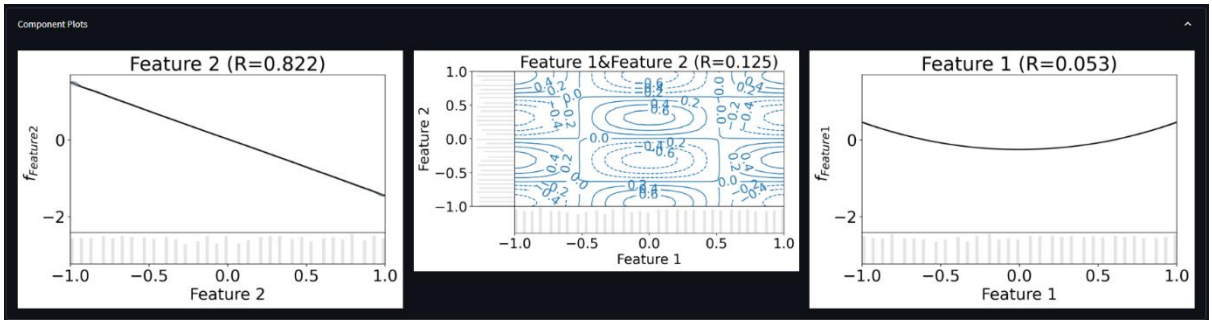

Figure 12 – Figures generate by the final software for the same function as fig. 11.

4.2 Software Results

In the pursuit of delivering a superior user experience, optimizing code to achieve high performance and fast execution is of paramount importance. Fast code not only enhances the efficiency of computational processes but also contributes significantly to the overall user satisfaction and engagement.

The full runtimes of different code implementations were measured and compared in order to assess their efficiency in processing a specific dataset. The dataset used for this analysis was "concrete.mat," which consists of 1030 datapoints and 8 dimensions. It is worth noting that the kernel employed in the analysis was limited to an interaction depth of 3. This was to simulate a typical use case of a moderately complex dataset.

By measuring the runtime timings, we gain insights into the computational efficiency and speed of the different code implementations. Faster runtimes signify a reduced processing time and a quicker execution of tasks. The full runtime measurement involved: loading the dataset, setting up the model, optimising the model, finding Sobol indices and generating component plots.

The following table illustrates the runtimes for four distinct software components: the example Jupyter notebook provided in the repository from the revisited paper, the initial coding attempt, the optimised software utilising custom OAK instead of the, and the final software built on the repository OAK.

The timings clearly demonstrate that the initial coding attempt would be impractically slow to use. This is attributed to the convoluted manner in which all computations were performed, relying on loops instead of vectorization.

In contrast, the revised software emerged as the fastest, as it implemented all functionalities in an optimized manner focused on speed. This outcome was expected since it heavily, sacrificed certain functionalities to excel in speed.

Although the final software is only approximately half as fast as the speed-optimized code, it is significantly less concerning than the initial code, given that the durations are around the one-minute mark. Moreover, the final software outperforms the Jupyter notebook from the repository, completing the tasks in just under 60% of the time. This improvement can be

attributed to the addition of "hasattr" checks, which eliminate redundant runs of expensive functions (further details in section 3.3.5).

It is important to note that the reported runtime also includes user interactions, such as clicking buttons and selecting relevant options, whereas the notebook only required running a modified notebook. Including the time taken to modify the notebook to accommodate a different dataset would significantly increase the overall time, particularly for individuals who are not familiar with the process. The fact that the final software outperforms the notebook further contributes to the goal of achieving a user-friendly experience.

Table 3 – A table showing results from timing the full runtime of varying software implementations.

Software	Repository Example	Initial Software	Revised Software	Final Software
Execution time	00:02:24	01:20:14	00:00:53	00:01:31

5. Conclusion

The development and refinement of the software tool for this project have yielded a valuable resource for conducting efficient and customizable data analysis using the Orthogonal Additive Kernel. Through multiple iterations and a principled approach, the tool has evolved to result in a software solution that offers both performance and functionality.

A significant challenge encountered during the development process was striking a balance between incorporating additional features and maintaining acceptable runtime performance. As the tool expanded to include more capabilities, the execution time became a notable concern, particularly when handling larger datasets. Nevertheless, the final iteration of the software effectively addressed this issue by implementing caching techniques. By strategically storing and retrieving frequently accessed data, models, and plots, the tool achieved enhanced efficiency and user-friendliness while retaining the desired features.

The software tool's interface was purposefully designed to provide a user-friendly experience and flexibility for data input. Users can either upload their own files or select from a curated collection of example files. Interactive features enable users to assign names to features and customise options for dataset and model configuration, empowering users to experiment with different approaches. The interpretation page provides interactive graphs and visualizations, facilitating effective analysis and informed decision-making.

From a theoretical standpoint, the software tool validates the accuracy and reproducibility of the key findings presented in the original research paper. Negative Log-Likelihood (NLL) is used as a metric to evaluate the quality of predictions made by the Gaussian Process (GP) model on test data. The NLL values obtained for the implemented models consistently demonstrate superior performance of the OAK models compared to alternative kernel options. Component plots are utilized to visually assess the accuracy of the model, and the software successfully reproduces the expected plots, confirming the validity of the results.

Efficiency and runtime performance are critical considerations for the software tool. Comparative measurements of different code implementations establish the optimized nature of the final software, resulting in significantly improved processing speed compared to earlier

iterations. By optimizing computations, leveraging caching techniques, and streamlining code structure, the final software achieves notable advancements in processing efficiency, making it highly practical for real-world applications.

Although the software tool has reached a significant milestone in its development, there remain several avenues for improvement that can enhance its functionality and user experience. These potential areas of future work include the following:

Enhanced Data Loading Page: The software tool currently supports a range of file formats for data loading. However, expanding this support to include additional file formats would provide users with greater flexibility and convenience. By incorporating robust libraries or implementing custom parsers, the tool could accommodate a wider range of data formats, ensuring seamless data integration and analysis.

Additional Optimization Options: The software tool already offers various optimization options for dataset and model configuration. However, further implementation of advanced dataset and optimization options could provide users with more options to fine-tune their analysis. Incorporating these extra selections would enable users to extract optimal model configurations with improved efficiency and accuracy.

Expanded Plotting Options: While the software tool currently generates component plots using the oak repository code, there is potential for further development to enable the generation of interactive plots. By leveraging interactive plotting libraries such as Plotly or Bokeh, users would gain the ability to explore and manipulate the plots directly within the tool's interface. This interactive visualization capability would enhance the user's understanding of the data and facilitate deeper insights.

Rewriting the OAK Repository: A significant undertaking for future work would involve rewriting the borrowed repository to enhance its performance, maintainability, and integration within the software tool. Although this is a complex and time-consuming task, rewriting the codebase would enable better alignment with the software's architecture and design principles. This, in turn, would facilitate seamless updates, bug fixes, and feature additions, while ensuring the long-term sustainability and scalability of the software.

These suggested areas for future work represent valuable opportunities to refine and expand the capabilities of the software tool.

In conclusion, the development of the software tool presented in this report represents a successful evolution towards an efficient and adaptable solution for Gaussian Process-based data analysis. The user-friendly interface, extensive customization options, and accurate results allow the software tool to exist as a valuable resource for researchers and practitioners engaged in Gaussian Process analysis and data exploration.

References

[1] - David Duvenaud, Hannes Nickisch and Carl Edward Rasmussen. *Additive Gaussian Processes*. 2011

[2] – Xiaoyu Lu, Alexis Boukouvalas and James Hensman. *Additive Gaussian Processes Revisited*. 2022

[3] – Streamlit Documentation, https://docs.streamlit.io

[4] – Repository from [2], https://github.com/amzn/orthogonal-additive-gaussian-processes

[5] – GPflow Documentation, https://gpflow.github.io/GPflow/develop/index.html

[6] – TensorFlow Documentation, https://www.tensorflow.org/api_docs

Appendix

A) Gaussian Processes

A.1 Gaussian Processes Optimisation

The log marginal likelihood is used for the optimisation of the hyperparameters by being the function to minimise. It is given by the following equation.

$$-\frac{1}{2}\mathbf{y}^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{y}-\frac{1}{2}\log|\mathbf{K}+\sigma_n^2\mathbf{I}|$$

The first term of this equation is considered the data fit term and scores how well the model fits the data. The second term is the complexity penalty that prevents the model from overfitting. While there is also third term, this remains constant and is therefore not necessary for minimisation.

A.2 Gaussian Processes Prediction

The predictions are the distributions generated by the model for a given set of inputs. The predictive distribution at any given point x_* is given by the following normal distribution.

$$\mathrm{p}(y_*|x_*,\mathbf{x},\mathbf{y}) \sim N\begin{pmatrix}\mathbf{k}(x_*,\mathbf{x})^{\mathbf{T}}[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{y}, \\ \mathbf{k}(x_*,\mathbf{x})+\sigma_n^2-\mathbf{k}(x_*,\mathbf{x})^{\mathbf{T}}[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{k}(\mathbf{x},x_*)\end{pmatrix}$$

A.3 Gaussian Processes Computation Improvement

Matrix algebra can be applied to the log marginal likelihood in A.1 to improve its computational efficiency. As the covariance matrix (**K**) is always symmetric positive definite (after a very small identity matrix is added**)**, a Cholesky decomposition can be performed. Using the Cholesky decomposition of the matrix allows for much faster computation of both terms of this equation.

The first term requires an inverse of the matrix, and this can be done by solving the decomposition triangularly for an identity matrix. This computation is an order of magnitude faster than a standard inverse (44 seconds vs 1.4 seconds).

The second term requires the determinant of the matrix, this can be computed effectively by using the fact that the determinant of a triangular matrix is the product of the diagonal. With the logarithm, the product becomes a sum, and the squared term becomes a multiplication by 2 which in turn cancels out the half.

This results in the much more efficiently computed equation below.

$$\mathbf{K}+\sigma_n^2\mathbf{I}=\mathbf{L}\mathbf{L}^{\mathbf{T}}$$

$$-\frac{1}{2}\mathbf{y}^T(\mathbf{L}^{-1})^{\mathbf{T}}\mathbf{L}^{-1}\mathbf{y}-\sum\log diag(\mathbf{L})$$

B) Additive Kernels

B.1 Additive Kernel Structure

The following formula from the original paper [1] shows that the additive kernel is broken down into different order of kernels of ascending dimensionality.

$$k_{add_1}(\mathbf{x},\mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \quad (1)$$

$$k_{add_2}(\mathbf{x},\mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \quad (2)$$

$$k_{add_n}(\mathbf{x},\mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \prod_{d=1}^{N} k_{i_d}(x_{i_d}, x_{i_d}') \quad (3)$$

Each order of the additive kernel is composed of all possible combinations of n base kernels where n is the dimension. The entire additive kernel is given by the sum of each of these terms for all dimensions, weighted by an additional hyperparameter.

For a 3-dimensional dataset the orders are:

1st order: $\sigma_1{}^2\left(k_1(x_1,\, x_1') + k_2(x_2,\, x_2') + k_3(x_3,\, x_3')\right)$

2nd order: $\sigma_2{}^2\left(\left(k_1(x_1,\, x_1') * k_2(x_2,\, x_2')\right) + \left(\left(k_1(x_1,\, x_1') * k_3(x_3,\, x_3')\right)\right) + \left(k_2(x_2,\, x_2') * k_3(x_3,\, x_3')\right)\right)$

3rd order: $\sigma_3{}^2\left(k_1(x_1,\, x_1') * k_2(x_1,\, x_1') * k_3(x_1,\, x_1')\right)$

B.2 Additive Kernel Computational efficiency

The original paper [1] shows the Newton-Girard formulae for computing each order of the additive kernel.

$e_0 \triangleq 1$ | $s_k(z_1, z_2, \ldots, z_D) = \sum_{i=1}^{D} z_i^k$

$$k_{add_n}(\mathbf{x},\mathbf{x}') = e_n(z_1,\ldots,z_D) = \frac{1}{n}\sum_{k=1}^{n}(-1)^{(k-1)} e_{n-k}(z_1,\ldots,z_D) s_k(z_1,\ldots,z_D) \quad (6)$$

An example is shown below for the first 2 orders of a 4-dimensional dataset.

$$\text{base kernels} = a, b, c, d$$

Combinations $O(2^D)$:

$$e_1 = a + b + c + d$$
$$e_2 = ab + ac + ad + bc + bd + cd$$

The combinations require direct computing of each term in the order

Newton-Girard $O(D^2)$:

$$e_0 = (1,1,1,1)\ s_1 = (a,b,c,d)\ s_2 = (a^2, b^2, c^2, d^2)$$
$$e_1 = \frac{1}{1}(-1)^0(1,1,1,1)\odot(a,b,c,d) = (a+b+c+d)$$

$$e_2 = \frac{1}{2}\begin{pmatrix} (-1)^0(a,b,c,d)\odot(a,b,c,d) \\ +(-1)^1(1,1,1,1)\odot(a^2,b^2,c^2,d^2) \end{pmatrix}$$

$$e_2 = \frac{1}{2}\begin{pmatrix} a^2+2ab+2ac+2ad+b^2+2bc+2bd+c^2+2cd+d^2 \\ -a^2+b^2+c^2+d^2 \end{pmatrix}$$

$$e_2 = ab+ac+ad+bc+bd+cd$$

C) Orthogonal Additive Kernel

C.1 Orthogonality Constraint

The following formula from the revisited paper [2] show both the definition of the orthogonality constraint and the closed form solution of the orthogonality constraint applied to a SE-kernel.

$$S_i := \int f_i(x_i)p_i(x_i)dx_i = 0,$$

$$\tilde{k}_i(x_i,x_i') = k_i(x_i,x_i') - \mathbb{E}[S_i f_i(x_i)]\mathbb{E}[S_i^2]^{-1}\mathbb{E}[S_i f_i(x_i')],$$

$$\mathbb{E}[S_i f_i(\cdot)] = \int p_i(x_i)k_i(x_i,\cdot)dx_i,$$

$$\mathbb{E}[S_i^2] = \int\int p_i(x_i)p_i(x_i')k_i(x_i,x_i')dx_i dx_i'. \qquad (8)$$

$$\tilde{k}(x,x') := \exp\left(-\frac{(x-x')^2}{2l^2}\right) - \frac{l\sqrt{l^2+2\delta^2}}{l^2+\delta^2}\times \exp\left(-\frac{((x-\mu)^2+(x'-\mu)^2)}{2(l^2+\delta^2)}\right). \qquad (10)$$

C.2 Sobol Indices

This is covered in detail in section 4 and appendix G of the revisited paper [2]

D Project Code

This can be found at https://github.com/lp551/IIB-Project

E Risk Assessment Retrospective

The risk assessment submitted to the Safety Office at the start of the Michaelmas term, outlined one risk which was computer use. This risk posed no consequence over the duration of this progress. The risk is still a valid risk so would be included again if the project was started over. No other risks were noted over the course of the project.