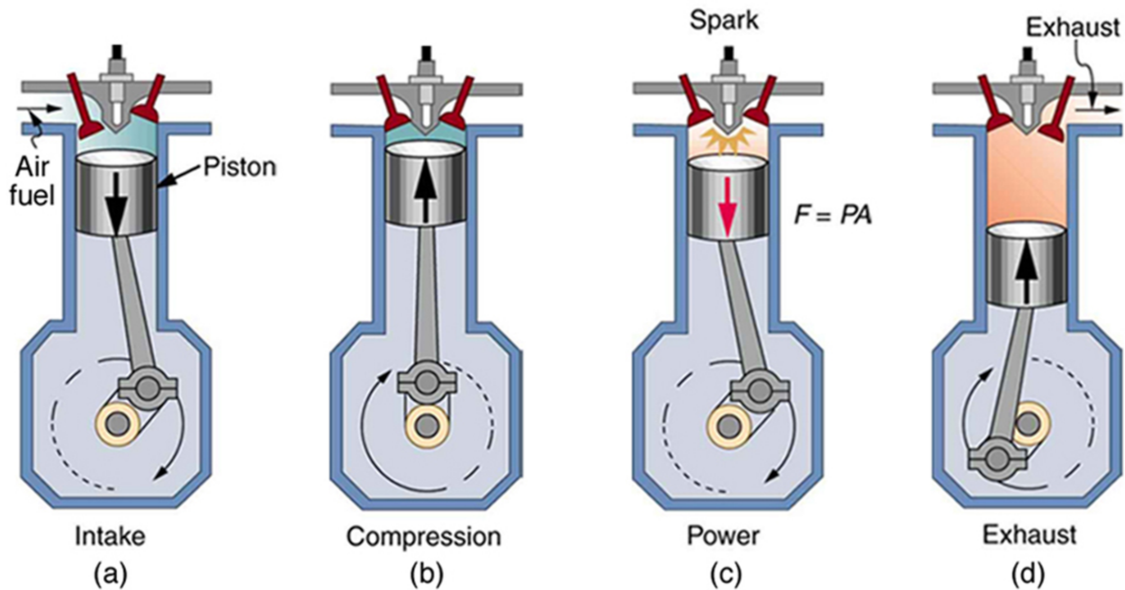
CHALMERS
UNIVERSITY OF TECHNOLOGY



Multidimensional Data-Driven Modelling of Engine Test Cell Data

Using Gaussian Process Regression and Neural Networks to Model Volumetric Efficiency of Four-Stroke Internal Combustion Engines

Master's thesis in Engineering Mathematics and Computational Science

HELENA ANDERSSON

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Multidimensional Data-Driven Modelling of Engine Test Cell Data

Using Gaussian Process Regression and Neural Networks to Model Volumetric Efficiency of Four-Stroke Internal Combustion Engines

HELENA ANDERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2021

Multidimensional Data-Driven Modelling of Engine Test Cell Data
Using Gaussian Process Regression and Neural Networks to Model Volumetric Efficiency of Four-Stroke Internal Combustion Engines
HELENA ANDERSSON

© HELENA ANDERSSON, 2021.

Supervisor: Per Andersson-Hedberg, T-Engineering
Supervisor: Anton Johansson, Department of Mathematical Sciences
Examiner: Serik Sagitov, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Schematic description of the four strokes in a four-stroke internal combustion gasoline engine. See Figure 2.2.

Typeset in LaTeX
Gothenburg, Sweden 2021

Multidimensional Data-Driven Modelling of Engine Test Cell Data
Using Gaussian Process Regression and Neural Networks to Model Volumetric Efficiency of Four-Stroke Internal Combustion Engines
HELENA ANDERSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In the journey towards a more sustainable vehicle fleet, requirements for lower emissions and improved energy efficiency in gasoline engines lead to more components being added to the internal combustion engines. This adds to the degrees of freedom when trying to model air flow in the engine using volumetric efficiency. This paper presents a way of modelling volumetric efficiency from engine test cell data provided by T-Engineering – a company that designs and develops control systems for vehicles. The model uses Gaussian process regression (GPR) for inter- and extrapolation, including noise reduction of the measurement data. Furthermore, a local interpretable model-agnostic explainer (LIME) is used to find regions of uncertainty by explaining what features contribute to increasing the variance of the GPR predictions. In addition, a neural network model is implemented in order to improve the prediction runtime, with the purpose of enabling real-time predictions in the control systems.
The model(s) were found to give a more physically accurate description of volumetric efficiency than the one currently used at T-Engineering. The runtime for making predictions for 50 data points with the neural network was ~ 0.14 ms on an AMD Ryzen 7 PRO 4750U with Radeon Graphics 1.70 GHz and 32.0 GB RAM. It remains to investigate what the runtime on a limited CPU in the control systems will be.

Keywords: Gaussian process regression, LIME, neural networks, volumetric efficiency, test-cell data, multidimensional modelling

Acknowledgements

I would like to thank all the people who, in one way or another, have helped and supported me throughout this thesis project.
A special thanks to my supervisors for their knowledge and support when I embarked on the journey of learning about the new subjects needed to pursue this thesis. Thank you, Per Anderson-Hedberg, for presenting the idea to this master's thesis and for entrusting the project to me. Thank you, Anton Johansson, for joining as supervisor and for teaching me about machine learning. This thesis would not have been possible without the rewarding discussions I have had with the two of you.
A warm thank you also to the people at T-Engineering for introducing me to your company, and for your curiosity in my thesis work. Your questions and inputs about the project have helped me learn even more.
Last but not least, thank you to my "colleagues" at the master's thesis office. Completing a master's thesis on your own through remote work can be cumbersome, but having you to share my working days with has turned it into a great experience.

Helena Andersson, Gothenburg, September 2021

Contents

List of Figures **xi**

List of Tables **xvii**

List of Algorithms **xix**

1 Introduction to Multidimensional Test Cell Data Modelling **1**
1.1 Objectives and Aim of Project 1
1.2 Description of Modelling Challenges and Selected Methods 2
1.3 Project Scope 2

2 Theory Behind Combustion Engines and Modelling Methods **5**
2.1 Four-Stroke Internal Combustion Engines 5
2.1.1 The Parts of the Engine 5
2.1.2 The Four Strokes 6
2.1.3 Engine Control Systems and Importance of Air Estimation . . 7
2.1.4 Scavenging 9
2.2 Gaussian Process Regression 9
2.2.1 Introduction to Regression Using Gaussian Processes 10
2.2.2 Kernels and Hyperparameters 12
2.3 Local Interpretable Model-Agnostic Explanations 15
2.4 Neural Networks 16
2.4.1 Definition and Construction 16
2.4.2 Training 17

3 Underlying Technologies of Modelling Methods **21**
3.1 Gaussian Process Regressor 21
3.2 Local Interpretable Model-Agnostic Explainer 22
3.3 Neural Network 23

4 Methodology **25**
4.1 Description of Model 25
4.1.1 Gaussian Process Regression Model 25
4.1.2 Neural Network Model 28
4.2 Building the Model 30
4.3 Visualization 33
4.4 Optimizing Prediction Runtime of Model 34

5 Results **41**

6 Discussion **47**
6.1 Comparison with Current Model 47
6.2 Justifying Selected Modelling Methods 48
6.3 Noise Reduction Process 48
6.4 Future Investigations 50

7 Conclusion **51**

Bibliography **53**

List of Figures

2.1 Schematic description of a four-stroke internal combustion gasoline engine. The parameters of equation (2.1) are the engine speed n – i.e. the RPM of the crankshaft (in the bottom of the figure), the pressure p_i in the intake manifold (at the top right of the figure), and the intake and exhaust cam phasers – ϕ_i and ϕ_e respectively – that are placed on the camshafts (at the top of the figure). Adapted from [5]. CC BY-SA 3.0 6

2.2 Schematic description of the four strokes in a four-stroke internal combustion gasoline engine. Figure (a) shows the first stroke (intake), where a mixture of air and fuel is inducted through the intake valve. The piston is moving downward. Figure (b) shows the second stroke (compression), where the piston moves upward and the air/-fuel mixture is compressed, which increases the pressure. Figure (c) shows the third stroke (power), where the spark plug at the top of the cylinder ignites the air/fuel mixture – causing the piston to move downward. Figure (d) shows the fourth stroke (exhaust), where the piston is again moving upward and the spent air/fuel mixture is released through the exhaust valve. (The intake and exhaust manifolds are mirrored compared to Figure 2.1.) From [8]. CC BY 4.0. 8

2.3 Example of noise free Gaussian process regression in one dimension. Figure (a) shows sample functions from the prior distribution, and Figure (b) shows the posterior distribution of functions given two data points, with the solid line representing the mean prediction. In both figures, the shaded region represents a 95% confidence interval. Notice how the confidence interval is smaller in Figure (b), where additional information is added, in particular close to the data points. From [11]. Reposted with permission. 12

2.4 Example of hyperparameter optimization when there are multiple local maxima in the marginal likelihood. Figure (a) shows a contour plot of the marginal likelihood as a function of the hyperparameters (in this case noise standard deviation σ and characteristic length-scale ℓ). There are two local maxima, marked with +. The two models corresponding to these local maxima are shown in Figures (b) and (c). Figure (b) shows the model for the global optimum, with relatively low noise and short length-scale. The model in Figure (c) has higher noise and a longer length-scale, and thus explains most fluctuations in the data as noise. From [11]. Reposted with permission. 14

2.5 Example of noise free Gaussian process regression in two dimensions, with an RBF kernel. Figure (a) shows the isotropic case, where the length-scale is $\ell = 1$ for both input parameters, hence they both are equally important for the output. In Figure (b), the length-scale is $\boldsymbol{\ell} = (1, 3)^T$, hence the output varies more rapidly as a function of x_1 than x_2. From [11]. Reposted with permission. 15

2.6 Example to illustrate how LIME works. The complex decision function f of some black box machine learning model is represented by the blue and pink background. The instance of interest is represented by the bold red cross, and the sampled instances are given by red crosses and blue dots. Notice that the sizes of the crosses and dots vary, to represent the weight π_x assigned to them based on proximity to the instance of interest. The predictions (red cross or blue dot) are given by evaluating f at the sampled points. Finally, the dashed line represents the explanation that is locally – but not globally – faithful (linear in this case). 16

2.7 Schematic illustration of a neural network. The number of neurons i in the input layer represent the number of input variables, and the number of neurons j in the output layer represent the number of output variables. The number of hidden layers k and the number of neurons in these layers (which can vary between layers – note that the number of neurons in hidden layer 1 is n but hidden layer k has m neurons), are chosen when building the neural network. Each neuron in the hidden layers has a bias b, see equation (2.23). To save space in the figure, only the weights w, see equation (2.23), for the first neuron (which is connected to the feature component x_1) in the first hidden layer are shown. 18

4.1 Flowchart of the program for the GPR/LIME model (to the left in the figure) and the program for the neural network model (to the right in the figure), as well as how they are connected. The red ovals mark the Start and End of each program. The yellow shapes with a curved bottom represent one or several files being loaded into – or given as output from – the programs. The green rectangles represent processes in the program and the purple parallelograms represent outputs from – and inputs to – these processes. 26

4.2 2D plot of radial basis function interpolation model (currently used at T-Engineering), for $\phi_\mathrm{i} = 10$, $\phi_\mathrm{e} = 10$, and $n = 3500$ RPM. The anomaly at $p_\mathrm{i} \approx 150\,\mathrm{kPa}$ has no reasonable physical explanation and thus has to be captured as an outlier by the GPR model. The data is scaled for NDA reasons. 32

4.3 2D plots of volumetric efficiency η_V as a function of p_i for the initial and smooth GPR model. The other features are fixed at $\phi_\mathrm{i} = 10$, $\phi_\mathrm{e} = 10$, and $n = 3500$ RPM. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\times$. The fitted model is represented by the black dashed line and the gray shaded area represents the 95% confidence interval. 35

4.4 3D plot of volumetric efficiency η_V as a function of p_i and n for the smooth GPR model. The cam phasers are fixed at $\phi_\mathrm{i} = 0$ and $\phi_\mathrm{e} = 0$. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the p_i interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\blacksquare$. The fitted model is represented by the surface area. Notice that the filtered out data points are mainly around $p_\mathrm{i} \approx 50\,\mathrm{kPa}$, which is where η_V starts decreasing more rapidly toward zero. Compare with Figure 5.2. The data is scaled for NDA reasons. 36

4.5 Example of the visualization of the GPR model using widgets in Jupyter Notebook. The sliders above the plots control the confidence interval (shown as the light blue area) used in the noise reduction process, as well as the parameters `alpha` and `alpha_smooth`. The plots show a 2D intersection of the GPR model, where the cam phasers are fixed to $\phi_\mathrm{i} = 10$, $\phi_\mathrm{e} = 10$, and the engine speed is fixed to $n = 3500$ RPM. When moving the sliders, the size of the confidence interval, as well as the shape of the model, will change. The plots are connected, so if the slider of the initial model is changed, so will one for the smooth model. Compare with Figure 4.3. The data is scaled for NDA reasons. 37

4.6 Development of losses – Figure (a) – and natural logarithm of losses – Figure (b) – over epochs for the training of the neural network. The losses are calculated using equation (2.24). Since the lines for training data and test data follow each other in both figures, we can assume that the model does not overfit. 38

4.7 2D plot of volumetric efficiency η_V as a function of p_i for the neural network model. The other features are fixed at $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The measured data points are marked with red $\star$, and the black dashed line represents the fitted model. Compare with Figure 4.3. The data is scaled for NDA reasons. 39

4.8 3D plot of volumetric efficiency η_V as a function of p_i and n for the neural network model. The cam phasers are fixed at $\phi_i = 0$ and $\phi_e = 0$. The measured data points in are marked by $\star$, and the fitted model is represented by the surface area. Compare with Figure 4.4. The data is scaled for NDA reasons. 40

5.1 2D plot of volumetric efficiency η_V with respect to p_i for the GPR model, the neural network model, and the RBF model. The other features are fixed at $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The measurement data points are marked with green $\star$, the black solid line represents the fitted GPR model, the red dashed line represents the fitted neural network model, and the blue dotted line represents the fitted RBF model. The GPR and neural network model follow each other, whereas the RBF model fluctuates more and turns more slowly toward $\eta_V = 0$. Compare to Figures 4.2, 4.3, and 4.7. The data is scaled for NDA reasons. 42

5.2 3D plot of volumetric efficiency η_V as a function of p_i and n for the smooth GPR model. The cam phasers are fixed at $\phi_i = 50$ and $\phi_e = 50$. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the p_i interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\blacksquare$. The fitted model is represented by the surface area. Notice that the filtered out data points are mainly at low p_i and low n, and that η_V is high for high values of p_i and low n. Compare with Figure 4.4. The data is scaled for NDA reasons. 43

5.3 Histograms of volumetric efficiency error distribution for the GPR model and the neural network model. The red histograms and red dashed lines indicate the error (and kernel density distribution) for the full dataset, while the black histograms and black solid lines indicate the error (and kernel density distribution) for the filtered data set. Both histograms for the filtered data set are unimodal and symmetric; however, the one for the neural network model has slightly heavier tails. Compare with Figure 5.5. 45

5.4 Probability plots of volumetric efficiency error for the GPR model and the neural network model (filtered data set). The red line is a least-squares regression line fit to the points. In both plots, the points are close to this linear relationship between theoretical and sample quantiles, deviating slightly at the ends. 46

5.5 Kernel density distribution of volumetric efficiency error for the neural network model (filtered data set), when its underlying GPR model has been trained with 72% – black solid line, 54% – red dashed line, and 36% – blue dotted line – of the measurement data, respectively. The error distribution has heavier tails when a smaller proportion of data is used in training. Compare with Figure 5.3, where the GPR model has been trained with 72% of the measurement data. The data is scaled for NDA reasons. 46

6.1 Comparison between two GPR models, where the model in Figure (a) has no lower bound to the `length_scale` parameter – as opposed to the one in Figure (b) which has a lower bound of 1.5. Notice how this makes the surface in Figure (b) smoother than the one in Figure (a), in line with what is physically reasonable. 49

List of Tables

4.1 Example of `DataFrame` containing measurement data for engine speed n (`EngSpd`), intake manifold pressure p_i (`p_Man`), intake cam phaser ϕ_i (`ICPC`), exhaust cam phaser ϕ_i (`ECPC`), and volumetric efficiency η_V (`eta_vol`). These quantities are further described in Section 2.1. The data is scaled for NDA reasons. 30

4.2 The optimized values for the length scale hyperparameter of the Matérn kernel for the four input features $(n, p_\mathrm{i}, \phi_\mathrm{i}, \phi_\mathrm{e})$. The "Initial model" column represents the values for the first step in the GPR model, and the "Smooth model" column represents the values for the smooth GPR model – after noise reduction was performed. The lower length scale bound was set to 1.5, which affected the length scale for intake manifold pressure p_i. 32

4.3 Example of output from the LIME explainer for a point/data row with high variance. For this data row, the highest contributing feature (they are presented in descending order) is `ICPC > 40.00`, i.e. that the intake cam phaser ϕ_i is set to a high crank angle. Since this explanation has the highest variance correlation, it will be stored in the `explanations` list. Notice that `p_Man` has a negative variance correlation, meaning that the intake manifold pressure p_i is in an interval where the variance of the model decreases. 33

5.1 Runtimes for prediction of volumetric efficiency for 50 data points from the GPR and neural network model, respectively, calculated as the mean over 1 800 runs. The programs are run on a AMD Ryzen 7 PRO 4750U with Radeon Graphics 1.70 GHz and 32.0 GB RAM. . . . 44

List of Algorithms

3.1 The algorithm implemented by the `scikit-learn` class `GaussianProgressRegressor`. Line 3 and 4 calculate the predictive mean – compare to equation (2.17), line 5 and 6 calculate the predictive variance – compare to equation (2.18), and line 7 calculates the log marginal likelihood. As seen in line 2, the implementation uses Cholesky factorization (see Appendix A.4 in [2]) for matrix inversion. . 21
3.2 Algorithm describing the training process of a neural network. The `for` loop is run for a pre-defined number of epochs. 24

4.1 Pseudo code describing the nested `for` loop used to create multiple plots for visualization in Jupyter Notebook. 34

1
Introduction to Multidimensional Test Cell Data Modelling

This chapter provides an introduction to the subject of multidimensional test cell data modelling. In Section 1.1, the objectives and aim of the project are listed. Section 1.2 describes the challenges of the current model at T-Engineering – that this project thus has to deal with – as well as the selected modelling methods. Finally, in Section 1.3 the scope of the project is discussed.

T-Engineering works with designing and developing control systems for different types of vehicle systems. Recent requirements for lower emissions and improved energy efficiency in vehicles lead to more components being added to the engines. This, in turn, leads to more degrees of freedom in the functions describing relations between different parameters of the engine. Trying to model these multidimensional relationships using measurement data can be difficult, especially when there is noise in the data. Another challenge is the limited CPU power that comes with trying to keep component costs low.
The purpose of this master's thesis is to find an efficient and fast way to inter- and extrapolate multidimensional functions. More specifically, the project focuses on modelling volumetric efficiency (further described in Section 2.1) which is a function of four parameters. Furthermore, noise reduction must be performed on the collected data to obtain reasonably smooth functions. The produced model should be able to make real-time predictions using limited CPU power.

1.1 Objectives and Aim of Project

This section describes the key objectives and overall aim of the project.

An initial objective of the project is to build a model that handles multidimensional input. Noise reduction must be performed, and the model should accurately describe the measurement data. A further objective is to find a way to identify and handle lack of data for some intervals, e.g. where measurements are difficult to perform. The multidimensional format makes it challenging to visualize this in plots; hence, there is a need for other methods in order to discover these regions. The overall aim of the project is, in line with the purpose, to represent the obtained model in an efficient way that can be used on a limited CPU in the control systems.
The objectives can be formulated as the following questions, to be investigated:

- Is it possible to build a model that accurately describes the relation between

input and output parameters, indicating uncertainty where needed?
- Can the multidimensional results be visualized in a pedagogical way?
- Can the model operate in real time on a limited CPU?

1.2 Description of Modelling Challenges and Selected Methods

This section describes the challenges of the current model at T-Engineering. It also presents the three modelling methods – Gaussian process regression, local interpretable model-agnostic explanations (LIME) and neural networks – that will be used in this project.

The current model at T-Engineering uses radial basis function interpolation (see [1] for a description of this method). It requires a lot of measurement data, which makes it time-consuming when it comes to generating the data (i.e. performing the measurements). Furthermore, the model does not indicate where the uncertainty of the predictions is specifically high; hence, there is no direct way of telling which regions need more measurement data and which regions do not. Should this be known, the measurements in the test cell can be planned more precisely and thus be more efficient. The noise reduction in the current model does not capture some measurement errors in the data. In Section 4.2, an anomaly in the data is discussed which is not properly handled by the current model.
In order to handle these modelling challenges, Gaussian process regression was chosen as the main modelling method. Gaussian process regression – further described in Sections 2.2 and 3.1 – makes it possible to know where the model is uncertain, by calculating the variance of the model predictions in each point. Furthermore, it does not require too much prior knowledge of the data – only main characteristics such as smoothness or periodicity. [2] For the purpose of finding which features are contributing to increasing the variance, it is also necessary to explain the predictions for the variance. This is done using a local surrogate model called LIME, which is further described in Sections 2.3 and 3.2. [3]
As Gaussian process regression might not be fast enough in its predictions (see Section 3.1) to satisfy the real time condition presented in Section 1.1, a neural network was used as an addition to the Gaussian process regression model. Neural networks are further described in Sections 2.4 and 3.3.

1.3 Project Scope

This section gives a brief description of the scope of the project.

As stated above, the project aims to model volumetric efficiency (further described in Section 2.1) – a function of four parameters. There are several other multidimensional functions that are important at T-Engineering, e.g. spark maps, efficiency maps and air models. The decision to focus on volumetric efficiency was based on two main factors: there is lots of measurement data for this quantity, and it is relatively straightforward to obtain from the measurement data (since it is not measured

explicitly). The idea is to use the concepts from this model when developing models for other quantities.

The selected modelling methods are described in Section 1.2. There are other possible methods that could be used for addressing these challenges, such as linear modelling, decision trees, or random forest [4]. However, both linear methods and decision trees have some difficulties in capturing complex behaviors and require a lot of initial expert knowledge about the data. Furthermore, random forest – while being being capable of capturing complex patterns – does not indicate uncertainty of its predictions.

The main purpose of the visualization part of the project is to find a pedagogical way to assimilate information from the model. Thus, the focus is not on exploring many different visualization packages or softwares but instead on finding an adequate and understandable representation of the data. This is done by making 2D and 3D plots in Python using `matplotlib` and by using interactive widgets in Jupyter Notebook.

2

Theory Behind Combustion Engines and Modelling Methods

This chapter provides theoretical information that might be needed to understand the rest of the report. Section 2.1 gives an introduction to a four-stroke internal combustion engine and the modelled quantities. The theory behind Gaussian process regression is presented in Section 2.2. Section 2.3 describes the theory behind local interpretable model-agnostic explanations (LIME). Finally, Section 2.4 gives a theoretical description of neural networks.

2.1 Four-Stroke Internal Combustion Engines

This section describes the parts of a four-stroke internal combustion engine (Section 2.1.1) as well as the four strokes of the engine (Section 2.1.2), and the physical quantities of the model. The importance of air estimation, and how it is connected to volumetric efficiency, is described in Section 2.1.3. Section 2.1.4 gives a brief explanation of the scavenging process.

The model introduced in Chapter 1 should describe the volumetric efficiency of a four-stroke internal combustion (IC) gasoline engine, as specified in Section 1.3. Volumetric efficiency η_V (defined further in Section 2.1.3) is a function of four parameters:

$$\eta_\mathrm{V} = f(n, p_\mathrm{i}, \phi_\mathrm{i}, \phi_\mathrm{e}). \tag{2.1}$$

Here, n is the engine speed measured in revolutions per minute (RPM), p_i is the intake manifold pressure measured in kPa, and ϕ_i and ϕ_e are the intake and exhaust cam phasers, respectively. In order to understand these parameters, below is a brief description of how a four-stroke IC engine works.

2.1.1 The Parts of the Engine

Figure 2.1 shows a schematic image of a four-stroke IC gasoline engine (also known as a *spark ignition* or *SI* engine, as opposed to a diesel engine which is *compression ignition* or *CI*). At the top, there are two types of manifolds with valves: an intake manifold where a mixture of air and fuel (gasoline) is inserted into the combustion chamber in the beginning of the four-stroke cycle, and an exhaust manifold where the exhaust gases are released at the end of the cycle. Above the valves are camshafts

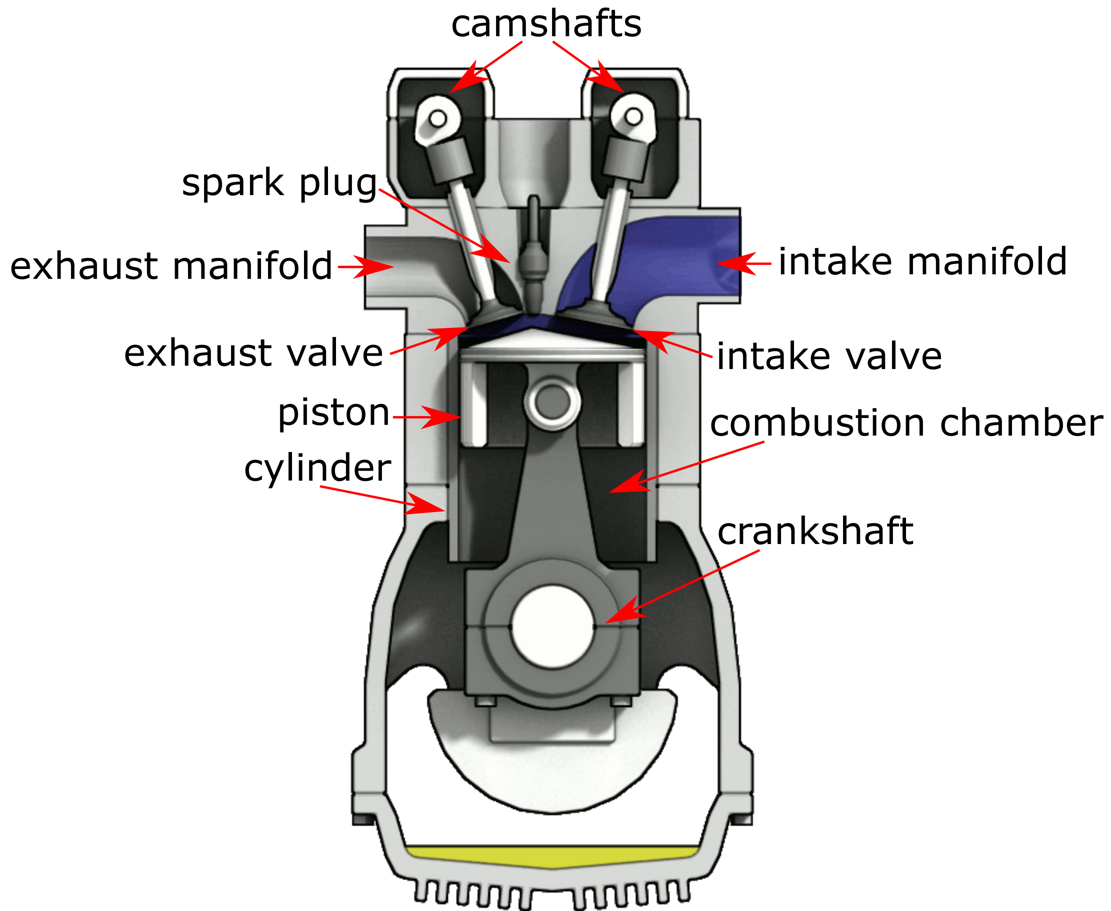


Figure 2.1: Schematic description of a four-stroke internal combustion gasoline engine. The parameters of equation (2.1) are the engine speed n – i.e. the RPM of the crankshaft (in the bottom of the figure), the pressure p_i in the intake manifold (at the top right of the figure), and the intake and exhaust cam phasers – ϕ_i and ϕ_e respectively – that are placed on the camshafts (at the top of the figure). Adapted from [5]. CC BY-SA 3.0

that operate the valves, i.e. opening and closing them in time with the strokes. The camshafts have phasers that control the timing of the valves by controlling the camshaft's position in relation to the crankshaft. The intake cam phaser and the exhaust cam phaser are ϕ_i and ϕ_e in equation (2.1) above. In between the valves is a spark plug which is used to ignite the air/fuel mixture. In the cylinder, there is a piston moving up and down during the strokes. The piston is connected to a crankshaft that converts the linear motion of the piston to rotational motion. The RPM of the crankshaft is given as the engine speed n in equation (2.1). [6]

2.1.2 The Four Strokes

In Figure 2.2, the four strokes of the SI engine are shown. A stroke is defined as the piston's movement from the *top dead center (TDC)* – closest to the valves – to the *bottom dead center (BDC)* – furthest from the valves, or vice versa. At first,

the piston is at the TDC and the air/fuel mixture is inducted into the combustion chamber through the intake valve, which opens just before the stroke begins. This is known as the *intake stroke*, shown in Figure 2.2a. The intake manifold pressure p_i in equation (2.1) is the pressure of the air/fuel mixture as it enters the combustion chamber. The piston moves down toward BDC, causing a suction of air/fuel mixture into the combustion chamber. Next, the intake valve closes and the piston starts to move upward toward TDC again, thus compressing the air/fuel mixture and increasing the pressure in the combustion chamber. This is known as the *compression stroke*, shown in Figure 2.2b. When the piston reaches TDC, the spark plug ignites the air/fuel mixture, causing it to expand and push the piston down toward BDC. This is known as the *power* or *combustion stroke*, shown in Figure 2.2c. Finally, the exhaust valve opens and the exhaust gases are released as the piston moves up toward TDC. This is known as the *exhaust stroke*, shown in Figure 2.2d. When the piston reaches TDC, the exhaust valve closes and a new four-stroke cycle begins. [6]
Even at TDC, there is still some space between the piston and the top of the cylinder, known as the *clearance volume*. This means that, in the exhaust stroke, not all gases will be pushed out. The remaining exhaust gases in the clearance volume are called *residual gases*. They will mix with the incoming mixture of air and fuel that enters the engine in the intake stroke. A large proportion of residual gases will decrease the engine performance since there is not as much unused fuel to ignite in the power stroke. [6]
For each stroke in the cycle, the crankshaft moves 180° – or 180 crank degrees (as opposed to cam degrees, which will be explained shortly). The camshafts are connected to the crankshaft at a 1:2 ratio, meaning they revolve once for every two turns of the crankshaft. One cam degree is therefore equal to two crank degrees. [7]
In equation (2.1), ϕ_i and ϕ_e are measured in crank degrees.

2.1.3 Engine Control Systems and Importance of Air Estimation

As mentioned in Chapter 1, T-Engineering develops control systems for vehicle systems. When controlling SI engines, the air mass trapped in the cylinder (see Figure 2.1) is used to estimate

- the amount of torque produced by the engine
- how much fuel to inject in order to meet the desired air/fuel ratio
- when to ignite the air/fuel mixture.

Hence, knowing the trapped air mass in the cylinder is critical in these control systems. Unfortunately, it cannot be measured directly. There are devices such as a *mass air flow sensor (MAF)*; however, they are expensive and typically located away from the cylinder, where conditions in the engine are favorable and with low variations, thus measuring incorrect values during transients (i.e. changes of intake manifold pressure p_i and/or engine speed n).
Fortunately, there is another way of determining the trapped air mass – a method called speed density. This method is based on modelling how well the engine captures air during the intake stroke (see Section 2.1.2) using measured physical quantities such as temperature T, intake manifold pressure p_i, and volumetric efficiency η_V.

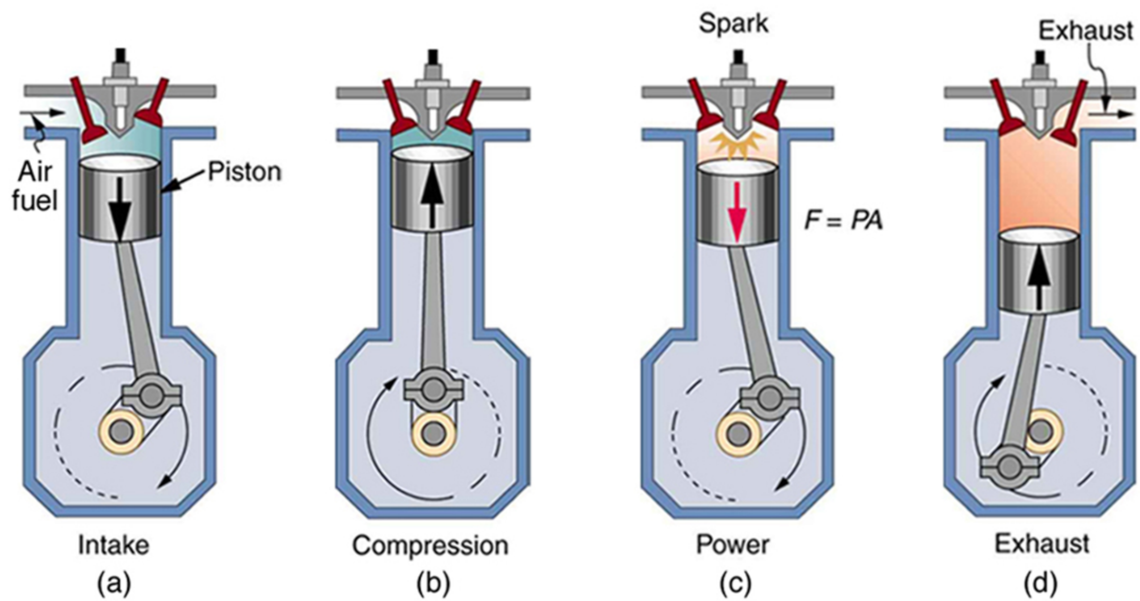


Figure 2.2: Schematic description of the four strokes in a four-stroke internal combustion gasoline engine. Figure (a) shows the first stroke (intake), where a mixture of air and fuel is inducted through the intake valve. The piston is moving downward. Figure (b) shows the second stroke (compression), where the piston moves upward and the air/fuel mixture is compressed, which increases the pressure. Figure (c) shows the third stroke (power), where the spark plug at the top of the cylinder ignites the air/fuel mixture – causing the piston to move downward. Figure (d) shows the fourth stroke (exhaust), where the piston is again moving upward and the spent air/fuel mixture is released through the exhaust valve. (The intake and exhaust manifolds are mirrored compared to Figure 2.1.) From [8]. CC BY 4.0.

The trapped air mass m_a is calculated from the general gas equation as

$$\eta_\text{V} p_\text{i} V_\text{d} = m_\text{a} R T \Rightarrow m_\text{a} = \eta_\text{V} \frac{p_\text{i} V_\text{d}}{RT}, \tag{2.2}$$

with V_d being the swept cylinder volume and R the specific gas constant given by

$$R = \frac{\tilde{R}}{M}, \tag{2.3}$$

where $\tilde{R} \approx 8.31\,\text{J}\,\text{K}^{-1}\,\text{mol}^{-1}$ is the ideal gas constant and $M \approx 0.028\,\text{kg}\,\text{mol}^{-1}$ is the molar mass of air. Benefits of the speed density method include low costs – a pressure sensor is less expensive than a MAF – and that measurements are made closer to the cylinder, meaning that it handles transients better.
In order to use the speed density method, we need to be able to model the volumetric efficiency – η_V in equation (2.2) – of the engine. The volumetric efficiency η_V of a four-stroke IC engine is defined as the proportion of swept cylinder volume V_d that is filled with air at intake manifold pressure p_i. In other words, one could say that η_V describes how well the cylinder volume is filled and thus used. A large η_V gives more power to the engine, since more fuel can then be inducted and burned for a given engine displacement. It is possible for η_V to reach above 100%, e.g. if p_i is higher than the pressure in the combustion chamber or at the exhaust valve. [6] Volumetric efficiency η_V can be measured in a test-cell by weighing the fuel before and after entering the engine. The ratio between air and fuel in the mixture is known; hence, the amount of air and thus η_V can be calculated. When running the car, η_V must be accurately modelled in order to be properly evaluated in the control systems.

2.1.4 Scavenging

The process between two four-stroke cycles (see Section 2.1.2) – when the exhaust valve is open to release the exhaust gases and the intake valve is open to induct new air/fuel mixture – is called *scavenging*. If there is a large overlap between the cam phasers ϕ_i and ϕ_e (see Section 2.1.1), the air flow increases and so does the volumetric efficiency η_V. However, since there is a large flow of air through the engine, η_V is difficult to measure in these cases. Furthermore, if the intake manifold pressure p_i is lower than the exhaust manifold pressure, residual gases (see Section 2.1.2) will flow back into the cylinder and impair combustion.

2.2 Gaussian Process Regression

This section presents the theory behind Gaussian process regression by first giving an introduction to the subject (Section 2.2.1) and then describing kernels and hyperparameters (Section 2.2.2).

Gaussian process regression (GPR) is a type of supervised machine learning, with a Bayesian approach [2]. In Bayesian inference, we assume that the parameter(s) of interest θ has a *prior* probability distribution $g(\theta)$ – prior meaning that this is the knowledge that we have about θ before introducing any data to our model. Given

some data x and the *likelihood* $f(x|\theta)$ (the likelihood of x given θ), we can compute the conditional distribution $h(\theta|x)$ using Bayes' formula:

$$h(\theta|x) = \frac{f(x|\theta)g(\theta)}{\int f(x|\theta)g(\theta)\mathrm{d}\theta}. \tag{2.4}$$

This is called the *posterior* distribution because it is calculated after introducing data to the model. Since the denominator is integrated over all θ it can be viewed as a constant and equation (2.4) can be more simply expressed as

$$\text{posterior} \propto \text{likelihood} \times \text{prior},$$

where $\propto$ means proportional to. [9]

2.2.1 Introduction to Regression Using Gaussian Processes

The Gaussian, or normal, distribution is well known in statistics. Its probability density function is given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad x \in \mathbb{R}, \tag{2.5}$$

where μ is the mean and σ^2 is the variance. A Gaussian distribution with mean μ and variance σ^2 is denoted $\mathcal{N}(\mu, \sigma^2)$. In the case of multiple variables, the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{V})$ has the joint probability density function

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{V}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})\boldsymbol{V}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})'\right], \quad \boldsymbol{x} \in \mathbb{R}^n. \tag{2.6}$$

Here $\boldsymbol{\mu} \in \mathbb{R}^n$ and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ is a positive definite symmetric matrix. [10]
A Gaussian process is defined as follows in [2]:

Definition 2.2.1. *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

For Gaussian process regression, we assume that the prior function $\boldsymbol{f}(\boldsymbol{x})$ (see Section 2.2) can be described by a zero mean Gaussian process, i.e. that

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_n) \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{bmatrix}\right), \tag{2.7}$$

where $k(\cdot,\cdot)$ is called a *kernel* or *covariance function.* The $n \times n$ covariance matrix (or *Gram matrix*) can also be denoted $K(X, X)$. There are different types of kernels which in turn have free parameters – called hyperparameters – that can be chosen to optimize the fit. See Section 2.2.2 for a further description of kernels and hyperparameters.
First, we consider the case where there is no noise in the data. Letting $\boldsymbol{f}_*$ describe the function values at the test points, we get the joint distribution

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{2.8}$$

of measured values $\boldsymbol{f}$ and predicted values $\boldsymbol{f}_*$. The conditional distribution for the posterior function $\boldsymbol{f}_*$ then becomes

$$\boldsymbol{f}_* | X, \boldsymbol{f}, X_* \sim \mathcal{N}\left(\bar{\boldsymbol{f}}_*, \operatorname{cov}(\boldsymbol{f}_*)\right), \tag{2.9}$$

where

$$\bar{\boldsymbol{f}}_* := \mathbb{E}\left[\boldsymbol{f}_* | X, \boldsymbol{f}, X_*\right] = K(X_*, X)\left[K(X,X)\right]^{-1} \boldsymbol{f} \tag{2.10}$$

$$\operatorname{cov}(\boldsymbol{f}_*) = K(X_*, X_*) - K(X_*, X)\left[K(X,X)\right]^{-1} K(X, X_*). \tag{2.11}$$

A more intuitive description: Given a prior distribution – that can be described as a Gaussian process – a number of sample functions are generated. Then, given a set of measurement data, the functions that pass through the measured data points are given a higher likelihood and thus have a greater impact on the posterior distribution (see equation (2.4)) over the functions. Figure 2.3 shows an example in one dimension, where the sample functions from the prior distribution are shown in Figure 2.3a and the posterior distribution is shown in Figure 2.3b. The dashed lines are the sample functions that match the data set, and the solid line represents the mean prediction $\bar{\boldsymbol{f}}_*$ in equation (2.10). The confidence interval is shown as the shaded region and is given from the covariance $\operatorname{cov}(\boldsymbol{f}_*)$ in equation (2.11) as twice the standard deviation. Since the uncertainty is smaller closer to the measured data points, this can also give a good indication as to where more measurements need to be performed. [2]
However, it is likely that there is noise in the data and hence the underlying function that describe the data might not pass exactly through the measured data points. We then instead assume that we have made n observations $\{y_i\}_{i=1}^n \in \mathbb{R}$ corresponding to input $\{\boldsymbol{x}_i\}_{i=1}^n$ as

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i, \tag{2.12}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ represents i.i.d. Gaussian distributed noise that is independent of $f(\boldsymbol{x}_i)$. Then $\boldsymbol{y}(\boldsymbol{x})$ – the sum of independent Gaussian random variables – is a Gaussian process described by

$$\boldsymbol{y} \sim \mathcal{N}(\mathbf{0}, K(X,X) + \sigma^2 I), \tag{2.13}$$

where I is the identity matrix, and we get the joint distribution

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2.14}$$

The likelihood is a factorized Gaussian given by

$$\boldsymbol{y} | \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2 I) \tag{2.15}$$

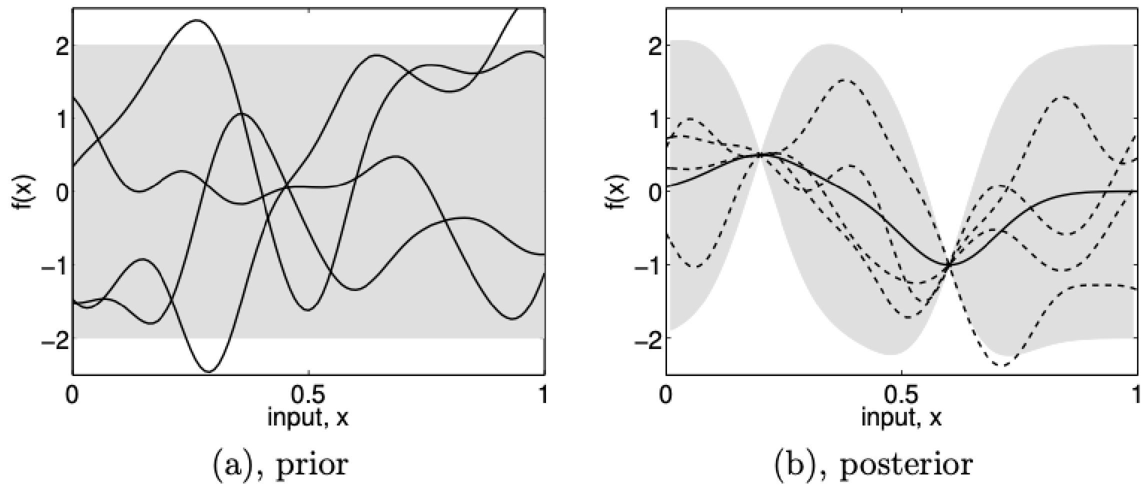


Figure 2.3: Example of noise free Gaussian process regression in one dimension. Figure (a) shows sample functions from the prior distribution, and Figure (b) shows the posterior distribution of functions given two data points, with the solid line representing the mean prediction. In both figures, the shaded region represents a 95% confidence interval. Notice how the confidence interval is smaller in Figure (b), where additional information is added, in particular close to the data points. From [11]. Reposted with permission.

(which in the noise-free case would be $\boldsymbol{f}|\boldsymbol{f}$, clearly having zero variance). The conditional distribution – equation (2.9) – for the posterior function $\boldsymbol{f}_*$ now becomes

$$\boldsymbol{f}_*|X, \boldsymbol{y}, X_* \sim \mathcal{N}\left(\bar{\boldsymbol{f}}_*, \operatorname{cov}(\boldsymbol{f}_*)\right), \tag{2.16}$$

where

$$\bar{\boldsymbol{f}}_* := \mathbb{E}\left[\boldsymbol{f}_*|X, \boldsymbol{y}, X_*\right] = K(X_*, X)\left[K(X,X) + \sigma^2 I\right]^{-1} \boldsymbol{y} \tag{2.17}$$

$$\operatorname{cov}(\boldsymbol{f}_*) = K(X_*, X_*) - K(X_*, X)\left[K(X,X) + \sigma^2 I\right]^{-1} K(X, X_*), \tag{2.18}$$

compare with equations (2.10) and (2.11). [2] Since the posterior distribution is dependent on the measurement data, a GPR model will return to the mean of the prior in regions with no measurement data. For many commonly considered kernels, assuming a zero mean prior means that the model will go toward a zero mean for e.g. extrapolation. [12]

2.2.2 Kernels and Hyperparameters

In order to find an adequate prediction $\bar{\boldsymbol{f}}_*$ – equation (2.17) – we must use a suitable kernel. An example of a simple kernel is the *dot product kernel*, given by

$$k(x_i, x_j) = \sigma_0^2 + x_i \cdot x_j. \tag{Dot Product Kernel}$$

This is equal to Bayesian linear regression [13], with a $\mathcal{N}(0, \sigma_0^2)$ prior on the bias and $\mathcal{N}(0,1)$ priors on the coefficients. Another common kernel is the *squared exponential – or radial basis-function (RBF) – kernel*, defined as

$$k(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2\ell^2}\right). \qquad \text{(RBF Kernel)}$$

Here, $d(\cdot,\cdot)$ denotes the Euclidean distance and $\ell > 0$ is a hyperparameter called the length-scale parameter. [4] The characteristic length-scale of a kernel can be described as the distance we have to move in input space before significant changes in the function values can occur [2]. This kernel is infinitely differentiable, resulting in very smooth functions. For functions that are not infinitely differentiable, the RBF kernel can be generalized into the *Matérn kernel*:

$$k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{\ell}d(x_i, x_j)\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{\ell}d(x_i, x_j)\right), \qquad \text{(Matérn Kernel)}$$

where $\Gamma(\cdot)$ is the gamma function, $K_\nu(\cdot)$ is a modified Bessel function, and $\nu > 0$ is a hyperparameter controlling the smoothness of the function. If $\nu \to \infty$, we get the RBF kernel. Functions that are (at least) once, or twice, differentiable will have $\nu = 3/2$ and $\nu = 5/2$, respectively. With $\nu = 5/2$, we get

$$k(x_i, x_j) = \left(1 + \frac{\sqrt{5}}{\ell}d(x_i, x_j) + \frac{5}{3\ell}d(x_i, x_j)^2\right)\exp\left(-\frac{\sqrt{5}}{\ell}d(x_i, x_j)\right). \qquad (2.19)$$

If we sum infinitely many RBF kernels with different length-scales, we get the *rational quadratic kernel*. It is defined as

$$k(x_i, x_j) = \left(1 + \frac{d(x_i, x_j)^2}{2\alpha\ell^2}\right)^{-\alpha}, \qquad \text{(RQ Kernel)}$$

where $\alpha > 0$ is called a scale-mixture parameter.

There are many other kernels; however, these are the ones used in this project. Kernels can also be combined, e.g. by adding or multiplying, in order to get more complex ones [13]. By parameterizing the hyperparameters of the kernel into a vector $\boldsymbol{\theta}$, we can use the gradient ascent method to maximize the marginal likelihood (or log-marginal-likelihood) of $\boldsymbol{\theta}$ in order to find its optimal values. With the gradient ascent method it is, however, possible to find a local maximum that might not be global. Hence, if there are several local maxima, it is possible to find two (or more) models that interpret the data differently. Figure 2.4 shows an example where there are two local maxima in the marginal likelihood. [4] In Section 3.1, there is a description of how to handle this issue when implementing Gaussian process regression in Python.

If the input space has more than one dimension, the kernels can be isotropic – invariant to rotations in input space – or anisotropic. Different parameters of the input space might have different importance on the output, and the kernel must then reflect that. Figure 2.5 shows examples of Gaussian process regression models in two dimensions for an isotropic – Figure 2.5a – and anisotropic – Figure 2.5b – kernel. [2]

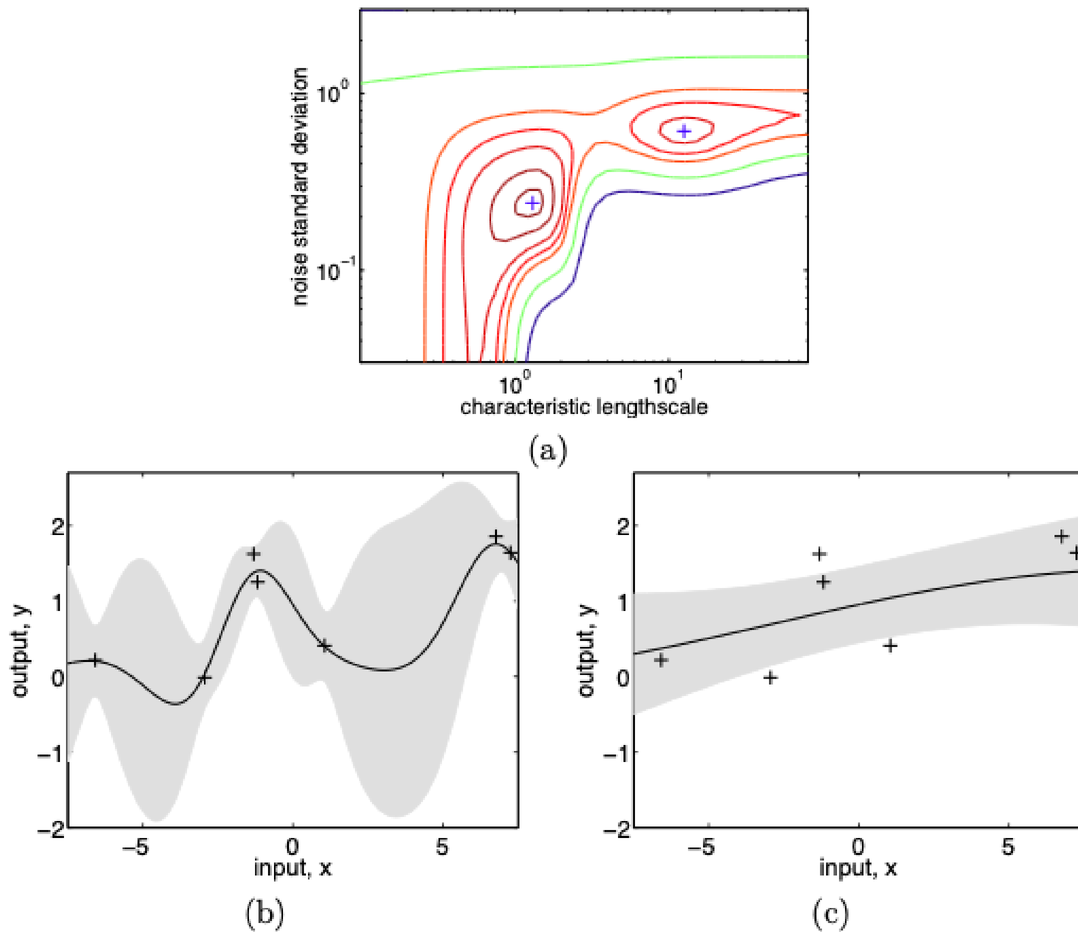


Figure 2.4: Example of hyperparameter optimization when there are multiple local maxima in the marginal likelihood. Figure (a) shows a contour plot of the marginal likelihood as a function of the hyperparameters (in this case noise standard deviation σ and characteristic length-scale ℓ). There are two local maxima, marked with +. The two models corresponding to these local maxima are shown in Figures (b) and (c). Figure (b) shows the model for the global optimum, with relatively low noise and short length-scale. The model in Figure (c) has higher noise and a longer length-scale, and thus explains most fluctuations in the data as noise. From [11]. Reposted with permission.

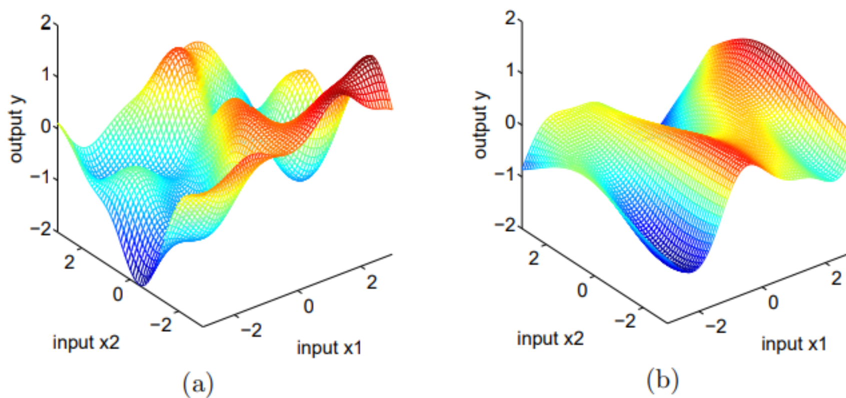


Figure 2.5: Example of noise free Gaussian process regression in two dimensions, with an RBF kernel. Figure (a) shows the isotropic case, where the length-scale is $\ell = 1$ for both input parameters, hence they both are equally important for the output. In Figure (b), the length-scale is $\boldsymbol{\ell} = (1, 3)^T$, hence the output varies more rapidly as a function of x_1 than x_2. From [11]. Reposted with permission.

2.3 Local Interpretable Model-Agnostic Explanations

This section gives an introduction to the theory behind local interpretable model-agnostic explanations (LIME).

Local interpretable model-agnostic explanations (LIME) is a type of local surrogate model used to explain individual predictions of black box machine learning models. It works by making predictions of the black box model for variations of the data, with the purpose of finding a local approximation of the model that is more interpretable. The variations are created by perturbing one feature at a time to see which features have the most impact. An interpretable model is then fit to the new data set, where the distance between the sampled instances and the instance of interest is used as weight. If the original model is given by $f : \mathbb{R}^d \to \mathbb{R}$ – where d is the number of features – and the instance of interest is denoted $x \in \mathbb{R}^d$, we can denote the interpretable representation of x as $x' \in \{0, 1\}^{d'}$, where $d' < d$ is the number of features used in the explainer. The explanation is then given by

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g), \tag{2.20}$$

where $g : \mathbb{R}^{d'} \to \mathbb{R}$ is the model in the family G of possible explanation models that minimizes the loss $\mathcal{L}$ while keeping the model complexity Ω low. The loss $\mathcal{L}$ measures how close the explanation g is to the prediction of the original model f. Model complexity Ω concerns how many features are used in the explanation, and in order to get a model that is interpretable by humans this needs to be kept low. The parameter $\pi_x(z)$ is called proximity measure between an instance z and the instance of interest x, and is used to define the neighborhood around x. [3][14]
Since LIME is model-agnostic, i.e. treating the original model as a black box, $\mathcal{L}(f, g, \pi_x)$ should be minimized without making any assumptions about f. To learn

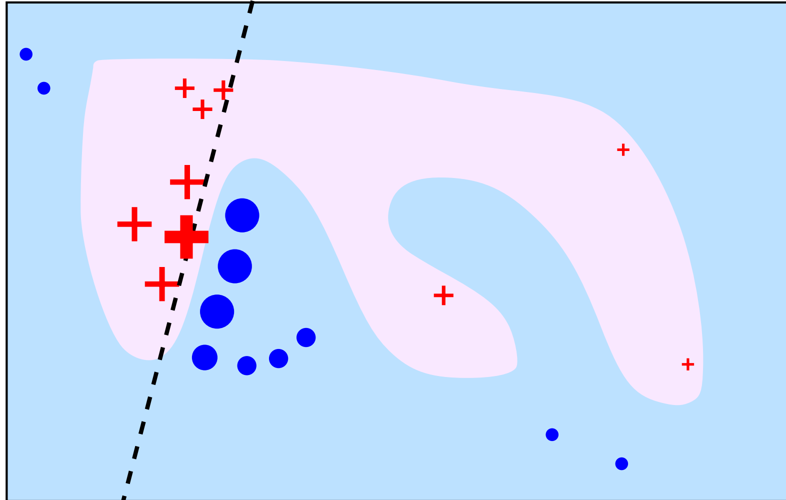


Figure 2.6: Example to illustrate how LIME works. The complex decision function f of some black box machine learning model is represented by the blue and pink background. The instance of interest is represented by the bold red cross, and the sampled instances are given by red crosses and blue dots. Notice that the sizes of the crosses and dots vary, to represent the weight π_x assigned to them based on proximity to the instance of interest. The predictions (red cross or blue dot) are given by evaluating f at the sampled points. Finally, the dashed line represents the explanation that is locally – but not globally – faithful (linear in this case).

the local behavior around an instance x, $\mathcal{L}(f, g, \pi_x)$ is approximated by drawing samples around its interpretable representation x' and weighing them by π_x – since we want the model to be local, samples closer to x' are given a higher weight. Given a perturbed sample $z' \in \{0,1\}^{d'}$, the sample $z \in \mathbb{R}^d$ from the original representation is recovered. The function value $f(z)$ is then obtained from the black box model and used as a label – for classification – or feature weight – for regression – in the explanation model. Equation (2.20) is then optimized based on the set of perturbed samples and their associated labels or feature weights. Figure 2.6 shows an example where the global model is fairly complex; however, locally it is possible to find an interpretable explanation. [14]

2.4 Neural Networks

This section describes the theory behind neural networks – definition and construction (Section 2.4.1) as well as training (Section 2.4.2).

2.4.1 Definition and Construction

Artificial neural networks are inspired by the neural networks of the human brain and consist of different types of *neurons* that are connected to each other. Henceforth, the terms neural networks (NN) and neurons will refer to the artificial kind unless otherwise stated. A neuron can be described as a parameterized function

$$y = f(x_1, \ldots, x_n; w_1, \ldots, w_n), \tag{2.21}$$

where $\{x_i\}$ are the input variables and $\{w_i\}$ are the parameters – or *weigths* – of the neuron. The function f is called an activation function and can be e.g. a sigmoid function or a *Rectified Linear Unit function (ReLU).* The ReLU function is defined as

$$f(v) = \max(0, v), \tag{2.22}$$

i.e. $f(v) = v$ if $v > 0$ and 0 otherwise. The input v to the activation function is a weighted sum of the variables, sometimes combined with a constant bias term b:

$$v = b + \sum_{i=1}^{n} w_i x_i. \tag{2.23}$$

The neurons are connected as nodes in a network, see Figure 2.7 for an example illustration. There is one input layer – where the number of nodes is the same as the number of input variables, a specified number of hidden layers, and an output layer – where the number of nodes is the same as the number of output variables. In Figure 2.7, we see that the input variables $x_1, \ldots, x_i$ are each connected to all n neurons in the 1st hidden layer, and that each neuron in the kth hidden layer is connected to all output variables $y_1, \ldots, y_j$. Furthermore, the connections are all weighted such that the weight $w^k_{n,m}$ represents the connection between the nth neuron in the $(k-1)$th layer and the mth neuron in the kth layer. Each neuron in the hidden layer also has a bias, where b^k_n represents the bias for the nth neuron in the kth layer. [15]
The number of hidden layers, as well as the number of neurons in each hidden layer, is chosen when constructing the network. There is no strict rule for this; however, as the number of layers increase so does the complexity, computation time and risk of overfitting.

2.4.2 Training

When training the neural network, the goal is to minimize the *loss function*, i.e. the prediction error of the network. For regression purposes, it is common to use the *mean square error (MSE)*:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{2.24}$$

where n is the number of samples, y_i is the target output and $\hat{y}_i$ is the predicted output. The optimization of MSE uses *stochastic gradient descent (SGD).* This is a variant of gradient descent that, instead of calculating the gradient for the entire training data set, uses a small and randomly selected subset to approximate the gradient of the loss function. The number of training samples used in the subset is specified by the *batch size.* A small batch size gives a high frequency of parameter updates, with the smallest possible batch size being one (i.e. a subset of just one

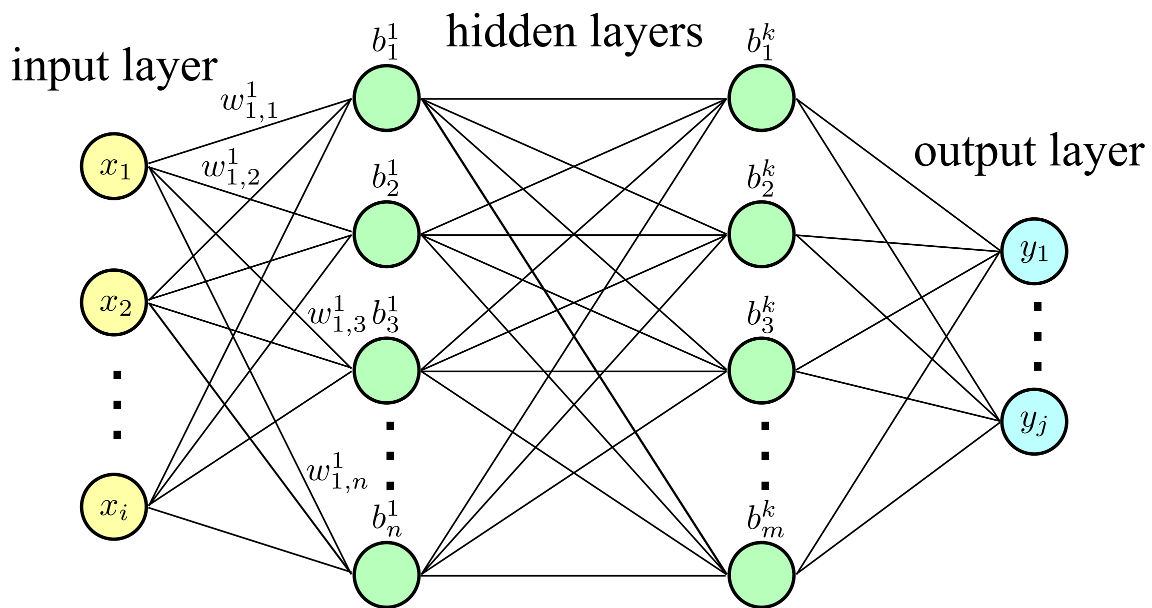


Figure 2.7: Schematic illustration of a neural network. The number of neurons i in the input layer represent the number of input variables, and the number of neurons j in the output layer represent the number of output variables. The number of hidden layers k and the number of neurons in these layers (which can vary between layers – note that the number of neurons in hidden layer 1 is n but hidden layer k has m neurons), are chosen when building the neural network. Each neuron in the hidden layers has a bias b, see equation (2.23). To save space in the figure, only the weights w, see equation (2.23), for the first neuron (which is connected to the feature component x_1) in the first hidden layer are shown.

sample). If the batch size is equal to the number of samples in the training data set, we get the regular gradient descent (see [16] for a description of this method). [17] When taking a step in SGD, the weights and biases of the neurons are updated from left to right according to

$$w_{i+1} = w_i - \eta \frac{\partial L}{\partial w}(w, b) \tag{2.25}$$

$$b_{i+1} = b_i - \eta \frac{\partial L}{\partial b}(w, b), \tag{2.26}$$

where η is the *learning rate* or *step size* of SGD, and L is the loss function (2.24). [18] This is called a *feed-forward* structure. In contrast, when an iteration through the network is done, the errors are updated from right to left before the next iteration. This process – updating the neurons forward and the errors backward – is called *backpropagation* and is used to decrease the loss of the network predictions. The network is trained for a specified number of iterations (or *epochs*) that is empirically decided upon. [15]

3

Underlying Technologies of Modelling Methods

This chapter provides information about the underlying technologies used when implementing the modelling methods – described in Chapter 2 – in Python. Section 3.1 explains how GPR is implemented using `scikit-learn`. The Python package `lime`, which is used to implement LIME, is explained in Section 3.2. Finally, Section 3.3 explains how the neural network is implemented using `PyTorch`.

3.1 Gaussian Process Regressor

This section explains how GPR (see Section 2.2) is implemented in Python.

In order to perform Gaussian process regression in Python, an open source machine learning library called `scikit-learn` is used. More specifically, it has a class called `GaussianProcessRegressor` that implements GPR. The implementation is based on Algorithm 3.1 (Algorithm 2.1 in [2]). [4]

1: **input**: X (inputs), $\boldsymbol{y}$ (targets), k (kernel), σ^2 (noise level), $\boldsymbol{x}_*$ (test input)
2: $L := \text{cholesky}(K(X, X) + \sigma^2 I)$
3: $\boldsymbol{\alpha} := L^\top \setminus (L \setminus y)$
4: $\bar{f}_* := \boldsymbol{k(x_*)}^\top \boldsymbol{\alpha}$
5: $\boldsymbol{v} := L \setminus \boldsymbol{k(x_*)}$
6: $\mathbb{V}[f_*] := k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{v}^\top \boldsymbol{v}$
7: $\log p(\boldsymbol{y}|X) := -\frac{1}{2}\boldsymbol{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$
8: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\boldsymbol{y}|X)$ (log marginal likelihood)

Algorithm 3.1: The algorithm implemented by the `scikit-learn` class `GaussianProgressRegressor`. Line 3 and 4 calculate the predictive mean – compare to equation (2.17), line 5 and 6 calculate the predictive variance – compare to equation (2.18), and line 7 calculates the log marginal likelihood. As seen in line 2, the implementation uses Cholesky factorization (see Appendix A.4 in [2]) for matrix inversion.

The `GaussianProcessRegressor` assumes that the prior has a constant mean of zero, as in equation (2.7), unless otherwise specified. To specify what kernel (see Section 2.2) to use in the `GaussianProcessRegressor`, a `kernel` parameter is passed to it. The `kernel` parameter includes what kernel(s) to use, as well as the initial

hyperparameter values. The hyperparameters of the kernel is set via a parameter called `theta` (an `ndarray` of shape `(n_dims,)`). Depending on if the kernel used is isotropic or anisotropic (see Section 2.2.2), `theta` might have different values for different dimensions. When calling the method `fit`, the hyperparameters are then optimized using gradient ascent to maximize the log-marginal-likelihood. For the Matérn kernel (see Section 2.2.2), the hyperparameter `nu` is not optimized but fixed to the initial value that is set when defining the `kernel` instance. Values of `nu` that are not in `[0.5, 1.5, 2.5, inf]` will increase the computational cost heavily; hence, it is desireable to have `nu` fixed to one of these values. [4] Furthermore, as mentioned in Section 2.2.2, the ν hyperparameter specifies the smoothness of the function (i.e. how any times it is differentiable). Optimizing `nu` would therefore change this, which is not desireable.
Returning to the issue, mentioned in Section 2.2.2, that the marginal likelihood (or log-marginal-likelihood) function of the hyperparameters might have several local maxima. To avoid accidentally finding a local maximum that is not global, it can be necessary to restart the optimization process a few times with random initial values of the hyperparameters to ensure that the found maximum is global. Thus, `GaussianProcessRegressor` has a parameter called `n_restarts_optimizer` where it is possible to specify how many times to restart the optimization process. In the first run, the initial hyperparameter values are the ones given by `theta` (or default if none are given); however, in the following runs the initial values are sampled log-uniform randomly from the space of allowed hyperparameter values. [4]
If there is noise in the data, the parameter `alpha` can be passed. This parameter represents the (estimated) variance of the Gaussian noise of the training data measurements, i.e. σ^2 in equation (2.13). Depending on if the noise level is the same for all data, `alpha` can be a `float` or – if the noise level varies – an `ndarray` of shape `(n_samples,)`. [4]
When a model has been fitted to the data, it is possible to make predictions for new input data by calling the method `predict`. It returns the mean of the predictive distribution at the query points, i.e. $\bar{\boldsymbol{f}}_*$ in equation (2.17). If the parameter `return_std` is set to `True`, the standard-deviation of the predictive distribution at the query points – $\sqrt{\text{cov}(\boldsymbol{f}_*)}$ from equation (2.18) – is returned as well. The `predict` function requires that the kernel function is evaluated in all prediction points, which can be time-consuming. [4]

3.2 Local Interpretable Model-Agnostic Explainer

This section explains how LIME (see Section 2.3) is implemented in Python.

To implement the LIME model, an open source Python package called `lime` is used. More specifically, the class `LimeTabularExplainer` is used since the data to be explained is in tabular, or matrix, format (as opposed to e.g. images or text). The training data is passed as a parameter to the `LimeTabularExplainer`, as well as a parameter specifying the `mode` – `'regression'` or `'classification'`. The features are perturbed (see Section 2.3) by sampling from a $\mathcal{N}(0, 1)$ distribution and performing inverse mean-centering and scaling according to the means and standard

deviations of the training data. [19]
When calling the method `explain_instance` – with parameters `data_row` and `predict_fn` – predictions are generated for a prediction instance. The parameters `data_row` and `predict_fn` is the data row of interest and the predict function used by the black box model, respectively. The explanation object returned can be represented by e.g. a list or a plot of the features and their corresponding correlation to the prediction, see table 4.3 for an example. [19]

3.3 Neural Network

This section describes how the neural network (see Section 2.4) is defined and trained in Python.

The neural network is implemented using the open source machine learning library `PyTorch` and, in particular, its `torch` package. The `torch` package in turn has a package called `nn` whose class `nn.Module` is used for defining the neural network as a class. When defining the `Network` class, the number of layers and neurons in each layer has to be specified (see Section 2.4.1). The layers of the neural network used in this project are *fully connected* (i.e. each neuron in one layer is connected to all neurons in the following layer) and defined as `nn.Linear(in_features, out_features)`, where `in_features` is the number of features in the previous layer (input layer if it is the first layer) and `out_features` is the number of features in the following layer (output layer of it is the last layer). As mentioned in Section 2.4.1, the input and output layer of the entire model have the same number of features as the number of input and output features, respectively, of the model. However, for the hidden layers, the number of input and output features are chosen when defining the `Network`. The class `nn.Linear` applies a linear transformation to the incoming data, i.e.

$$\boldsymbol{y} = \boldsymbol{x}A^T + \boldsymbol{b}. \tag{3.1}$$

The weights in A and biases in $\boldsymbol{b}$ are initialized from a $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ distribution, where $k = \frac{1}{n}$ and n is the number of input features. [20]
Another step in defining the `Network` is specifying how to move forward through the neural network. This is done by defining a `forward` function, where the activation function (see Section 2.4.1) for each layer is defined. For this project, the ReLU function – equation (2.22) – is used by calling `nn.functional.relu`. [20]
When the `Network` is properly defined, we can create an instance of it and begin training using the `torch` package `optim`. It has a class called `optim.SGD` that implements SGD (see Section 2.4.2). Two parameters of `optim.SGD` are `params` – specifying which parameters to optimize, and `lr` – specifying the learning rate (or step size) of the optimizer. Apart from creating an instance of `optim.SGD`, a loss function (see Section 2.4.2) also has to be specified. In this project, this is done using the class `nn.MSEloss`, which uses mean squared error – equation (2.24) – for loss calculation. [20]
When training the `Network`, a `for` loop is defined where – for each epoch – the steps in Algorithm 3.2 are performed.

```
1: foreach epoch do:
2:     set all gradients to 0                    ▷ to avoid accumulation of gradients
3:     make predictions from Network
4:     calculate loss using nn.MSEloss
5:     calculate gradients
6:     take one step (update parameters) in optim.SGD
```

Algorithm 3.2: Algorithm describing the training process of a neural network. The `for` loop is run for a pre-defined number of epochs.

When the specified number of epochs have been run through, the `Network` is trained and ready to make predictions.

4

Methodology

This chapter provides a description of the different steps in the methodology. First, an overview description of the model is given in Section 4.1. Section 4.2 describes how the model was built using GPR and LIME. The visualization of the resulting model is described in Section 4.3. Finally, Section 4.4 gives a description of how the prediction runtime of the model was optimized using a neural network.

4.1 Description of Model

This section gives an overview of the structure of the programs, in order to make subsequent sections easier to follow. Figure 4.1 shows a flowchart of the program structure. Section 4.1.1 describes the GPR model (shown to the left in the figure) and Section 4.1.2 describes the neural network model (shown to the right in the figure). The processor that the programs are run on is a AMD Ryzen 7 PRO 4750U with Radeon Graphics 1.70 GHz and 32.0 GB RAM.

4.1.1 Gaussian Process Regression Model

The flowchart of the program for the GPR model – described in Section 4.2 – is shown to the left in Figure 4.1. Measurement data in the `engine_data.mat` file is loaded into the program, along with extreme points in the `extreme_points.csv` file. The latter (further described in Section 4.2) contains fingered ”measurement data” from a separate extrapolation program with the purpose of improving the extrapolation of the GPR model. Next, the data is preprocessed: the features of interest (n, p_i, ϕ_i, ϕ_e and η_V) are extracted from the measurement data, the data is split up into training, testing and validation data sets, and the extreme points are added to the training data set. These three data sets (`train_df`, `test_df` and `rest_df`) are then used in the GPR model.
The data is normalized (see sections 3.1 and 4.2) by calculating the `numpy.mean` and `numpy.std` of `train_df`, and then using this to normalize all three data sets using

$$X_\mathrm{norm} = \frac{X - \hat{\mu}}{\hat{\sigma}}, \tag{4.1}$$

where $\hat{\mu}$ and $\hat{\sigma}$ are estimations of the mean and standard deviation, respectively. Furthermore, the data sets are split into input data $X = (n, p_\mathrm{i}, \phi_\mathrm{i}, \phi_\mathrm{e})$ and output data $y = \eta_\mathrm{V}$. The `kernel` and `alpha` parameters (see Section 3.1) are set, as well as `n_restarts_optimizer` specifying how many times to restart the hyperparameter

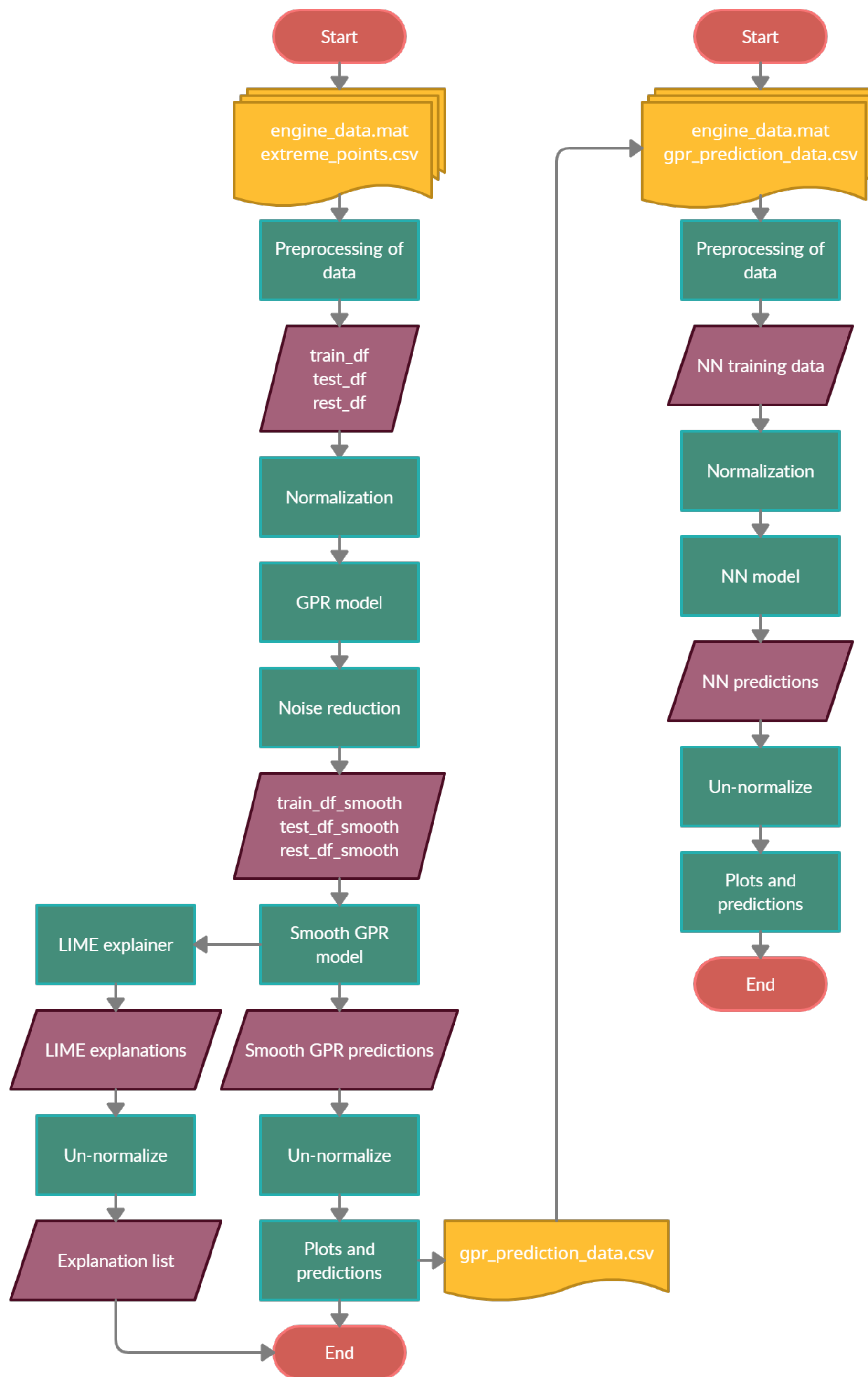


Figure 4.1: Flowchart of the program for the GPR/LIME model (to the left in the figure) and the program for the neural network model (to the right in the figure), as well as how they are connected. The red ovals mark the Start and End of each program. The yellow shapes with a curved bottom represent one or several files being loaded into – or given as output from – the programs. The green rectangles represent processes in the program and the purple parallelograms represent outputs from – and inputs to – these processes.

optimization. The GPR model is then initiated and trained using the code snippets below.

```
gpr = GaussianProcessRegressor(
    kernel = kernel,
    alpha = alpha,
    n_restarts_optimizer = n_restarts_optimizer
    )

gpr.fit(
    X_train,
    y_train
    )
```

The tuning parameters of this model are listed below (and further described in Section 3.1).

- The noise level parameters `alpha` and `alpha_smooth`.
- The confidence interval percentage to be used in the noise reduction process (see Section 4.2).
- The hyperparameter ν of the Matérn kernel.
- The bounds of the length-scale hyperparameter ℓ of the Matérn kernel.
- The number of times to restart the optimization of the log-likelihood of the hyperparameters.

In the noise reduction process, predictions from the `gpr` instance are made using

```
y_train_pred, sigma_train = gpr.predict(
    X_train,
    return_std = True
    )
```

where the returned standard deviation `sigma_train` is used to calculate the confidence interval to be used in the noise reduction process (see Section 4.2). Data points outside the confidence interval are given the property `train_df['outlier']==True`, and the points with `train_df['outlier']==False` are stored in a new `DataFrame` called `train_df_smooth`. The same is done for `test_df` and `rest_df`.
In the smooth GPR model, the initiation and training is done as above, however with `alpha_smooth` and `X_train_smooth`. Moreover, the `kernel` parameter is set to the one found in the first GPR model – making it possible to set `n_restarts_optimizer` to zero (see Sections 3.1 and 4.2).
When the smooth GPR model is trained, predictions are made using the manually defined function `gpr_smooth_pred` (see Section 4.2), that uses `gpr_smooth.predict` but sets all $\eta_V < 0$ to 0.[1] The data is then un-normalized in order to make sense of plots and comparisons by reversing equation (4.1) for the input and output data, and by multiplying the returned standard deviaton by `numpy.std(y_train)`. The function `get_gpr_variance` is used to calculate the variance as `sigma ** 2`, and points with high variance (see Section 4.2 for a description of the condition for this)

[1]Since η_V is normalized in this stage of the program, 0 is actually $(0 - \hat{\mu})/\hat{\sigma}$.

are given the property `df['high_variance']==True`.
Concerning regions of high variance, the `lime` explainer (see Section 3.2) is initiated as

```
explainer = lime_tabular.LimeTabularExplainer(
    training_data = numpy.array(X_train),
    mode = 'regression',
    feature_names = X_train.columns
    )
```

and then used for explaining prediction instances with high variance from

```
exp = explainer.explain_instance(
    data_row = df[['EngSpd', 'p_Man', 'ICPC', 'ECPC']],
    predict_fn = get_gpr_variance
    )

high_variance_exp_df = pandas.DataFrame(
    exp.as_list(),
    columns = ['variables', 'variance_correlation']
    )
```

where an example output is shown in table 4.3. The model is then tested on the test and validation data (both `test_df` and `test_df_smooth`) and plots such as Figure 4.3 and 4.4 are generated. The predictions made when making the plots are stored in a file called `gpr_prediction_data.csv` to be used in the neural network program (see Section 4.1.2).

4.1.2 Neural Network Model

The flowchart of the program for the neural network model – described in Section 4.4 – is shown to the right in Figure 4.1. Prediction data from the GPR model is loaded into the program through the `gpr_predicton_data.csv` file. The data sets are prepared similarly as in Section 4.1.1, i.e. split into `train_df`, `test_df` and `rest_df`, normalized using equation (4.1) – with $\hat{\mu}$ and $\hat{\sigma}$ being calculated as `numpy.mean` and `numpy.std` of `train_df`, and split into input data $X = (n, p_{\text{i}}, \phi_{\text{i}}, \phi_{\text{e}})$ and output data $y = \eta_{\text{V}}$.
The `Network` class is defined as

```
class Network(nn.Module):

    def __init__(self, n_input_features):
        super().__init__()

        self.fc1 = nn.Linear(n_input_features, n_hidden_neurons)
        self.fc2 = nn.Linear(n_hidden_neurons, n_hidden_neurons)
        self.fc3 = nn.Linear(n_hidden_neurons, 1)

    def forward(self, x):
```

```
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

where the first function is the definition of the network structure (number of hidden layers and number of neurons in these layers, see Sections 2.4.1 and 3.3) and the second function defines how to move forward in the network (here we are using the ReLU function, see equation (2.22) and Section 3.3). The tuning parameters of this model are listed below (and further described in Section 2.4).

- The number of hidden layers and the number of neurons in each hidden layer (note that these do not have to be the same in each layer).
- Learning rate/step size, i.e. how large steps are taken in SGD.
- Batch size, i.e. how much data to use when approximating the gradients in SGD.
- The number of epochs/iterations to go through when training the network.

In order to use the data for training and testing in the neural network, the data is converted to `torch` objects. After initiating an instance of `Network` as

```
my_net = Network(n_input_features = n_input_features)
```

where `n_input_features = 4` in this case ($n, p_{\mathrm{i}}, \phi_{\mathrm{i}}, \phi_{\mathrm{e}}$), the SGD optimizer and MSE loss function (see Sections 2.4.2 and 3.3) are set using the code snippets below.

```
opt = optim.SGD(
    my_net.parameters(),
    lr = learning_rate
    )

loss_fn = nn.MSELoss()
```

The neural network `my_net` is trained for the specified number of epochs as described in Section 2.4. For each epoch, the loss is calculated for both `y_train` and `y_test` and saved in an array in order to compare these losses to ensure that the training actually improves the network and that there is no overfitting. Figure 4.6 shows the loss development over epochs for both training and testing data of the neural network model.

When the neural network is trained, predictions are made using the function `nn_pred` (see Section 4.4) that predicts from `my_net` but sets all $\eta_{\mathrm{V}} < 0$ to 0.[2] The data is then un-normalized in order to make sense of plots and comparisons by reversing equation (4.1) for the input and output data.

Finally, the model is tested on the test and validation data – as well as the measurement data used in the GPR model – and plots such as Figure 4.7 and 4.8 are generated.

[2]Since η_{V} is normalized in this stage of the program, 0 is actually $(0 - \hat{\mu})/\hat{\sigma}$.

Index	`EngSpd`	`p_Man`	`ICPC`	`ECPC`	`eta_vol`
1	999.39	52.40	1.02	4.65	0.938197
2	999.52	51.76	10.00	4.63	0.948425
⋮	⋮	⋮	⋮	⋮	⋮
6936	5500.19	78.08	10.01	39.97	0.967746

Table 4.1: Example of `DataFrame` containing measurement data for engine speed n (`EngSpd`), intake manifold pressure p_i (`p_Man`), intake cam phaser ϕ_i (`ICPC`), exhaust cam phaser ϕ_i (`ECPC`), and volumetric efficiency η_V (`eta_vol`). These quantities are further described in Section 2.1. The data is scaled for NDA reasons.

4.2 Building the Model

This section explains how the model described in Section 4.1.1 was built using GPR and LIME in Python.

The final model is made for multiple dimensions; however, initially the model was built with a one dimensional input space to ensure that it was working as desired. When the results were satisfying, the dimensions of the input space were increased stepwise up to the final four dimensions. The measurement data, provided by T-Engineering, for volumetric efficiency η_V was loaded into Python and stored in a `pandas DataFrame` (see table 4.1 for an example). It was then split into a training data set (`train_df` with 72% of the data), a testing data set (`test_df` 18% of the data) and a validation data set (`rest_df` with 10% of the data) to be used for a final comparison. This is to ensure that the model is not overfitting. The proportion of data in the different sets was decided empirically with the purpose of keeping training time low without losing model accuracy (see Section 6.1 for a discussion on this matter). The training data set was split into input data $X = (n, p_\mathrm{i}, \phi_\mathrm{i}, \phi_\mathrm{e})$ and output data $y = \eta_\mathrm{V}$, and then normalized in order to better match the assumed zero mean prior (see Section 3.1). For the first part of the model, an instance of `GaussianProcessRegressor` (see Section 3.1) was created. Since initially it can be difficult to know which kernel is best suited for the model, a list of simple and complex kernels (created by adding and multiplying the kernels presented in Section 2.2.2) was set up and run through in a `for` loop. After comparison, a Matérn kernel with $\nu = 2.5$ – equation (2.19) – was deemed the best suited kernel for this problem since it both fitted the data well and was reasonably fast in training and predicting. The `n_restarts_optimizer` parameter was empirically chosen to be 5, to avoid accidentally finding a local maximum that is not global (see Sections 2.2.2 and 3.1). Since the optimization is restarted `n_restarts_optimizer` times during training, a high value for this parameter increases the training time; hence, 5 was deemed a suitable value to ensure that a global maximum was found while keeping training time reasonably low. The `alpha` parameter, describing the noise level of the data, was set by calculating the standard error of `train_df['eta_vol']` and then multiplying it by 5 (a number that was empirically decided upon).

In order to perform noise reduction, after the first GPR fit, the data inside a certain confidence interval was kept in a new `DataFrame` (`train_df_smooth`, etc.) which

was then used for a second fit. To calculate the confidence interval, the returned standard deviation from the `fit` method (see Section 3.1) was used. For the second fit, a new instance of `GaussianProcessRegressor` was created, with initial kernel hyperparameters being the optimized ones from the first model. This way, it was possible to reduce the `n_restarts_optimizer` to 0, since the first fit had already come close to the global maximum of the log-marginal likelihood. The second instance of `GaussianProcessRegressor` also had a smaller value for `alpha` called `alpha_smooth` – twice the standard error, since the filtered data had a lower noise level. Since volumetric efficiency η_V is always ≥ 0, a function called `gpr_smooth_pred` was defined that uses the `predict` method from the second instance of `GaussianProcessRegressor`, but sets all values < 0 to 0.

Even before the visualization part of the project (see Section 4.3) began, some visualization had to be done in order to properly evaluate the model. This was done using `matplotlib` to make 3D surface plots for fixed values of the cam phasers ϕ_i and ϕ_e, as well as a 2D plot where the engine speed n was also fixed. Figure 4.2 shows a similar 2D plot for the current model at T-Engineering, where $\phi_\mathrm{i} = 10$, $\phi_\mathrm{e} = 10$ and $n = 3500$ RPM. It shows an anomaly at $p_\mathrm{i} \approx 150\,\mathrm{kPa}$ that has no reasonable physical explanation but is not captured as an outlier by the model. The combination of noise level parameters (`alpha` and `alpha_smooth`) and confidence interval in the noise reduction process for the GPR model thus have to capture this properly. After some evaluations, a 95% confidence interval was deemed suitable for noise reduction. The major property to look for in the 3D plots was if the surfaces were smooth or dented, and then find a suitable combination of initial noise level parameter `alpha` and smooth noise level parameter `alpha_smooth` that gave smooth enough surfaces. In addition to this, the lower bound for the `length_scale` parameter of the kernel was set to 1.5 to ensure smooth surfaces despite high noise in the data. See Section 6.3 for a discussion on this matter.

As mentioned in Section 2.2.1, GPR does not perform very well with extrapolation. In order to make the model more reliable at the ends as well, a few extreme points were added manually. These extreme points were found by modelling $\eta_\mathrm{V} \cdot p_\mathrm{i}$ as a function of p_i. This relationship is nearly linear and therefore easier to extrapolate. By fixing and looping through different values of ϕ_i, ϕ_e and n, and saving the intersection values for $p_\mathrm{i} = 300$ (the upper end of the p_i interval) and $\eta_\mathrm{V} = 0$ (the lower end of the η_V interval) to a .csv file, the extreme points could then be loaded to the main model and added to `train_df`. Since there was lack of measurement data in the vicinity of the added extreme points, some of these were initially filtered out in the noise reduction process – thus losing their impact on the final model. To ensure that this would not happen, the manually added extreme points were given a lower value for the `alpha` parameter (see Section 3.1), which made the model adapt more to these points.

The optimized values for the `length_scale` hyperparameter, for both the initial and smooth GPR model, are presented in table 4.2. The lower bound of 1.5 for the `length_scale` parameter affected the length scale for intake manifold pressure p_i.

When making the 3D plots mentioned above, a mesh grid with values for engine speed n, intake manifold pressure p_i, and cam phasers ϕ_i and ϕ_e, was created and the model was evaluated at all points. Using a manually defined function called

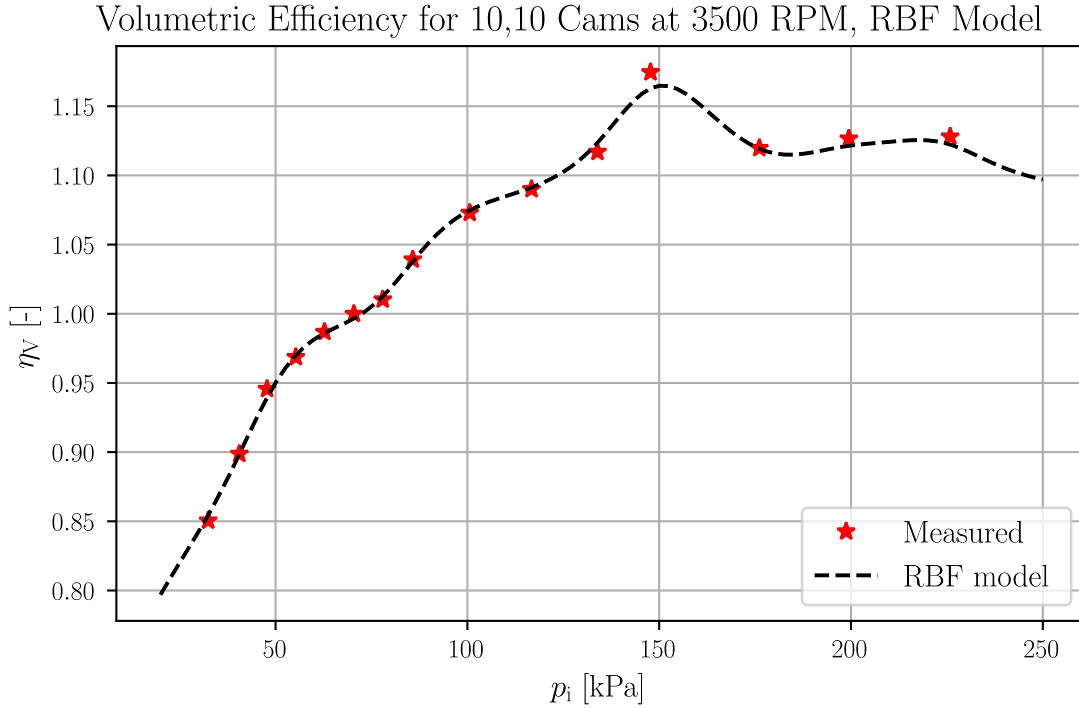


Figure 4.2: 2D plot of radial basis function interpolation model (currently used at T-Engineering), for $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The anomaly at $p_i \approx 150$ kPa has no reasonable physical explanation and thus has to be captured as an outlier by the GPR model. The data is scaled for NDA reasons.

Length scale ℓ	Initial model	Smooth model
Engine speed n	2.79	4.10
Intake manifold pressure p_i	1.50	1.50
Intake cam phaser ϕ_i	4.44	4.66
Exhaust cam phaser ϕ_e	4.36	5.23

Table 4.2: The optimized values for the length scale hyperparameter of the Matérn kernel for the four input features (n, p_i, ϕ_i, ϕ_e). The "Initial model" column represents the values for the first step in the GPR model, and the "Smooth model" column represents the values for the smooth GPR model – after noise reduction was performed. The lower length scale bound was set to 1.5, which affected the length scale for intake manifold pressure p_i.

variables	variance_correlation
ICPC > 40.00	0.000211009
ECPC > 39.99	0.00015933
EngSpd > 3494.32	0.000113644
90.76 < p_Man <= 134.44	-3.52714e-05

Table 4.3: Example of output from the LIME explainer for a point/data row with high variance. For this data row, the highest contributing feature (they are presented in descending order) is `ICPC > 40.00`, i.e. that the intake cam phaser ϕ_i is set to a high crank angle. Since this explanation has the highest variance correlation, it will be stored in the `explanations` list. Notice that `p_Man` has a negative variance correlation, meaning that the intake manifold pressure p_i is in an interval where the variance of the model decreases.

`get_gpr_variance`, the variance at each point could also be returned. Hence, it was possible to evaluate which points had high variance – based on a pre-defined condition – and put these in a separate `DataFrame`. For this model, points that had variance $>$ `alpha_smooth` (i.e. greater than the estimated σ^2 of the training data, see Section 3.1) were considered to have high variance. However, to make the model more user-friendly, it is better to know which *regions* have high variance, instead of listing every point where the variance is high. In order to find these regions, a LIME explainer (see Sections 2.3 and 3.2) was used. The `get_gpr_variance` function was used as its predict function, which made it possible to get explanations for which features contributed to increasing the variance. Table 4.3 shows an example output from the LIME explainer. The output for a `data_row` was stored in a `DataFrame`, and these were in turn stored in another `DataFrame` called `high_variance_exp_dfs`. The major contributing feature from each `DataFrame` in the `high_variance_exp_dfs DataFrame` (e.g. `ICPC` in the table 4.3 example), as well as its explanation, was stored in a list called `explanations`. This list was then searched through for unique values in order to get an overview of which regions have high variance.

4.3 Visualization

This section describes the visualization part of the project, focusing mainly on the interactive visualization.

The key objective for the visualization part of the project is for it to be as pedagogical as possible. It should be easy to understand the model predictions, despite the four-dimensional input space. As mentioned in Section 4.2, visualization had to be taken into consideration quite early on in the project. Hence, it built upon the way that the current model at T-Engineering is visualized. The 2D plots rendered by the model (shown in Figure 4.3) have a similar design to the one in Figure 4.2; however, the confidence interval of the model is shown as the gray area, and the markers are different depending on if they were part of the training (blue star), testing (orange dot) or extrapolation (green star) data set. The choice to fix $\phi_\mathrm{i} = 10$, $\phi_\mathrm{e} = 10$ and

$n = 3500$ RPM is based on knowledge regarding where the engine performs well. In table 4.2, we can also see that p_i has the shortest length-scale and thus varies most within its interval.
When rendering the 3D plots, ϕ_i and ϕ_e were fixed at – and looped through – the values $[0, 10, 20, 30, 40, 50]$ respectively, thus generating 36 plots with p_i and n as variables. Again, table 4.2 shows that ϕ_i and ϕ_e have the longest length-scales, and thus vary less than p_i and n, which motivates the choice to fix these two parameters. Figure 4.4 shows an example of a 3D plot for $\phi_i = 0$ and $\phi_\mathrm{e} = 0$.
In addition to the figures presented in this report, an attempt to make the visualization more interactive was made using `widgets` in the web application Jupyter Notebook. The aim of this visualization is to show how the `alpha` parameters and confidence interval used in the noise reduction process (see Section 4.2) affect the model. In order to generate the plots necessary, the GPR model was built into a nested `for` loop, as presented in Algorithm 4.1. The model was run through for different values of `alpha`, `alpha_smooth` and `ci_percentage`, and for each time a 2D plot of the initial and smooth model was saved as an .svg image. These images were then loaded into a Jupyter Notebook program with sliders to control the values of `alpha`, `alpha_smooth` and `ci_percentage`. When the sliders are set to a certain combination of values, the corresponding plots are shown. Figure 4.5 shows an example of this visualization.

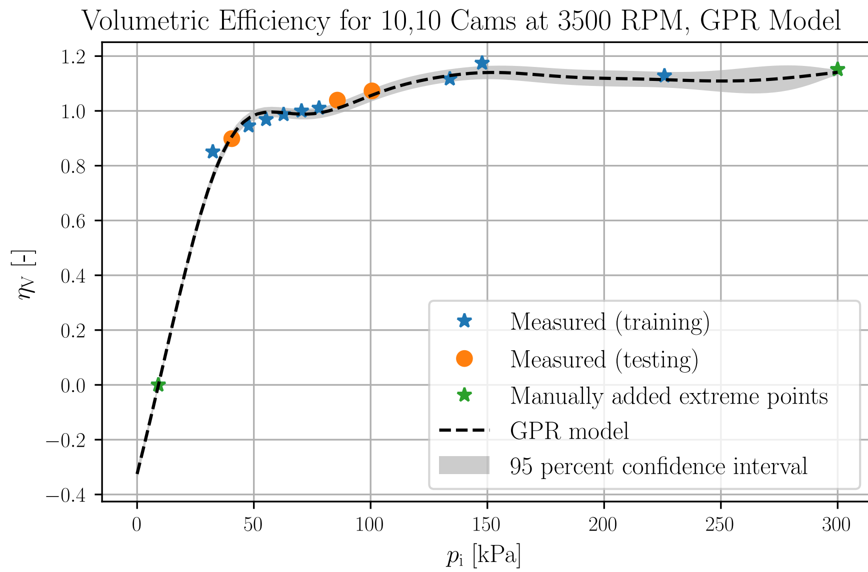
```
1: foreach alpha do:
2:     foreach alpha_smooth do:
3:         foreach ci_percentage do:
4:             GPR model
5:             save 2D plots to .svg file
```

Algorithm 4.1: Pseudo code describing the nested `for` loop used to create multiple plots for visualization in Jupyter Notebook.

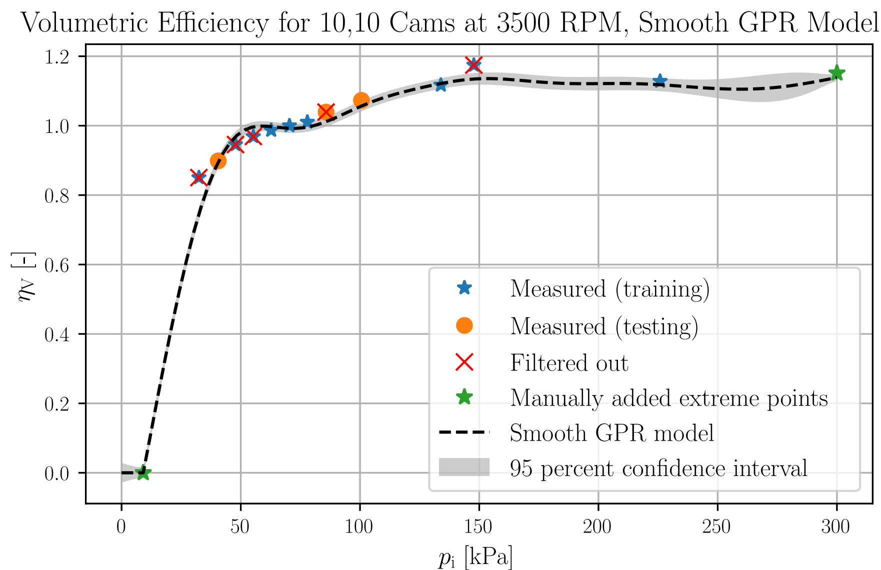
4.4 Optimizing Prediction Runtime of Model

This section explains how the model in Section 4.2 was adapted using a neural network in Python, in order to optimize the prediction runtime. The attained model is described in Section 4.1.2.

One of the key objectives (see Section 1.1) of this project is to build an efficient model that can operate in real time on a limited CPU. Thus, when the model was built, its prediction runtime had to be optimized. In order to do this, the model was approximated using a neural network. The model described in Section 4.2 was used to create lots of training data for the neural network (since they often require a lot of data for training), which was then saved to a .csv file and loaded into the neural network model. Similarly as in Section 4.2, the data was split into `train_df`, `test_df` and `rest_df`, normalized, and then split into input data $X = (n, p_\mathrm{i}, \phi_\mathrm{i}, \phi_\mathrm{e})$ and output data $y = \eta_\mathrm{V}$. An instance of `Network` (see Section 3.3) was defined with 2 hidden layers, each with 64 neurons. The network was trained for 200 epochs,



(a) 2D plot of initial GPR model for $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The data is scaled for NDA reasons.



(b) 2D plot of smooth GPR model for $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The data is scaled for NDA reasons.

Figure 4.3: 2D plots of volumetric efficiency η_V as a function of p_i for the initial and smooth GPR model. The other features are fixed at $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\times$. The fitted model is represented by the black dashed line and the gray shaded area represents the 95% confidence interval.

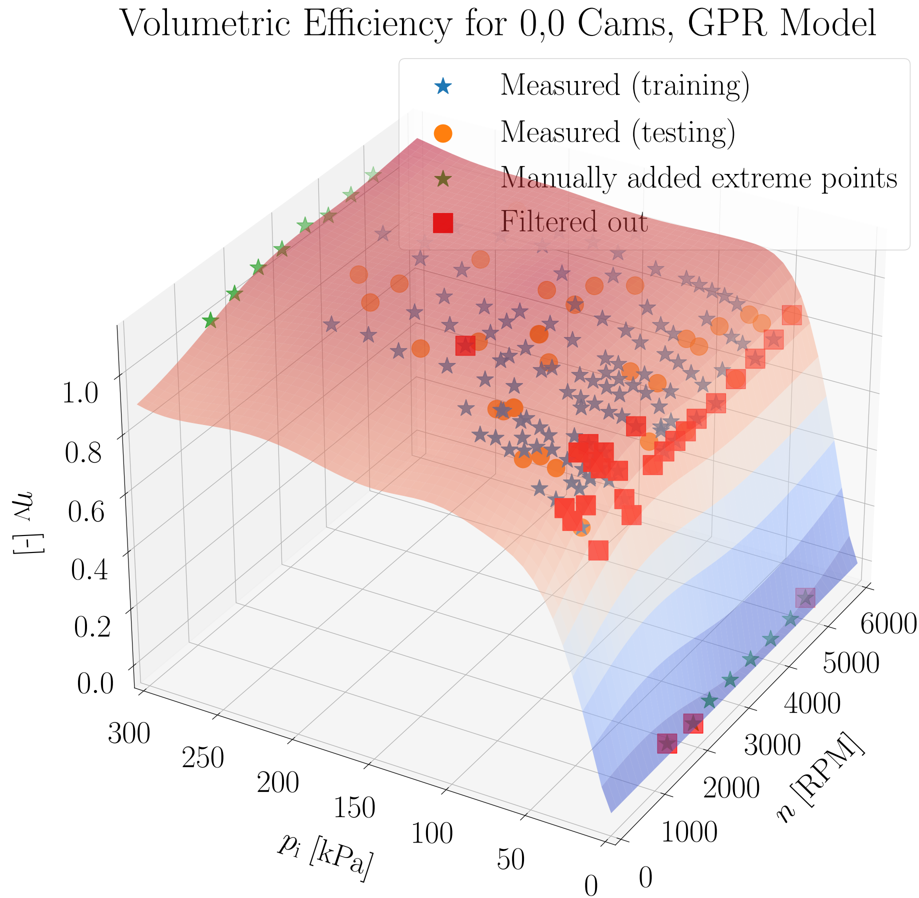
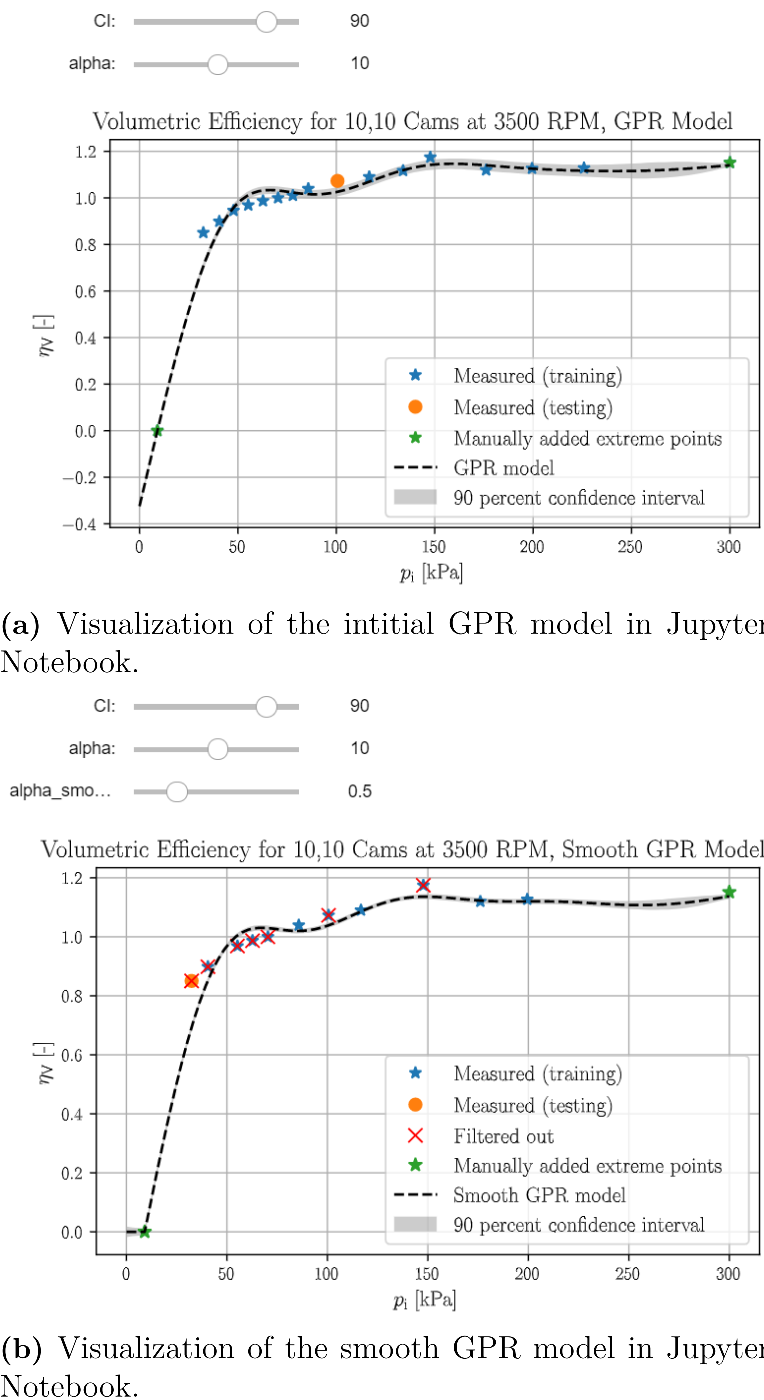


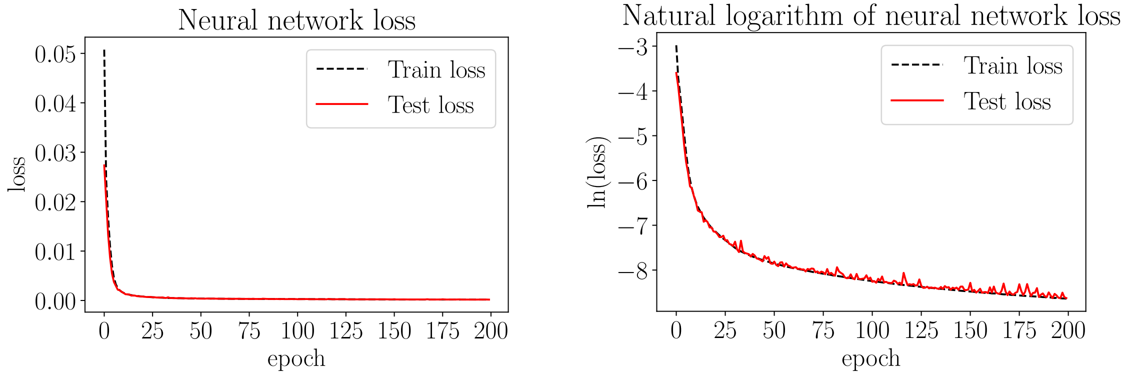
Figure 4.4: 3D plot of volumetric efficiency η_V as a function of p_i and n for the smooth GPR model. The cam phasers are fixed at $\phi_i = 0$ and $\phi_e = 0$. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the p_i interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\blacksquare$. The fitted model is represented by the surface area. Notice that the filtered out data points are mainly around $p_i \approx 50\,\text{kPa}$, which is where η_V starts decreasing more rapidly toward zero. Compare with Figure 5.2. The data is scaled for NDA reasons.



(a) Visualization of the intitial GPR model in Jupyter Notebook.

(b) Visualization of the smooth GPR model in Jupyter Notebook.

Figure 4.5: Example of the visualization of the GPR model using widgets in Jupyter Notebook. The sliders above the plots control the confidence interval (shown as the light blue area) used in the noise reduction process, as well as the parameters `alpha` and `alpha_smooth`. The plots show a 2D intersection of the GPR model, where the cam phasers are fixed to $\phi_i = 10$, $\phi_e = 10$, and the engine speed is fixed to $n = 3500$ RPM. When moving the sliders, the size of the confidence interval, as well as the shape of the model, will change. The plots are connected, so if the slider of the initial model is changed, so will one for the smooth model. Compare with Figure 4.3. The data is scaled for NDA reasons.



(a) Development of losses over epochs for the neural network. The black dashed line represents the loss for the training data and the red solid line represents the loss for the test data.

(b) Development of the natural logarithm of losses over epochs for the neural network. The black dashed line represents the loss for the training data and the red solid line represents the loss for the test data.

Figure 4.6: Development of losses – Figure (a) – and natural logarithm of losses – Figure (b) – over epochs for the training of the neural network. The losses are calculated using equation (2.24). Since the lines for training data and test data follow each other in both figures, we can assume that the model does not overfit.

and for each epoch the test data was evaluated as well in order to avoid overfitting. Figure 4.6 shows how the loss – equation (2.24) – and the natural logarithm thereof developed over the different epochs, for both `train_df` and `test_df`. Both lines follow the same pattern; hence, we can assume no overfitting.
When the network had been trained, predictions were made using a manually defined function called `nn_pred` that (as `gpr_smooth_pred` in Section 4.2) makes predictions from the network but sets all $\eta_{\mathrm{V}} < 0$ to 0. Then 2D and 3D plots were made similarly as in Section 4.2 and compared to the plots from the GPR model to ensure that there were no major differences for the neural network model – see Figure 4.7 and Figure 4.8. Furthermore, the measurement data that was used in the GPR model was also loaded into the neural network model and predictions for this data were made using the trained network. The predictions were compared with the measurements by calculating the absolute error.

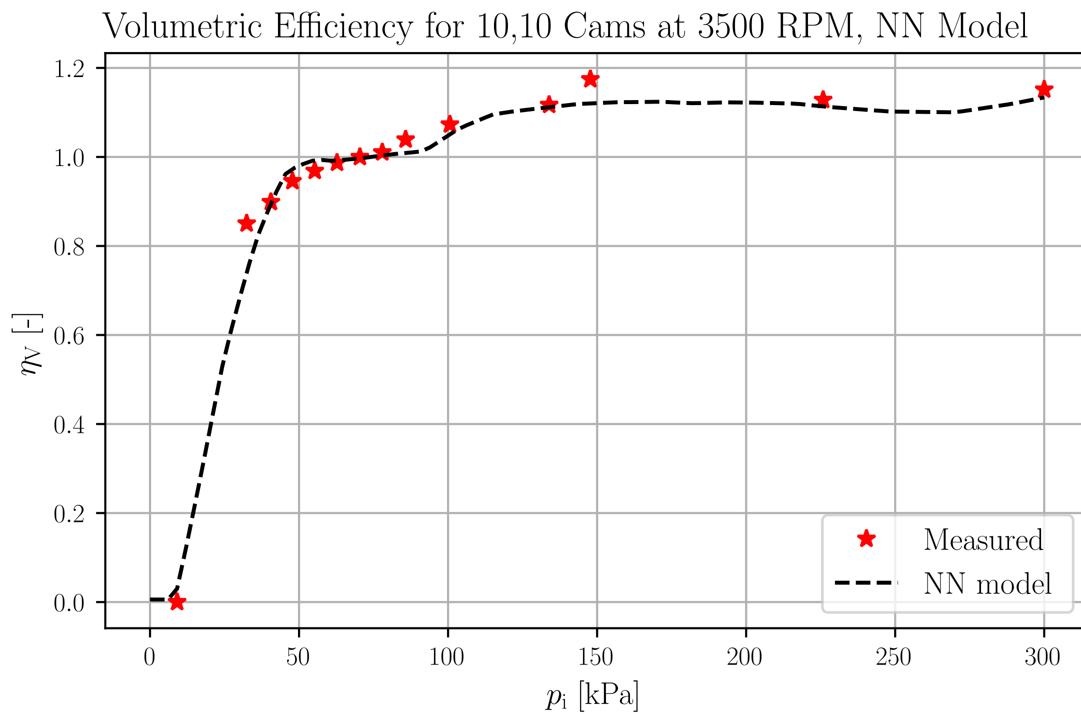


Figure 4.7: 2D plot of volumetric efficiency η_V as a function of p_i for the neural network model. The other features are fixed at $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The measured data points are marked with red $\star$, and the black dashed line represents the fitted model. Compare with Figure 4.3. The data is scaled for NDA reasons.

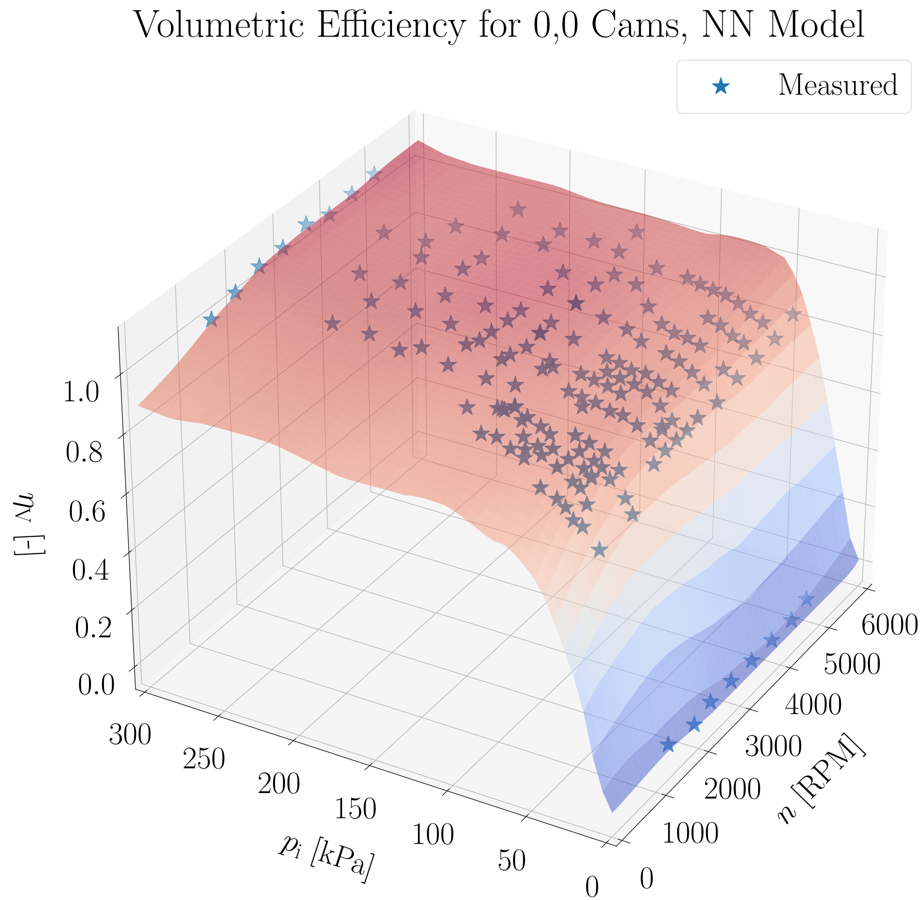


Figure 4.8: 3D plot of volumetric efficiency η_V as a function of p_i and n for the neural network model. The cam phasers are fixed at $\phi_\mathrm{i} = 0$ and $\phi_\mathrm{e} = 0$. The measured data points in are marked by $\star$, and the fitted model is represented by the surface area. Compare with Figure 4.4. The data is scaled for NDA reasons.

5
Results

This chapter presents the results obtained when evaluating the models described in Section 4.1.

In the description of the GPR model in Sections 4.1.1 and 4.2, the model actually consists of two GPR models – one initial and one smooth model, where the latter is used after the noise reduction process. However, since the final results are predicted from the smooth GPR model, the use of "GPR model" in this chapter will refer to the smooth model unless otherwise stated.

Figure 5.1 shows a 2D plot of η_V with respect to p_i for the GPR model (Section 4.1.1) – black solid line, the neural network model (Section 4.1.2) – red dashed line – and the RBF model – blue dotted line – currently used at T-Engineering. The engine speed is fixed at $n = 3500$ RPM and the cam phasers are fixed at $\phi_\mathrm{i} = 10$ and $\phi_\mathrm{e} = 10$. Compare to Figures 4.2, 4.3, and 4.7. The GPR model and the neural network model follow each other well, indicating that there were no major accuracy losses in the prediction runtime optimization of the model. In contrast to these models, the RBF model is more fluctuating and does not extrapolate in the same way – it does not reach $\eta_\mathrm{V} = 0$ within the p_i interval (since this model does not implement the manually added extreme points, as described in Section 4.2). See Section 6.1 for further comparisons between the GPR/neural network model and the RBF model.

3D plots from the GPR model and neural network model are shown in Figures 4.4 and 4.8. In these figures, the cam phasers ϕ_i and ϕ_e are both set to 0 which is a setting for which the engine runs well (see Section 2.1.4). The data points that are filtered out in the noise reduction process are mainly around $p_\mathrm{i} \approx 50\,\mathrm{kPa}$, which is where the slope of the surface is at its steepest. See Section 6.3 for a discussion on this matter. Figure 5.2 shows a 3D plot for the GPR model where the cam phasers are set to $\phi_\mathrm{i} = 50$ and $\phi_\mathrm{e} = 50$. A large proportion of the measured data for low p_i is removed in the noise reduction process. For high p_i, the volumetric efficiency is high when n is low.

The results from the LIME explainer (see Section 4.2) showed that the variance was high for `ICPC>40.05` and/or `ECPC>40.00`, i.e. that the model is uncertain for high cam phaser settings. This is further discussed in Section 6.1.

Table 5.1 lists the runtimes for predicting volumetric efficiency of 50 data points in the GPR model and the neural network model. This was calculated by making predictions for a total of 90 000 data points – 50 data points at a time – and calculating the mean runtime over the 1 800 runs. The predictions of the neural network model are over 800 times faster than the ones of the GPR model. Note, however, that the

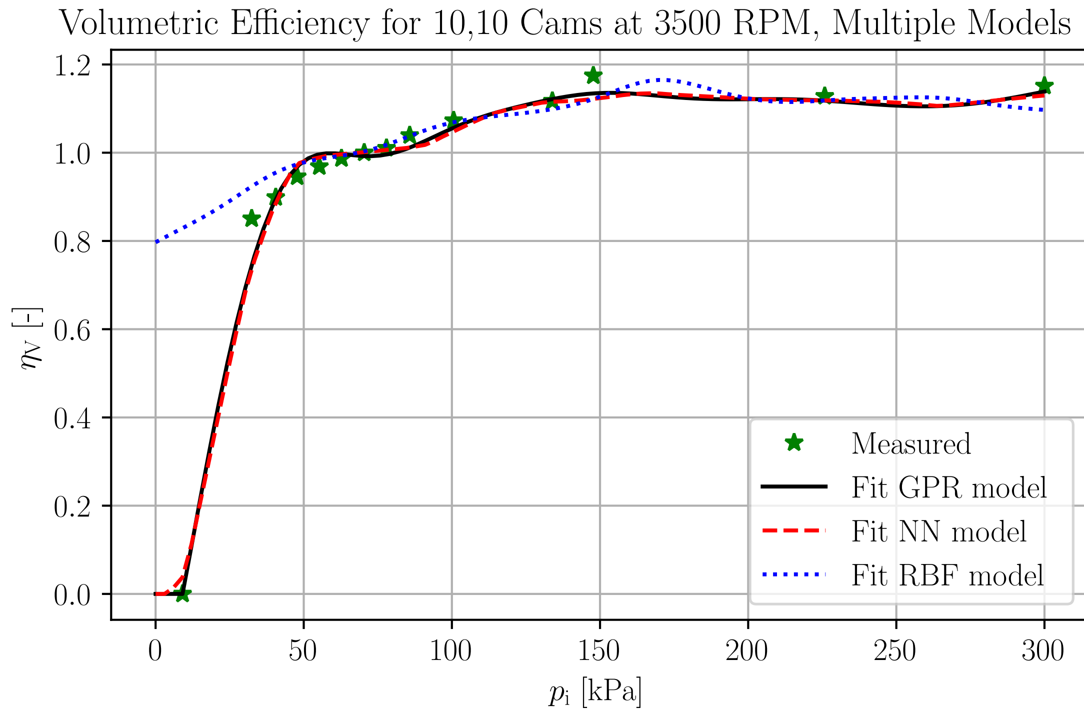


Figure 5.1: 2D plot of volumetric efficiency η_V with respect to p_i for the GPR model, the neural network model, and the RBF model. The other features are fixed at $\phi_i = 10$, $\phi_e = 10$, and $n = 3500$ RPM. The measurement data points are marked with green $\star$, the black solid line represents the fitted GPR model, the red dashed line represents the fitted neural network model, and the blue dotted line represents the fitted RBF model. The GPR and neural network model follow each other, whereas the RBF model fluctuates more and turns more slowly toward $\eta_V = 0$. Compare to Figures 4.2, 4.3, and 4.7. The data is scaled for NDA reasons.

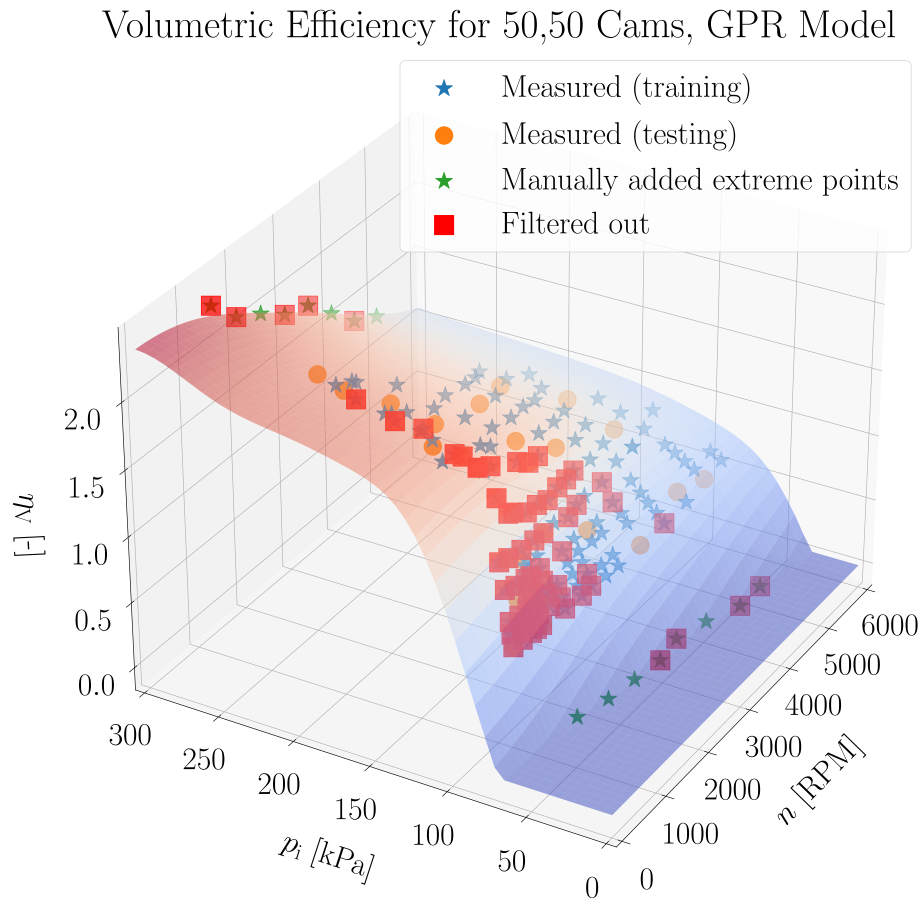
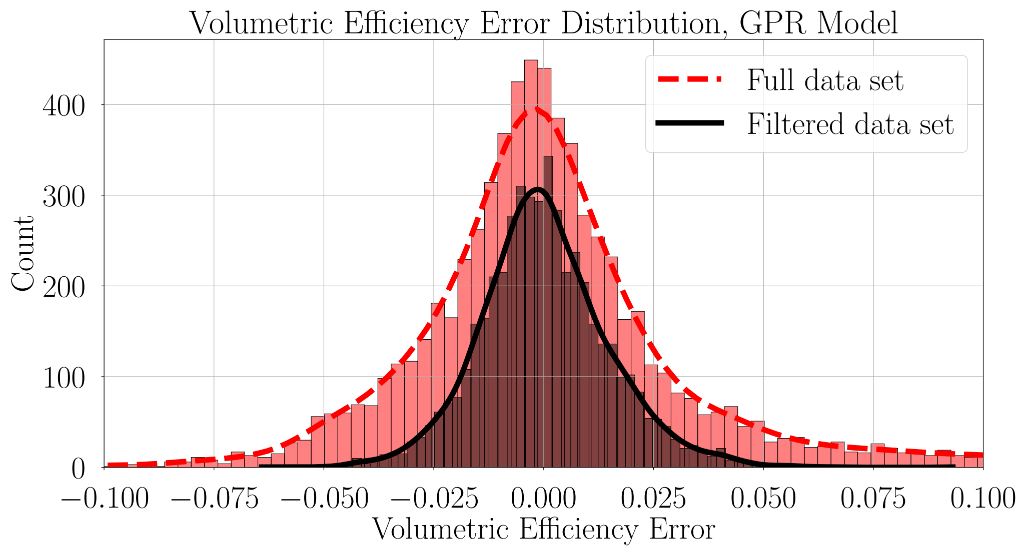


Figure 5.2: 3D plot of volumetric efficiency η_V as a function of p_i and n for the smooth GPR model. The cam phasers are fixed at $\phi_i = 50$ and $\phi_e = 50$. The data points in the training data set are marked by $\star$, where the blue stars represent measured data, and the green stars (at the ends of the p_i interval) represent the manually added extreme points. The orange $\bullet$ represent data points in the testing data set. Data points that are filtered out in the noise reduction process are marked with a red $\blacksquare$. The fitted model is represented by the surface area. Notice that the filtered out data points are mainly at low p_i and low n, and that η_V is high for high values of p_i and low n. Compare with Figure 4.4. The data is scaled for NDA reasons.

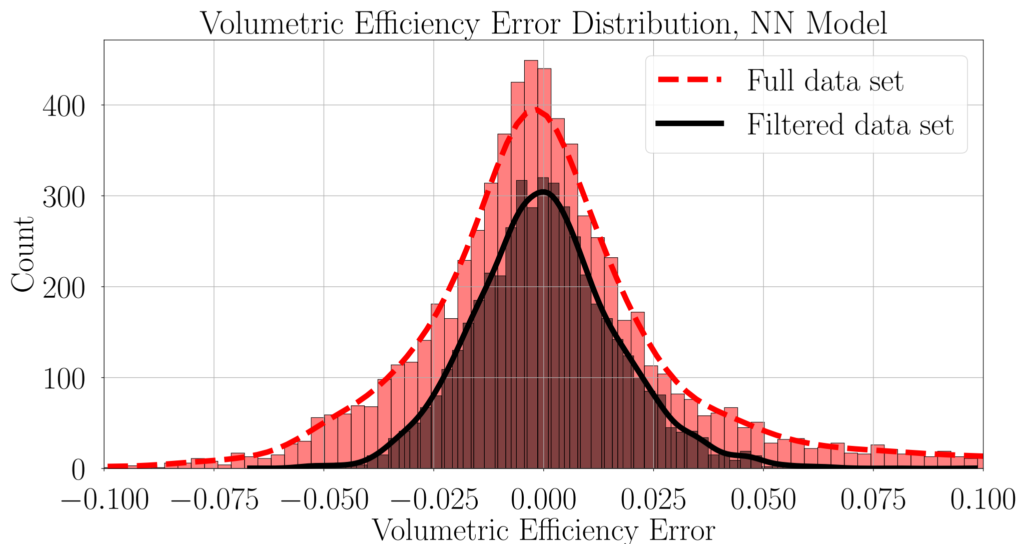
Model	Mean prediction runtime
GPR	112.40 ms
NN	0.138 90 ms

Table 5.1: Runtimes for prediction of volumetric efficiency for 50 data points from the GPR and neural network model, respectively, calculated as the mean over 1 800 runs. The programs are run on a AMD Ryzen 7 PRO 4750U with Radeon Graphics 1.70 GHz and 32.0 GB RAM.

programs are now run on a 1.70 GHz processor with 32.0 GB RAM (see Section 4.1), while the control systems have a 200 MHz processor with 120 kB RAM.
Figure 5.3 shows histograms of the error distribution for the GPR model (Section 4.1.1) and the neural network model (Section 4.1.2), for both the full – red histogram – and the filtered – black histogram – data set. The red dashed line and the black solid line indicate the kernel density distribution for the full and the filtered data set, respectively. Both models have unimodal and symmetric histograms; however, in order to test normality, probability plots of the errors were also made, see Figure 5.4. See Section 6.2 for a discussion of these plots.
The maximum error for the GPR model is 0.093 (scaled) found at `EngSpd=1399.58`, `p_Man=120,27`, `ICPC=50.0` and `ECPC=50.0`. The neural network model has a maximum error of 0.098 (scaled) found at the same data point as for the GPR model. Both the maximum errors and the histograms are similar for the two models, enhancing the proposition of no major accuracy losses in the transition to the neural network.
Figure 5.5 shows how the kernel density distribution of the neural network model varies with size of the training data set for its underlying GPR model. The smaller proportion of data used in training, the heavier-tailed is the error distribution, and thus the larger is the maximum error.

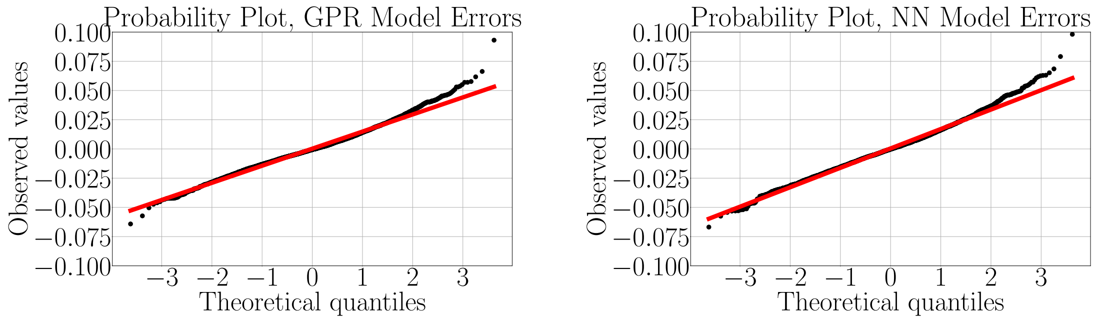


(a) Histogram of volumetric efficiency error distribution for the GPR model. The data is scaled for NDA reasons.



(b) Histogram of volumetric efficiency error distribution for the neural network model. The data is scaled for NDA reasons.

Figure 5.3: Histograms of volumetric efficiency error distribution for the GPR model and the neural network model. The red histograms and red dashed lines indicate the error (and kernel density distribution) for the full dataset, while the black histograms and black solid lines indicate the error (and kernel density distribution) for the filtered data set. Both histograms for the filtered data set are unimodal and symmetric; however, the one for the neural network model has slightly heavier tails. Compare with Figure 5.5.



(a) Probability plot of volumetric efficiency error for the GPR model. The data is scaled for NDA reasons.

(b) Probability plot of volumetric efficiency error for the neural network model. The data is scaled for NDA reasons.

Figure 5.4: Probability plots of volumetric efficiency error for the GPR model and the neural network model (filtered data set). The red line is a least-squares regression line fit to the points. In both plots, the points are close to this linear relationship between theoretical and sample quantiles, deviating slightly at the ends.

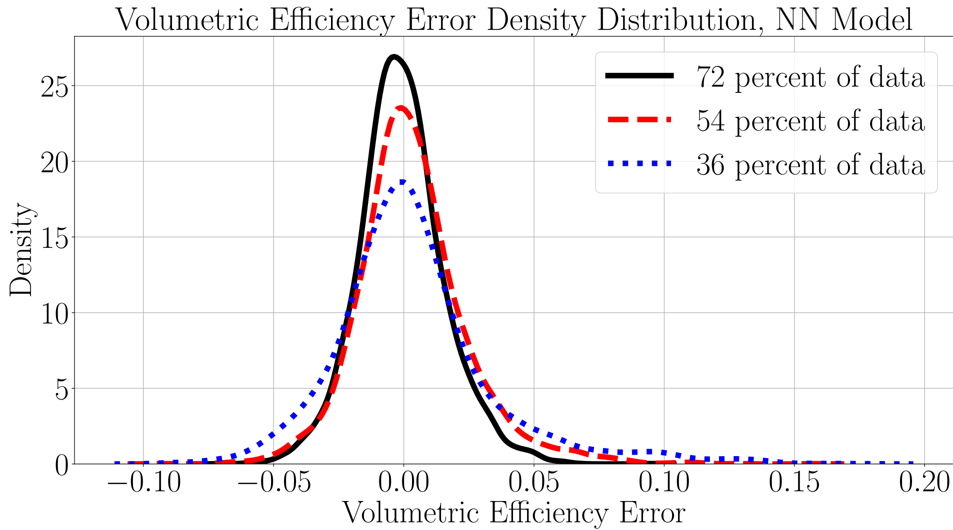


Figure 5.5: Kernel density distribution of volumetric efficiency error for the neural network model (filtered data set), when its underlying GPR model has been trained with 72% – black solid line, 54% – red dashed line, and 36% – blue dotted line – of the measurement data, respectively. The error distribution has heavier tails when a smaller proportion of data is used in training. Compare with Figure 5.3, where the GPR model has been trained with 72% of the measurement data. The data is scaled for NDA reasons.

6

Discussion

This chapter discusses the results presented in Chapter 5. Section 6.1 gives a comparison with the current model used at T-Engineering and section 6.2 discusses whether the decision to use these modelling methods can be justified by the results. The noise reduction process is discussed in Section 6.3. Finally, Section 6.4 presents subjects for future investigation.

6.1 Comparison with Current Model

This section compares the GPR, LIME and neural network models with the RBF interpolation model currently used at T-Engineering.

The challenges of the current model used at T-Engineering is presented in Section 1.2. Considering the amount of data needed for the model, the GPR model now uses 72% of the measurement data for training (see Section 4.2); however, adequate results were also obtained with only 54% of the data. Figure 5.5 shows a comparison of how the error density distribution of the volumetric efficiency predicted by the neural network model varies for different amounts of training data in its underlying GPR model.

As stated in Chapter 5, the LIME explainer showed that the variance of the model predictions was high for cam phaser settings > 40. This might be explained by the scavenging phenomenon, described in Section 2.1.4. High crank angles of the cam phasers lead to a large timing overlap between opening and closing of the intake and exhaust valves. The large air flow through the engine complicates the measurement of volumetric efficiency η_V since it is dependent on how much air enters the engine. Although these challenges are known at T-Engineering, their current RBF model does not indicate any uncertainty in these regions. Section 1.2 describes the issue of high uncertainty as a consequence of lack of measurement data; however, in light of these findings it is important to note that poor running conditions for the engine might also lead to uncertain predictions, since the measurements themselves are uncertain.

Looking at Figure 5.2, the scavenging phenomenon is once again noticed. There is a large overlap in the cam phasers, which the LIME explainer has interpreted as high model uncertainty; however, it is also visible by the number of measurement data points being filtered out in the noise reduction process. As stated in Section 2.1.4, for high cam settings and low intake manifold pressure p_i, the residual gases flow back into the cylinder, causing the engine to run poorly. For high p_i, the high flow

of air through the engine instead causes the volumetric efficiency to increase above 100% – as explained in Sections 2.1.3 and 2.1.4.

6.2 Justifying Selected Modelling Methods

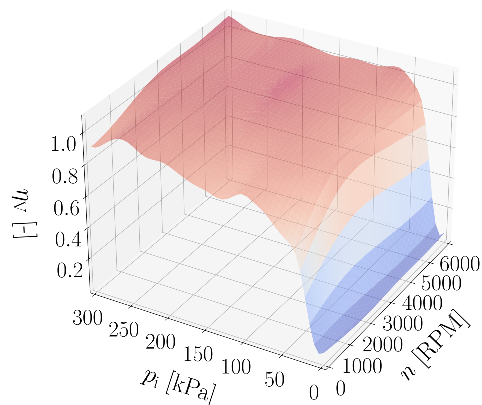
This section seeks to justify the selected modelling methods, presented in Section 1.2.

Section 1.2 presents and motivates the selected modelling methods. In light of the findings in Chapter 5 it is reasonable to ask whether these choices are justified by the results.
As stated in Section 2.2.1, we assume that the measurement noise in equation (2.12) is Gaussian distributed. The probability plots in Figure 5.4 show that the theoretical and sample quantiles have an almost linear relationship, which would indicate that the errors have a Gaussian distribution. There is however a slight deviation at the ends, indicating long tails – i.e. that the variance is higher than expected. Since this deviation is small, we can however justify the assumptions of Gaussian process regression and thus the choice of this modelling method.
Regarding the choice of a neural network to make fast predictions, table 5.1 shows that the mean runtime for making predictions for 50 data points is $\sim 0.14\,\text{ms}$. It should, however, be noted that the programs are now run on a reasonably fast processor and that the CPU in the control systems is more limited (see Chapter 5). Since the programs are written and run in Python and the final programs will be run with C code in Simulink, it is unfortunately hard to tell what the final prediction runtimes will be. This is something that T-Engineering will investigate further when implementing this model in Simulink.
Since the neural network model was so much faster in making predictions than the GPR model (see table 5.1), it makes sense so question why neural networks were not used as the only modelling method. As mentioned in Section 4.4, neural networks require a lot of training data, and since one of the challenges of the current model at T-Engineering (see Section 1.2) was that it required a lot of measurement data this would not have been a useful choice of initial modelling method. Moreover, the GPR model gives information about the variance of the predictions (see Sections 2.2.1 and 3.1), which is necessary for indicating regions of high uncertainty using the LIME model.

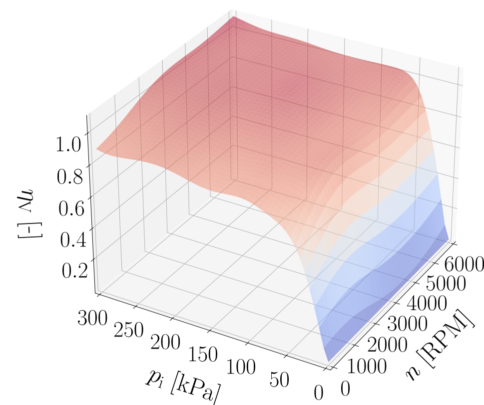
6.3 Noise Reduction Process

This section discusses the noise reduction process, focusing on the balance between a smooth function and following measurement data.

Sections 1.1 and 1.2 describe the need for noise reduction in the model. The data set that was provided in this project had high noise levels, which affected the results of the model. Since the modelled feature is a function describing physical behavior, it should be nice and smooth. This prior knowledge of the data proved to be useful in the building of the model.



(a) GPR model of volumetric efficiency η_V as a function of intake manifold pressure p_i and engine speed n, with intake and exhaust phasers fixed at $\phi_i = 0$ and $\phi_e = 0$. This model has not implemented the `length_scale_bounds` parameter. The data is scaled for NDA reasons.



(b) GPR model of volumetric efficiency η_V as a function of intake manifold pressure p_i and engine speed n with intake and exhaust phasers fixed at $\phi_i = 0$ and $\phi_e = 0$. This model has implemented the `length_scale_bounds` parameter with the lower bound set to 1.5. The data is scaled for NDA reasons.

Figure 6.1: Comparison between two GPR models, where the model in Figure (a) has no lower bound to the `length_scale` parameter – as opposed to the one in Figure (b) which has a lower bound of 1.5. Notice how this makes the surface in Figure (b) smoother than the one in Figure (a), in line with what is physically reasonable.

The building of the model is described in Section 4.2, and it is mentioned that the Matérn kernel was deemed most suitable for this model. The Matérn kernel has two hyperparameters (see Section 2.2.2), where the length-scale ℓ is the one being optimized in the `GaussianProcessRegressor` (see Section 3.1).

Initially, when training the model, the surfaces for fixed cam phasers ϕ_i and ϕ_e (see Section 4.3) were not smooth as expected. Conversely, they were dented due to the high amount of noise in the data. The prior knowledge about the physical behavior of volumetric efficiency η_V was then used in order to improve the credibility of the model. When defining the `kernel`, the parameter `length_scale_bounds` was added with a lower bound set to 1.5 (which was empirically decided upon). This sets a limitation to how low values the `length_scale` parameter can take during optimization – i.e. how fast fluctuations in the output data can occur, and thus gives a more smooth surface. Figure 6.1 shows a comparison between two models, where Figure 6.1a has no lower bound on the `length_scale` parameter and Figure 6.1b has a lower bound of 1.5.

However, since there is still a lot of noise in the data, this resulted in a larger amount

of data being removed in the noise reduction process. Removing data that does not fit the model might lead to overfitting and should be done carefully. Since the decision to introduce `length_scale_bounds` was based on prior knowledge about the physical behavior of the output, this was nevertheless deemed acceptable in this case. Furthermore, the proportion of data removed in the noise reduction process was carefully monitored to ensure that too much data was not filtered out.
Looking at Figures 4.4 and 5.2, we see that most of the filtered out data points are located around $p_\mathrm{i} \approx 50\,\mathrm{kPa}$ – i.e. where the slope of the surface gets steeper. Since there are many data points in this area – see also Figure 4.3b – the variance of the model predictions is smaller. Nevertheless, the rapid changes in volumetric efficiency η_V for $p_\mathrm{i} < 50\,\mathrm{kPa}$ makes the model more sensitive to perturbations. There is a fine balance between wanting low variance and not removing too many points, and in the case of rapid changes in output this was unfortunately not captured as well as hoped.

6.4 Future Investigations

This section presents subjects that might be interesting for future investigations.

GPR requires matrix multiplication when making predictions, which is can be a time-consuming operation depending on the size of the kernel matrix, which increases with the size of the training data set (see Section 2.2.1). Using a kernel such as (2.19) gives an even higher computational cost, since there is an exponential component. It would be interesting to investigate whether an approximation of the kernel, such as a Taylor expansion, would provide a suitable alternative to optimize computation time of the model. However, the time limit of the project did unfortunately not allow this.
The LIME explainer, described in sections 2.3 and 3.2, describes what features contribute to the variance and can thus help to find intervals of high variance. However, by multiplying features before introducing them to the LIME explainer, it could be possible to also find out if the high variance depends on a combination of features. Since the model is multidimensional, this might be desirable. The results of the current method (presented in Chapter 5) were deemed adequate for this project. It could nevertheless be an interesting area of investigation for future projects.

7 Conclusion

This chapter presents the conclusions drawn from the project, focusing on answering the research questions listed in Section 1.1.

The purpose of this master's thesis is to find and efficient and fast way to inter- and extrapolate multidimensional functions. In Section 1.1 the key objectives were presented as the following research questions:

- Is it possible to build a model that accurately describes the relation between input and output parameters, indicating uncertainty where needed?
- Can the multidimensional results be visualized in a pedagogical way?
- Can the model operate in real time on a limited CPU?

In light of the findings of this project, each of these research questions is discussed below.

Is it possible to build a model that accurately describes the relation between input and output parameters, indicating uncertainty where needed?
The models described in Section 4.1 give an accurate description of volumetric efficiency η_V as a function of engine speed n, intake manifold pressure p_i, intake cam phaser ϕ_i and exhaust cam phaser ϕ_e (see Section 2.1). Using the parameter `length_scale_bounds` to set a lower bound for the length-scale parameter ℓ of the Matérn kernel (see Sections 2.2.2 and 3.1) ensures that the model is smooth – in line with what is to be expected physically. By generating extreme points via a separate model of $\eta_\mathrm{V} \cdot p_\mathrm{i}$ as a function of p_i, the model is also able to extrapolate reasonably well.
A LIME explainer (see Sections 2.3 and 3.2) is used to find regions of uncertainty by finding explanations for which features contribute to increasing the variance of the predictions from the GPR model (described in Section 4.1). Furthermore, in the 2D plots of the GPR model – see Figure 4.3 – a 95% confidence interval for the model predictions is given as a means to visualize where the variance is high.

Can the multidimensional results be visualized in a pedagogical way?
The visualization part of the project is described in Section 4.3. By fixing the features that had the longest length-scale ℓ (see table 4.2) – and thus the least variation within its interval, the 3D – Figures 4.4 and 4.8 – and 2D plots – Figures 4.3 and 4.7 – for volumetric efficiency η_V with respect to the other features can show the most interesting properties of the models. Furthermore, as the plots are similar in design to the ones currently used at T-Engineering, they should be accessible to their employees.
In addition to the 2D and 3D plots presented in this report, a Jupyter Notebook program, shown in Figure 4.5, was made. It uses widgets to show how the `alpha`

and `alpha_smooth` parameters (see Sections 3.1 and 4.2) – as well as the confidence interval used in the noise reduction process – affect the final model. This makes the concept of noise level and variance more accessible to people that do not have a mathematical background, as it shows a connection between these tuning parameters and the smoothness of the function.

Can the model operate in real time on a limited CPU?
The operation that should be done in real time is prediction, i.e. the training time of the models is not of interest (up to a reasonable limit). The prediction runtimes of the GPR and neural network models are presented in table 5.1, and since the GPR predictions are only used as a step toward the neural network model, it is the prediction runtime for the neural network that is of interest. This was measured to ~ 0.14 ms, which is indeed fast. However, as these predictions were made using Python on a 1.70 GHz processor with 32.0 GB RAM – as opposed to the final C code in Simulink on a 200 MHz processor with 120 kB RAM – it is unfortunately not safe to say whether the real-time condition has been met.

Bibliography

[1] P. Virtanen et al., ”SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”, *Nature Methods*, vol. 17, pp. 261-272, 2020, doi: 10.1038/s41592-019-0686-2. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RBFInterpolator.html#scipy.interpolate.RBFInterpolator

[2] C. E. Rasmussen & C. K. I. Williams, ”Gaussian Processes for Machine Learning”, *the MIT Press*, 2006, ISBN 026218253X. 2006 Massachusetts Institute of Technology. www.GaussianProcess.org/gpml

[3] C. Molnar, ”Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.”, [Online], 2021-05-18, Available: https://christophm.github.io/interpretable-ml-book/ (accessed on: 2021-05-20)

[4] M. Blondel et al., ”Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] Wikimedia Commons, ”File:4StrokeEngine Ortho 3D.gif”, 2010. [Online] Available: https://commons.wikimedia.org/wiki/File:4StrokeEngine_Ortho_3D.gif (accessed on: 2021-06-07)

[6] C. R. Ferguson and A. T. Kirkpatrick, *Internal Combustion Engines: Applied Thermosciences*. 2nd ed., New York, NY, USA: John Wiley & Sons Inc., 2001.

[7] ISKY Racing Cams, ”Cam Degreeing”, 2018. [Online] Available: https://www.iskycams.com/cam-degreeing.html (accessed on: 2021-08-23)

[8] P. P. Urone and R. Hinrichs, ”Introduction to the Second Law of Thermodynamics: Heat Engines and Their Efficiency” in *College Physics*, Houston, TX, USA: OpenStax, 2012, ch 15.3, [Online], Avaliable: https://openstax.org/books/college-physics/ (accessed on: 2021-06-14)

[9] J. Rice, *Mathematical Statistics and Data Analysis*. 3rd ed., Duxbury, MA, USA: Thomson Brooks/Cole, 2007.

[10] G. Grimmett and D. Stirzaker, *Probability and Random Processes*. 3rd ed., Oxford, United Kingdom: Oxford University Press, 2001.

[11] Rasmussen, Carl Edward and Christopher K. Williams, Gaussian Processes for Machine Learning, pp. 3, 107, 116. © 2005 by the Massachusetts Institute of Technology, published by the MIT Press.

[12] C. Fonnesbeck, ”Chris Fonnesbeck: A Primer on Gaussian Processes for Regression Analysis | PyData NYC 2019”, *YouTube*, Jan. 2, 2020. [Video]. Available: https://www.youtube.com/watch?v=j7Ruu3Yu-70 (accessed on: 2021-03-24)

[13] D. K. Duvenaud, ”Automatic Model Construction with Gaussian Processes”, Ph.D. dissertation thesis, Pembroke College, University of Cambridge, Cam-

bridge, United Kingdom, 2014. [Online] Available: https://www.cs.toronto.edu/~duvenaud/thesis.pdf

[14] M. T. Ribeiro, S. Singh and C Guestrin, ”Why Should I Trust You: Explaining the Predictions of Any Classifier”, *CoRR*, vol. abs/1602.04938, 2016, https://arxiv.org/abs/1602.04938.

[15] B. Mehlig, *Machine learning with neural networks: An introduction for scientists and engineers*, Department of Physics, University of Gothenburg, 2021, https://arxiv.org/pdf/1901.05639.pdf

[16] N. Andréasson et al., *An Introduction to Continuous Optimization.* 3rd ed., Lund, Sweden: Studentlitteratur AB, 2016.

[17] Y. Trsuruoka et al., ”Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty”, in *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, Suntec, Singapore, 2009, pp. 477-485. [Online] Available: https://www.aclweb.org/anthology/P09-1054.pdf (accessed on: 2021-06-29)

[18] K.R. Prilianti et al., ”Performance Comparison of the Convolutional Neural Network Optimizer for Photosynthetic Pigments Prediction on Plant Digital Image”, in *Proceedings of the Symposium on BioMathematics (SYMOMATH)*, Depok, Indonesia, 2018, pp. 020020. [Online] Available: https://www.researchgate.net/publication/332333495_Performance_comparison_of_the_convolutional_neural_network_optimizer_for_photosynthetic_pigments_prediction_on_plant_digital_image (accessed on: 2021-08-25)

[19] M. T. Ribeiro, ”Local Interpretable Model-Agnostic Explanations (lime)”, *Read The Docs*, [Online], 2016, Available: https://lime-ml.readthedocs.io/en/latest/lime.html (accessed on: 2020-05-20)

[20] Torch Contributors, ”PyTorch Documentation”, 2019. [Online] Available: https://pytorch.org/docs/stable/index.html (accessed on: 2021-06-29)

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY