CHALMERS
UNIVERSITY OF TECHNOLOGY

Prediction of Vehicle Component Weights using Multi-Output Regression

An Exploration of Machine Learning Techniques to Perform Automated Data Harmonisation of Vehicle Component Weights

Master's thesis in Complex Adaptive Systems and Data Science and AI

FUNK ELINOR
SJÖGREN FRIDA

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2025

www.chalmers.se

Master's thesis 2025

Prediction of Vehicle Component Weights using Multi-Output Regression

An Exploration of Machine Learning Techniques to Perform Automated Data Harmonisation of Vehicle Component Weights

ELINOR FUNK
FRIDA SJÖGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2025

Prediction of Vehicle Component Weights using Multi-Output Regression
An Exploration of Machine Learning Techniques to Perform Automated Data Harmonisation of Vehicle Component Weights
ELINOR FUNK
FRIDA SJÖGREN

Advisor: Camilla Apoy, Volvo Cars
Supervisor and Examiner: Johan Jonasson, Department of Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover:

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Prediction of Vehicle Component Weights using Multi-output Regression

An Exploration of Machine Learning Techniques to Perform Automated Data Harmonisation of Vehicle Component Weights

Elinor Funk
Frida Sjögren
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

With the ongoing digitalisation, many companies are exploring opportunities to improve operational efficiency through machine learning. Volvo Cars is part of this trend, and their Weight Management and Optimization team seeks an automated way of managing data for component weights of vehicles. The team performs data harmonisation as a preliminary step, prior to production, which is currently computed manually in Excel. This is inefficient and highly time-consuming, however, machine learning models are believed to facilitate the process. Hence, this thesis aims to investigate the feasibility of implementing a machine learning model for automated data harmonisation, in combination with a pipeline for managing the weight data and calculations. The problem was formulated as a multi-output regression problem, and neural network models and ensemble models were deemed appropriate for the purpose. A significant part of this thesis involved experimenting with different preprocessing techniques and model configurations to find the optimal model performance.

The results of the experiments indicated that the ensemble models outperformed the neural network models, where XGBoost was the top candidate. The best-performing XGBoost model achieved a validation accuracy of 84% which was promising, however, the test accuracy was modest, reaching only 22%. This performance was not considered sufficient for the model to be implemented as a part of the final pipeline, since the effort required for deployment of the tool was believed to exceed the benefits. Nevertheless, there remains potential to optimise the model further, and this study can be considered a guideline for further work within this field. The results do not suggest that the use of ensemble models for predicting vehicle component weights is unsuitable, but rather that the optimal way of structuring this problem was not found.

Keywords: Machine Learning, Multi-Output Prediction, Neural Networks, Ensemble Models, XGBoost

Acknowledgements

First of all, we would like to thank our Chalmers supervisor and examiner, Johan Jonasson, and our Volvo Cars supervisors, Camilla Apoy and Pepe Tan. Your feedback and support throughout this thesis have been greatly appreciated. Furthermore, we would like to sincerely thank our friends and family for consistently supporting us throughout this thesis and our studies at Chalmers.

Elinor Funk and Frida Sjögren, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ART	Agile Release Train
CAD	Computer Aided Design
CNN	Convolutional Neural Network
FG	Function Group
GD	Gradient Descent
MAE	Mean Absolute Error
MSE	Mean Squared Error
PLM	Product Life Cycle Management
SGD	Stochastic Gradient Descent

Contents

List of Figures

List of Tables

1

Introduction

The Weight Management and Optimization team at Volvo Cars is responsible for handling a significant amount of data related to the weight of cars and their components. The team's tasks include axial weight distribution calculations and centre of gravity prognosis of vehicles. Computer-aided design (CAD) data contains relevant information required to perform the two tasks. Due to various reasons, the CAD data does not always hold the correct component weights. Thus, the CAD dataset needs to be complemented with weights from the Product Life Cycle Management (PLM) data. However, the PLM weights are not always accurate either, which is why the PLM data cannot simply replace the weights from the CAD data, and instead, careful alignment between the two datasets needs to be performed.

Aligning the weight data, referred to as data harmonisation, is a process that requires experience and expertise in vehicle components and their weights. Currently, the manual data harmonisation process is time-consuming, reliant on the executor, and inefficiently designed in Excel. Therefore, the Weight Management and Optimization team desires a new automated solution. Machine learning algorithms have been widely applied in the automotive industry to improve efficiency in, for instance, but not limited to, decision-making, quality control

and operational optimisation [1], [2]. Given its extensive success in a wide range of applications, machine learning models are considered to be a promising solution to the task.

1.1 Aim

The main goal of this thesis is to develop a machine learning model for data harmonisation. Furthermore, the aim is to create a tool that integrates this model to adjust weights, calculate axial weight distribution, and predict the centre of gravity for vehicles. The tool should also include an interactive interface that displays changes made by the machine learning model, allowing users to make adjustments.

A machine learning approach to the data harmonisation process involves developing a model that can predict the correct weights by analysing the original weights and other relevant component information from the CAD and PLM datasets. When a new car model configuration is introduced, the goal is to apply this tool to determine the correct weights, enabling the calculation of axial weight distribution and centre of gravity prognosis. New configurations may include new components, which is one of the key reasons for considering a machine learning model for weight prediction. A rule-based prediction model would not be able to account for variations or changes in the data over time, as it relies on a fixed set of predefined rules. In contrast, a machine learning model can potentially learn from data and adapt to new patterns or variations, improving its predictions as it encounters more examples.

Since the data harmonisation has been done manually for many years, there exists data that can be used to train a machine learning model,

that is, raw CAD and PLM data along with corresponding adjusted weights. In the datasets, components have features such as agile release train (ART) and function group. The data harmonisation process is considered complete when adjustments of weights are made to ensure that the two datasets align with a maximum difference of 500 grams at the function group level. The model will mimic the adjustments of the training data, and thus, the threshold of 500 grams will be the goal for the model as well. Once the data harmonisation is performed by the model, the idea is to output the changes for the user, such that they can be controlled and potentially adjusted afterwards. Thus, the machine learning model is not aimed to be an independent solution but rather a supportive aid to reduce engineering hours while also providing consistency.

1.2 Prior work

This thesis is a first attempt to automate the data harmonisation process, which has been conducted manually for many years within the Weight Management and Optimization team. Thus, there is no previous work within the company regarding this specific area and application. There is, however, a tool currently used for manual data harmonisation, computations, and visualisation. This tool holds useful information for the process of developing the automated tool that this thesis aims to do, specifically regarding the computations and visualisations. The tool is an Excel file that imports the raw CAD and PLM files. Further, the tool merges the two datasets on the function group level and displays the weight difference. Displaying the datasets' weight differences on the function group level shows the expert which function groups need weight adjustments of components. The com-

putations, that is, the axial weight distribution and centre of gravity prognosis, are pre-computed in a sheet in the Excel file and can be visualised on ART, function group and position level. Due to the large size of the datasets, the computations in Excel become excessively time-consuming. The final Excel file, which contains both datasets, calculations, and visualisations, is approximately 5 GB, making Excel an unsuitable tool for managing such large volumes of data.

1.3 Research questions

The research questions of this master's thesis are as follows:

1. Is it possible to construct an automated pipeline in Python to perform data harmonisation, axial weight distribution calculations and centre of gravity prognosis?

2. Can a machine learning model predict the correct weights with sufficient accuracy, and thereby facilitate the manual data harmonisation process?

   (a) If possible, what type of machine learning model achieves the highest performance in comparison to the previous manual adjustments?

   (b) What level of accuracy would be considered sufficient for the machine learning model to be implemented as a part of the harmonisation process?

1.4 Limitations

This thesis will focus on finding a machine learning-based approach to solving the problem of adjusting the weights. Other techniques, not

including machine learning, might also be suitable for such a problem; however, those will not be considered within this project.

Data harmonisation has been performed for multiple vehicle models with different configurations. However, in this project, the employed data will be limited to the vehicle models referred to as "Project 1", "Project 2" and "Project 3". Each project corresponds to one car model produced by Volvo Cars. Data for the remaining models was considered inadequate for the development of machine learning models. "Project 1" had the largest share of documented data harmonisation, and for this reason, the primary emphasis was put on a machine learning model trained on this data.

During the process of this project, it became evident that no machine learning model met the requirements for implementation in a final pipeline for data harmonisation. For this reason, the pipeline was finalised without the inclusion of a machine learning model.

1.5 Social, ethical, and ecological aspects

There are positive societal benefits with growth in machine learning infrastructure, for instance, that machine learning models are capable of detecting complex patterns, superior to human abilities [3]. However, the digitalisation and surge in machine learning usage require substantial computational resources, which increases energy consumption and water usage for cooling down computers. The environmental impact is an important aspect to consider when transitioning to more automated approaches. The whole machine learning pipeline, from data collection to model inference, should ideally be accounted for in machine learning development, however, this issue will not be addressed

further in this report.

An additional, crucial consideration in machine learning development is the risk of model bias and errors. To place complete confidence in machine learning predictions without human examination can have serious consequences. In this project, incorrect model predictions might lead to production shutdown or imperfect vehicles, making human supervision of the weight predictions essential. This goes particularly in the early stages of model deployment. The aim of the machine learning model is not to work as an independent tool for component weight predictions, but rather to function as an aid for the person computing the component calculations.

2

Theory

This chapter provides a theoretical introduction to the field of machine learning, focusing on ensemble models, neural networks, and preprocessing techniques.

2.1 Machine learning

The field of machine learning is about making predictions based on what is learned from data. It is achieved by the creation of computer programs that can process data, extract useful information and make predictions of unknown properties. Further, some machine learning systems also provide suggestions based on predictions, such as actions to take or decisions to make [4]. Machine learning approaches can be divided into two main components: the learning model and the learning algorithm. Both are pattern-detecting components used to predict something unknown. Given a dataset and its corresponding label set, the machine learning problem can be defined as determining how to fit a model between them [5], as illustrated in Figure 2.1. When fitting a model, the objective is to minimise a defined loss, which quantifies the difference between the prediction and the actual label. The learning algorithm uses the loss function to guide the process of optimisation by adjusting the model parameters to find the best

possible fit.

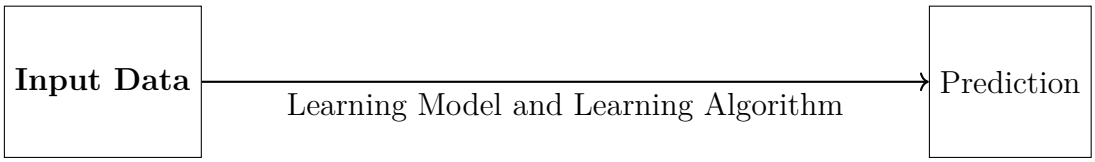

Figure 2.1: Illustration of the machine learning problem, where the input data is typically multi-dimensional, and the prediction is one-dimensional.

2.1.1 Types of machine learning

There are several ways that a machine learning problem can be characterised. One of the primary distinctions is based on whether the data includes labels. In supervised training, the model uses data along with known labels [5]. In this setup, the model learns patterns from the input features and associates them with the corresponding label. Later, when new data without a label is provided, the idea is that the model should be able to predict the label, having learned from many examples of input features and their corresponding labels. On the other hand, when the data does not come with labels, unsupervised learning techniques, also called clustering methods, are employed.

Another main distinction in supervised machine learning is based on the structure of the labels; the two main types of labels are categorical and numerical. Categorical labels mean that the label set is separable and thus can be divided into classes. A numerical label can be either a continuous real number or a discrete value. However, a categorical label is always discrete [4]. The two main divisions of supervised/unsupervised and classification/regression are represented in Figure 2.2.

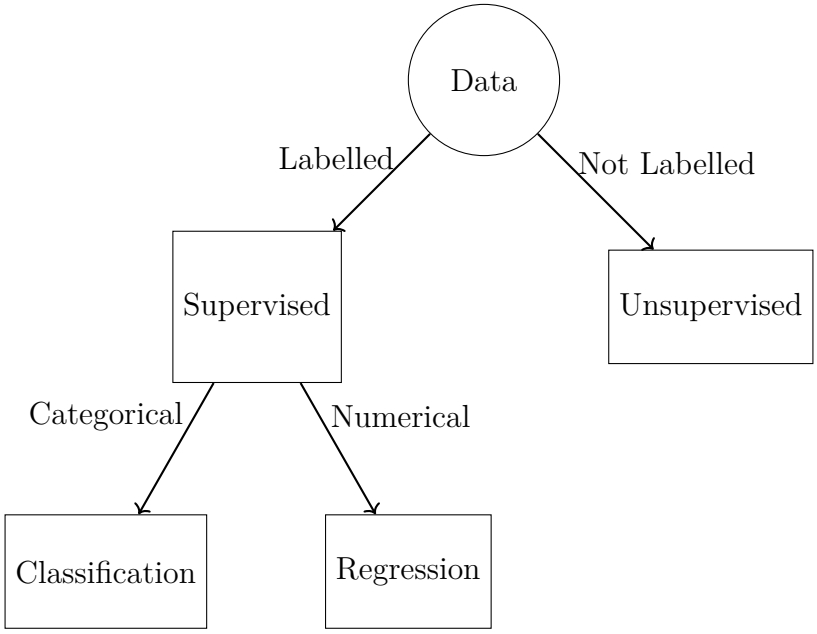

Figure 2.2: Visualisation of the supervised/unsupervised and classification/regression division of machine learning.

2.1.2 Training, validation and testing

In supervised learning, it is a common practice to split the available data into three sets called training, validation and test. The primary purpose of the split is to help the model become fully trained, validated and tested such that the loss can be minimised while the prediction accuracy for new data can be maximised [5]. The choice of loss function used during training depends on the objective and the structure of the data. In regression tasks, common loss functions include mean absolute error (MAE), mean squared error (MSE), and Huber loss. For classification, typical loss functions are cross-entropy, zero-one loss and hinge loss [6]. Accuracy is a widely used metric for evaluating the performance of machine learning models, especially in classification tasks. It is calculated as the proportion of correctly predicted outputs to the total number of predictions made by the model.

The training dataset consists of pairs of input and output, often referred to as features and labels. The training data serves as the foundation for both the learning model and the learning algorithm [4]. Once a model is trained, the validation dataset is used to measure its performance on new data. During validation, the model is provided with input data to make predictions on. Then the model predictions are compared to the true labels. One critical purpose of the validation process is to identify over-fitting. Over-fitting is the phenomenon of when a model tends to fit very precisely to the instances of the training data, resulting in learning noise from it rather than general patterns. Three main factors that often contribute to over-fitting are too complex models in relation to the complexity of the data, a too small training dataset and the presence of noise [7]. If the performance on the validation dataset is much lower than the training performance, this can be an indicator that the model is over-fitting.

In classification tasks, accuracy, as shown in Equation **??**, is commonly used to evaluate performance. However, in regression tasks, performance is typically measured using solely the predefined loss. Additionally, there are many other frequently used metrics such as specificity, precision, recall/sensitivity and F1 score for classification and Pearson's correlation coefficient for regression [8]. Quantifying models with various metrics enables model comparison, which can be an important step in the development of a machine learning system.

During the validation phase, it is common to adjust the model, and this process is often referred to as hyperparameter tuning. Many models provided by various machine learning packages often come with tunable parameters, called hyperparameters. Since there can be a large number of hyperparameters that can take large ranges of values,

validation is often an iterative process. Examples of hyperparameters will be presented in subsection 2.1.3. When dealing with deep learning models, such as neural networks, it is common to also consider different architectures of the model during the validation phase. More detailed information regarding neural networks as models and their validation process will be considered in the subsection 2.1.4. When satisfactory results are found from validation, the chosen model is applied to the test set. The test dataset must never have been exposed to the model until the final testing. The purpose of the testing is to introduce the final model to unseen data and ensure that the tuning parameters work efficiently [5]. Similar to the validation phase, this is achieved using a qualitative measure such as loss or accuracy.

2.1.3 Decision trees and ensemble methods

Ensemble models are machine learning models that utilise the technique of using multiple base learners (often decision trees), to obtain better prediction accuracy [10]. There exist various libraries suitable for machine learning that could be integrated with Python code. One such example is the Scikit-Learn package, which is an open-source library built upon the Python packages NumPy, SciPy, and Matplotlib [9]. Scikit-Learn provides a huge number of models within classification, regression and clustering. Within this subsection, examples of Scikit-Learn models will be introduced, with a certain focus on the Scikit-Learn ensemble module.

2.1.3.1 Decision tree

A decision tree is a supervised machine learning model that uses a hierarchical, rule-based approach to make predictions. It recursively divides the feature space, based on certain rules, and assigns corre-

sponding target values to the resulting leaf nodes. The rules of the splits are learnt from the input data [11]. The learning method can be applied to either classification or regression problems, depending on the nature of the target variable [5].

The Scikit-Learn Decision Tree Regressor is a decision tree model designed for regression tasks. The regressor uses the squared error as the default loss criterion. However, the criterion hyperparameter can also be set to absolute error, Friedman mean squared loss, or Poisson. The chosen criterion measures the performance of the splits made by the model. At each node of the tree, the model splits the data such that the loss is minimised, given that the splitting hyperparameter is set to the 'best' value. The other option for splitter is 'random', which means that the model will randomly select a feature to split the data on rather than finding and selecting the one that minimises the loss [12]. By default, the Decision Tree Regressor continues splitting the data until it can not reduce the variance in the target values any further. However, allowing the tree to grow without any restrictions can lead to over-fitting. The hyperparameters max_depth and min_samples_split can therefore be tuned to control the growth and prevent over-fitting. The min_samples_split hyperparameter sets the minimum number of samples required to allow a split, whilst the max_depth parameter specifies the allowed levels of the tree, limiting the number of splits. However, the model will only reach the max_depth if the min_samples_split condition is not overridden [12].

2.1.3.2 Random forest

Another hierarchical machine learning model is the random forest model. The model is based on numerous random decision trees, together forming a forest, also referred to as an ensemble. The random forest ensembles are created via bootstrap sampling [5]. Bootstrap sampling is a randomisation technique used to generate subsamples of the dataset. The subsamples are created by randomly selecting data points from the original dataset, with replacement, which means that the same data points can appear multiple times within each subsample [13]. Once the individual decision trees are trained on these bootstrap subsamples, the results are aggregated. These two processes, bootstrap sampling and aggregation, are collectively referred to as bagging [5]. The bagging process is visualised in Figure 2.3.

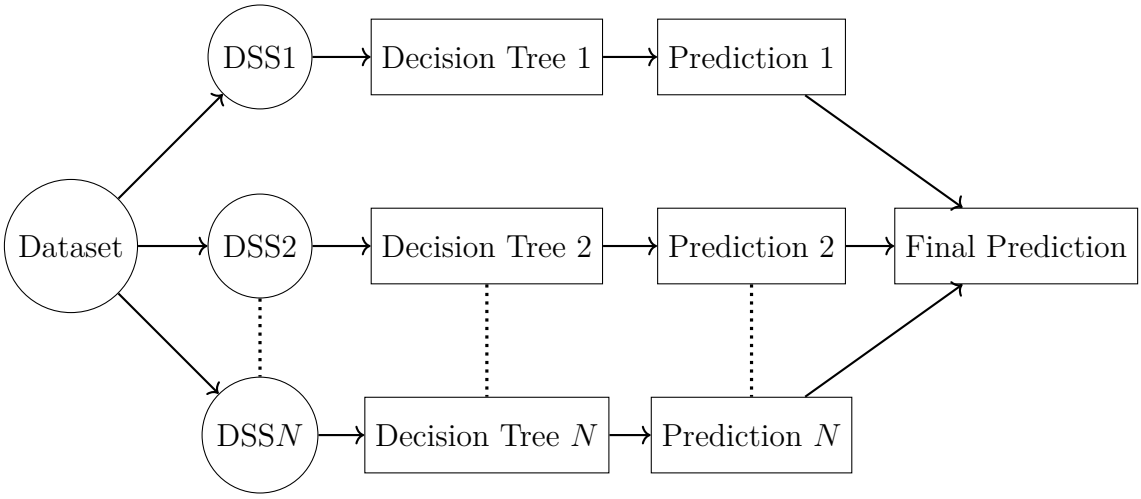

Figure 2.3: Visualisation of the bagging process; where DSS is short for data subsample and N is the number of trees.

The Scikit-Learn Random Forest Regressor is an estimator comprised of several Decision Tree Regressors. The estimator fits random subsamples of the data to each decision tree, and the final prediction is based on an average of all the individual tree predictions. The size of

the ensemble is specified by the n_estimators hyperparameter, where 100 is the default value. The number of samples that each tree uses to make the splits is determined by the bootstrap sampling process. If the bootstrap hyperparameter is set to True, it is then determined by the max_samples parameter; if this one is unspecified, the size of the whole dataset is used [14].

2.1.3.3 Extra trees

Extremely randomised trees, called extra trees, is also an ensemble method based on random decision trees. In contrast to the bagging approach of the random forest model, the extra trees learning algorithm utilises the whole dataset for each tree as default. As in random forest, the ensemble makes predictions based on majority voting among the trees in classification, and computes the average of the predictions for regression. The model introduces further randomisation by random splitting points, which is the key difference from the random forest model that seeks the optimal threshold to use for the split. The algorithm performs several random splits and chooses among them. Besides the random splitting point, extra trees also randomise the features to use for each tree [15]. In the ensemble method of Scikit-learn, the estimator Extra Trees can be found. The main hyperparameters of the estimator are n_estimators, as addressed previously, and max_features, which specifies the number of features to consider for each node split [16].

2.1.3.4 Gradient- and adaptive boosting

Another machine learning method that leverages ensembles of decision trees is gradient boosting. In boosting methods, decision trees are added sequentially. The added tree is trained based on the er-

ror of the other trees in the ensemble. The error from the previous trees corresponds to the negative gradient of the loss function [17]. Thus, the main difference between the random forest and the gradient boosting learning algorithm lies in the way the trees are added. While the random forest learning algorithm utilises bagging, the gradient boosting learning algorithm relies on boosting. The boosting process is visualised in Figure 2.4.

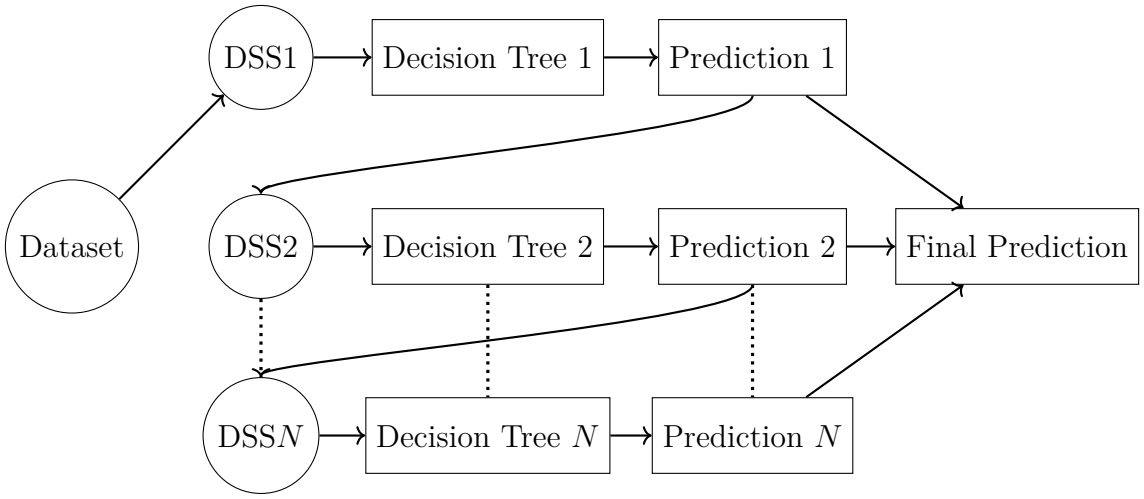

Figure 2.4: Visualisation of the boosting process; where DSS is short for data subsample and N is the number of trees.

Similar to the Random Forest Regressor and the Extra Trees Regressor, the Scikit-Learn ensemble module also provides the estimator Gradient Boosting Regressor [18]. The loss hyperparameter determines the loss function in the Scikit-Learn Gradient Boosting Regressor. By default, the loss is set to squared error, but other options are absolute error, Huber loss, and quantile loss [18]. Similarly to the Random Forest Regressor, the Gradient Boosting Regressor also provides the hyperparameter n_estimators to regulate the ensemble size. Furthermore, the XGBoost library also provides models under the gradient boosting framework [20]. As with the Gradient Boosting Regressor,

the XGBoost model creates an ensemble with an additive approach that minimises the loss function [22]. The default loss function for XGBoost is Mean Squared Error, the default value for the number of estimators n_estimators is 100, and the maximum depth of the tree max_depth is 6. Several other hyperparameters are included in the model, which can be tuned for optimal model performance. Researchers extensively utilise the XGBoost model and have proven state-of-the-art performance in many machine learning challenges [23].

Another ensemble boosting algorithm is AdaBoost, which was the first practical boosting algorithm. It uses adaptive boosting instead of gradient boosting [19]. The AdaBoost algorithm also uses decision trees as weak learners; however, the trees are only split once and are thus referred to as decision stumps. AdaBoost minimises an exponential loss function and works with weighted samples. By weighted samples, it means that the algorithm increases the weights of samples that are difficult to predict correctly and, conversely, decreases the weights of samples whose targets are easier to predict. Aside from the number of splits, which differs between Gradient Boosting and AdaBoost, the primary distinction lies in how each algorithm addresses prediction errors. Gradient Boosting leverages gradients to minimise these, whereas AdaBoost focuses on high-weight data samples to improve performance [10].

2.1.3.5 Multi output regressor and feature importance

A task within the area of supervised machine learning is multi-output classification and regression, which includes the prediction of multi-dimensional labels. Multi-output classification aims to predict multiple discrete output variables, and similarly, multi-output regression

aims to predict multiple real-valued variables [24]. Scikit-Learn provides a Multi Output Regressor that can be applied to regressors that do not support multi-output regression by default. The Multi Output Regressor estimator provides fit and predict [25].

All the above-presented Scikit-Learn Regressors provide a feature importance property that assigns an importance score to each feature. The provided score is a Gini importance, which is based on the normalised reduction of the used loss caused by the feature [14]. The higher the score, the more important the feature is for making the final prediction. Feature selection, which is a form of dimensionality reduction, can be based on feature importance scores. By removing features with low importance, runtime can be reduced, and in some cases, the performance of machine learning models can improve. However, it is important to perform feature selection carefully, as eliminating certain features may negatively affect model performance. Dimensionality reduction can also help address issues related to memory storage and computational costs [26].

2.1.4 Neural networks

Artificial neural networks, often referred to simply as neural networks, date back decades and are inspired by the dynamics of networks in the brain [27]. In the last decade, artificial neural networks have been widely applied across various industries [28], [29], mainly due to better hardware and larger training sets of higher quality [27]. Neural network models are applicable in both classification and regression tasks.

The feedforward neural network can be considered the most repre-

sentative deep learning model [30]. A feedforward network defines a mapping, $y = f(x;\theta)$, and learns the parameters based on training data that provides the best function for generalisation of the output. The network is organised in a layered structure, where the first layer is called the input layer, the intermediate layers are called hidden layers, and the final layer is called the output layer. The network weights define how the inputs are mapped to the outputs. The weights represent the strength of the connections between the neurons in the adjacent layers of the network. To introduce non-linearity in the function $y = f(x;\theta)$, activation functions are generally applied to one or several layers. θ represents all network parameters, including both weights w_{ij} and biases b_j. An example of an artificial neural network can be found in Figure 4.2. In this specific example, the network consists of two input neurons, three hidden neurons, and two output neurons. The arrows represent the network weights $w_{i,j}^{(n)}$ from neuron j to neuron i, which are parameters that are trained to fit the data. The output y_j^n from neuron j in layer n is defined in Equation 2.1. b_j is the bias term.

$$y_j^n = g\left(\sum_{i=1}^{n} w_{ij}x_i + b_j\right) \tag{2.1}$$

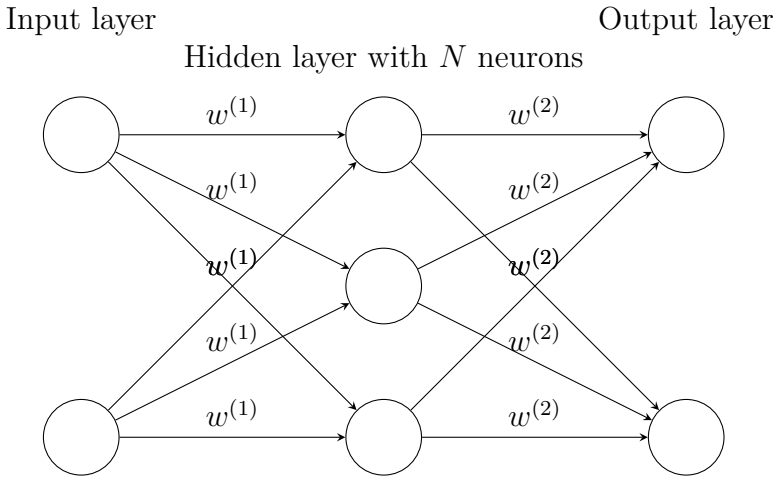

Figure 2.5: An example of a feedforward neural network. In this figure, the network consists of two input neurons, three hidden neurons, and two output neurons.

The default activation function for most feedforward neural networks is the rectified linear activation function (ReLU) $g(z) = max\{0, z\}$. Leaky ReLU, presented in Equation 2.2 is an alternative version of ReLU, where $g(x)$ is the activation function based on neuron output x, that allows negative values of minor magnitude [31]. This resolves an issue that can arise with the ReLU activation function, namely that the gradient becomes zero for negative values. In addition, Leaky ReLU prevents the occurrence of dead neurons in the network. When addressing a regression task with strictly positive real target values, ReLU and Leaky ReLU are the preferred methods.

$$g(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.2}$$

The parameters of the network are trained with back-propagation, by computing the gradient of the loss function with respect to each of the parameters [30]. Subsequently, the gradients are used to update

the parameters accordingly. The non-linearity of neural networks generally introduces a non-convex loss function, and as a result, global convergence is not guaranteed in all cases. Thus, neural networks are generally sensitive to the initialisation of parameters.

Constructing a neural network model requires many design choices such as optimiser, loss function and model architecture, which is one of their considerable strengths as well as a weakness [30]. The flexibility and adaptability that this yields represent an advantage over many other models. In addition, neural networks provide the possibility of mapping an input vector to an output vector of arbitrary dimension. However, the models are sensitive to hyperparameter tuning. More details of the design choices and hyperparameter tuning can be found in Section 2.1.4.1.

Training of neural networks is known to be a difficult task [37]. What is desirable to observe in neural networks training is a steadily decreasing loss, for training as well as validation. After a sufficient number of iterations, the loss will ideally stagnate. A heavily oscillating or non-decreasing training loss can signal a high learning rate, a poor model architecture, or exploding or vanishing gradients. Exploding gradients are, concisely explained, the result of numerous large gradients being multiplied together. Vanishing gradients, on the other hand, arise from increasingly small gradients. One desirable characteristic to observe in model training is that the curves for training and validation loss align closely over time. If the validation loss is significantly higher than the training loss, the model has over-fitted the data. In addition, the training should be terminated when the performance curves stagnate. An example of desirable loss curves can be found in Figure 2.6.

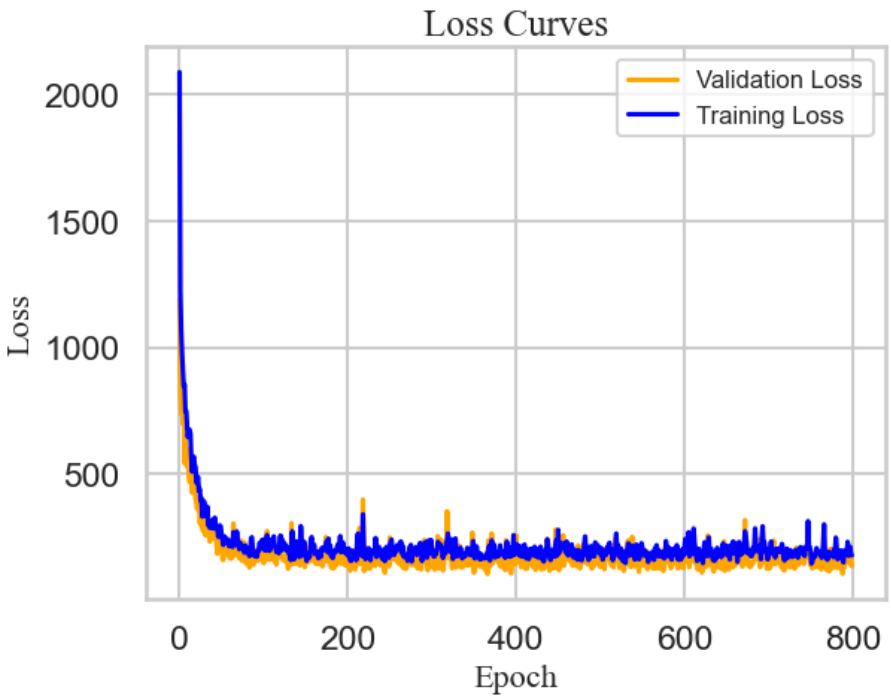

Figure 2.6: An example of optimal loss curves.

A less prevalent but possible observation when training a neural network model is a validation loss lower than the training loss. This means that the model generally performs better on the validation data than on the data that the model is fitted to. This phenomenon might be the consequence of data leakage [32] or extensive use of regularisation techniques to prevent over-fitting [30]. Examples of data leakage are if training and validation data include information that is not available during model inference or if the target variables are directly encoded in the inputs [32].

2.1.4.1 Model configurations and hyperparameter selection of neural networks

Hyperparameters are inputs to an algorithm that control its behaviour [33]. One example is the model width of a feedforward neural network, meaning the number of neurons in one layer. Tuning these parameters

is essential to obtain the optimal results of a machine learning model. An over-parameterised neural network model introduces a risk of over-fitting the data. Conversely, an insufficient number of neurons and layers can result in under-fitting. There are several general guidelines for initialising the width, one of which states that the optimal network width is most commonly in the range between the number of input neurons and the number of output neurons. [35]

There are three main methods for hyperparameter optimisation: Grid Search, Random Search [33], and Bayesian Hyperparameter Optimization [34]. Both Grid Search and Random Search are available in Scikit-Learn, where Grid Search is the most well-known. It is a brute-force exhaustive method, meaning that it iterates through all possible combinations of hyperparameters until the optimal one is found. Random Search is inexhaustive; hence, more efficient in terms of computation. However, Random Search does not guarantee that the most optimal configuration is found [33]. A more flexible and automated approach is to apply Bayesian Optimisation, a method that assumes that the objective function is sampled from a Gaussian process and maintains a posterior distribution for the function as observations are being made. Bayesian Hyperparameter Optimization is an in-exhaustive method as well, resulting in a lack of convergence guarantees [34].

A crucial design choice for model performance of neural network models is the loss function [36]. As mentioned previously, mean square error (MSE), mean absolute error (MAE), and Huber loss are three common selections of loss functions in regression tasks. MSE penalises large errors and assumes that the output distribution is Gaussian. If the data comprises a large number of outliers, the MSE loss function can impose excessively large penalties, making MAE more appropri-

ate. Huber loss, like MAE, demonstrates reduced sensitivity to outliers compared to MSE, while still operating smoothly around zero, unlike MAE. In all three equations below, y_n represents the labels, and $\hat{y}_n$ represents the prediction of the model. The total number of samples in the batch is denoted N, and θ are the model parameters. The threshold δ in Huber loss controls the balance between the quadratic and linear behaviour of the function [36].

$$\text{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.3}$$

$$\text{MAE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.4}$$

$$\text{Huber}_\delta(\theta) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise}\end{cases} \tag{2.5}$$

Optimisation algorithms, or optimisers, represent another critical design component in a neural network model [36]. In this section, details of some standard choices for optimisers will be provided, namely Stochastic gradient descent (SGD), RMSProp, and ADAM. RMSProp and ADAM are built on the same core idea as SGD, but with some nuance, and to fully understand SGD, one has to start with gradient descent (GD). GD is an optimisation algorithm that updates model parameters θ in each iteration based on Equation 2.6, where α represents the magnitude of the update, denoted as the learning rate.

$$\theta_{t+1} = \theta_t - \eta_t \nabla J(\theta_t) \tag{2.6}$$

The updates are computed based on all data points in the samples. For SGD, on the other hand, the stochasticity includes random samples n being chosen from the total set of samples $\{1, \ldots, N\}$. Equation 2.7 defines the update rule for SGD.

$$\theta_{t+1} = \theta_t - \eta_t \nabla J(\theta_t; x_n, y_n) \tag{2.7}$$

Parameter updates with RMSProp and ADAM are computed with random samples in a similar manner [36]. However, RMSProp and ADAM adapt the learning rate based on the moving average of the squared gradients. In addition, ADAM uses momentum, a method for the algorithm to accelerate speed along preferred directions during optimisation. This makes the process of updating parameters more stable. Momentum is useful since other methods, such as SGD, experience trouble navigating the optimisation landscape where the curvature is steep in some directions and flat in others.

Regardless of which optimiser is implemented, one has to carefully select a suitable learning rate to ensure optimal convergence [36]. The learning rate regulates the magnitude of the parameter updates, and a suboptimal one can lead to slow or no convergence. Mainly, one has to be mindful when selecting the learning rate for SGD.

2.2 Preprocessing

It is said that success in machine learning is not all about superior models and faster machines. What contributes to the success of top data scientists is rather their skill in preprocessing the data [38].

"If you put garbage in, you will only get garbage to come out.

By garbage here, I mean noise in data." [38].

A common practice in preprocessing is feature selection, which is a process of excluding redundant features to reduce the number of input variables to the machine learning model [38]. As mentioned above, this enables faster training of the machine learning algorithm, and if the right subset of features is chosen, the model accuracy is improved. There are three main approaches for feature selection: filter methods, wrapper methods and embedded methods. Filter methods are independent of the machine learning algorithm. The core principle of this method is to compute the score in statistical tests to find the correlation between the feature and the outcome variable. In wrapper methods, one tries a subset of features and trains a machine learning model using them. Based on the metrics from this training, one decides whether to retain a feature or discard it. Embedded methods are a combination of filter and wrapper methods, and are methods implemented in machine learning algorithms. Some examples of popular embedded methods are Gini, Lasso and Ridge regression.

Another essential step in preparing the data for the machine learning model is feature transformation [33]. This step generally includes scaling numerical features and encoding textual features. The scaling part is a crucial practice in machine learning preprocessing when numerical features have different scales or when their values span different ranges. Otherwise, large values may dominate over others in the analysis, leading to biased or incorrect results. Scaling the targets is generally not required. Three common methods for scaling numerical features are MinMaxScaler, StandardScaler and RobustScaler. MinMaxScaler is a normalisation method that shifts and rescales data such that it ends up within the range *min* to *max*, where the default

values for *min* are 0 and *max* is 1 [39]. The normalisation is carried out by the following formula:

$$X_{\text{scaled}} = \left(\frac{X - X_{\min}}{X_{\max} - X_{\min}} \right) \times (\max - \min) + \min$$

StandardScaler, on the other hand, only performs standardisation, resulting in a data distribution of unit variance [40]. This makes the StandardScaler method less affected by outliers. The standardisation is computed based on the formula below, where μ is the mean and σ is the standard deviation of the column:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

RobustScaler is a normalisation method that is more robust to outliers than MinMaxScaler [41]. This method subtracts the median of the data and then scales the data according to the Interquartile range (IQR). RobustScaler uses the following formula for normalisation:

$$X_{\text{scaled}} = \frac{X - \text{median}(X)}{\text{IQR}(X)}$$

When handling textual and categorical features, an encoding method is applied to the data [33]. This is of significance since most machine learning algorithms are compatible only with numerical values. Common encoding methods are OrdinalEncoder, LabelEncoder, OneHotEncoder and TfidfVectorizer. The applicability of the methods is highly dependent on the data. The three initially mentioned methods are generally applied to categorical features. OrdinalEncoder encodes values into a two-dimensional array, with values in the range 0 to M-1

[42], where M is the number of labels. LabelEncoder does encoding similarly, with values ranging from 0 to M-1, however, the encoded array is one-dimensional. LabelEncoder is, as the name suggests, primarily applied to the categorical training labels [43]. An issue with both of these encoding schemes is that categories close in order are considered numerically related, even though this may not always hold [33].

Encoding with OrdinalEncoder, LabelEncoder, OneHotEncoder and TfidfVectorizer can be exemplified by the component categories "Screw", "Steering Wheel", "Towbar" and "Flange Screw". OrdinalEncoder and LabelEncoder would interpret "Screw" and "Steering Wheel" to share more common properties than "Screw" and "Motor". As mentioned, this might not always be the case. To resolve this problem, OneHotEncoder or TfidfVectorizer can be applied. OneHotEncoder creates a matrix using a one-hot scheme, and is advantageous when dealing with multi-class categorical features. TfidfVectorizer is preferable when the semantic meaning of textual features is important. These encoders are available in Scikit-learn. An illustration of the encoding using Ordinal- and LabelEncoder can be found in Table 2.1. Encoding using OneHotEncoder and TfidfVectorizer is illustrated in Table 2.2.

Component Name	**OrdinalEncoder**	**LabelEncoder**
`"Screw"`	[1.]	1
`"Steering Wheel"`	[2.]	2
`"Towbar"`	[3.]	3
`"Flange Screw"`	[0.]	0

Table 2.1: A table illustrating the encoding of component names using OrdinalEncoder and LabelEncoder.

Component Name	OneHot	TfidfVectorizer
"Screw"	[1, 0, 0, 0]	[0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00]
"Steering Wheel"	[0, 1, 0, 0]	[0.00, 0.00, 0.00, 0.58, 0.58, 0.00, 0.58]
"Towbar"	[0, 0, 1, 0]	[0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00]
"Flange Screw"	[0, 0, 0, 1]	[0.62, 0.62, 0.49, 0.00, 0.00, 0.00, 0.00]

Table 2.2: A table illustrating the encoding of component names using OneHotEncoder and TfidfVectorizer. The seven encoding values in the TfidfVectorizer column are ["Flange" "Flange Screw" "Screw" "Steering" "Steering Wheel" "Towbar" "Wheel"].

The scores in the TfidfVectorizer are a combination of the "term frequency" (TF) and the "inverse document frequency" (IDF) [36]. Term frequency is the frequency of the word in the text, and document frequency represents how many times the word appears in all texts together. As demonstrated in Table 2.2, the Tfidf-based encoding scheme captures the relationship between the components "Screw" and "Flange Screw" in a favourable way. The TfidfVectorizer creates word vectors such that words of semantically similar meaning are close to each other.

3

Domain specific theory

The first section within this chapter provides a detailed description of the data harmonisation process and its main challenges. Further, this chapter includes data specifications of the CAD dataset and PLM dataset, along with the adjusted weight data. Following this, sections describing the aim and general idea of the two tasks, axial weight distribution calculations and centre of gravity prognosis are presented.

3.1 Data harmonisation process

The alignment of weight data, also referred to as data harmonisation, is the process of aligning component weights between the two datasets. Which of the CAD and PLM datasets holds the correct weight varies between components and vehicles. Thus, none of them can serve as master data independently. A component has the feature ART, function group and position in the respective dataset. These are all features describing what different groups a component belongs to. An ART is an organisational grouping, representing the department a component belongs to. Function group and position are physical groupings, corresponding to where components are located within the vehicle. The goal of data harmonisation is for the CAD and PLM datasets to align with a maximum of 500-grams difference at the func-

tion group level. The data harmonisation is highly time-consuming, requiring approximately one day per vehicle.

Components are grouped on the function group level to determine whether a function group fulfils the requirement of a maximum 500-gram difference. One factor complicating the process is the discrepancy in the number of components sharing the same ART and function group between the two sources. Generally, a single component in the PLM data corresponds to multiple smaller components in the CAD data. To exemplify, consider that one component in the PLM data is represented by five smaller components in the CAD data, all of which share the same ART and function group. x_1, x_2, x_3, x_4, x_5 are the weights of the CAD components and y_1 is the weight of the PLM component. This fictitious example is represented in table 3.1.

ART	Function group	CAD weight	PLM weight
1	1	x_1, x_2, x_3, x_4, x_5	y_1

Table 3.1: Structure of a fictitious function group example with selected columns; ART, function group, CAD weight and PLM weight.

Assume further that in this fictitious example, the summarised weight of the CAD component is less than the weight of the PLM component. Additionally, assume that the weight difference is more than 500 grams, such that:

$$x_1 + x_2 + x_3 + x_4 + x_5 + 500 < y_1 \tag{3.1}$$

Furthermore, it is assumed that the expert responsible for the data harmonisation knows that the weights in the PLM data are correct for these components. Since the weight difference on the function group level is more than 500 grams, adjustments are required. The expert

must analyse the weights in the CAD data. If the five components are identical, the weight difference will simply be distributed evenly among them. However, this is just one possible outcome among a large number of potential scenarios; the relationship between components in the two datasets may vary. The five components in the CAD data may not be identical, and their weights could differ from one another. The expert performing the data harmonisation will need to carefully assess these variations, taking into account the broader context and any additional relevant information, to ensure that the harmonisation process accurately reflects the true values across both datasets.

Another possible scenario is that components are missing in one of the datasets. One such example is that fluids are not incorporated in CAD models, which implies that all types of fluid are only present in the PLM data. For cases like this, new components are added. This is an example of a consistently occurring error, and another such example is that cables are always added into CAD as 1.5 times their actual size. Thus, cable weights are systematically increased. However, not all errors follow patterns like this; some of them are engineering mistakes. An example of such a mistake would be to accidentally fill an identity number as the weight, resulting in huge function group weight differences between the sets.

At present, data harmonisation and computations are carried out in Excel. Due to the large size of the datasets, the computations in Excel become excessively time-consuming. The final Excel file, which contains both datasets, calculations, and visualisations, is approximately 5 GB, making Excel an unsuitable tool for managing such large volumes of data.

There exists data of previously performed data harmonisation for the car projects referred to as "Project 1", "Project 2" and "Project 3". One car project corresponds to one car model constructed by Volvo Cars. In this study, the data will be limited to these car projects only. The harmonisation has been completed several times for the same vehicle, including different configurations, such as for "Project 1" with and without a tow bar or child seat. The adjusted weight data from these computations serve as ground truth data for the machine learning model, meaning that the model aims to correctly predict these adjusted component weights.

3.2 Calculation of axial weight distribution and centre of gravity prognosis

The harmonised data is used in calculations for the front and rear axle load of the vehicle. The performance of these calculations is crucial for driving manoeuvrability, staying within legal weight limits and reducing the risk of premature damage to the vehicle [44]. In addition, the vehicle weight is essential for safety, since an imbalanced car might behave in an unpredictable manner. The specific formulas for calculations of weight axial distribution will not be presented in this report, but in short, the centre of gravity for each component, the wheelbase and the positions of the corresponding axles are utilised.

4

Methodology

Since the data harmonisation is performed on function group level, it was decided early in the process to group components in this way in the datasets. As a consequence, the feature and target values are represented as arrays, thus, the problem can be framed as a multi-output regression problem. This condition on the input and output data brings limitations to which preprocessing techniques can be applied and which machine learning models can be used. The following sections will present the preprocessing methodology and the methodology for the implementation of the machine learning models considered in this project: neural networks and ensemble models.

4.1 Preprocessing

This section presents the different preprocessing steps applied to the data within this project. The first section covers the creation of data frames, followed by implementing a pipeline for data cleaning and transformation of training, validation and test data. The core components of the pipeline are imputation of missing values, scaling of numerical features, encoding of categorical features and data padding. This preprocessing pipeline is applied to a CSV file with data for training, validation and testing of the explored models. The data was

determined to be in the form of NumPy arrays instead of lists, which simplifies mathematical operations. However, this array structure creates complexity in the preprocessing pipeline, since many inbuilt preprocessing modules are designed for single-value data. The process of finding the optimal preprocessing techniques followed an iterative approach, where several encoding schemes and scaling methods were evaluated, as well as ways of structuring the data. As a result, multiple datasets of varying sizes were used for model training.

The inference data is preprocessed using the techniques that yielded the best model performance. Additionally, since the inference data lacks labels, the step for file merging differs slightly. Preprocessing of inference data will be discussed in greater detail in Section 4.1.4.

4.1.1 File merge

For each vehicle configuration, a data frame is constructed. This data frame is comprised of the raw CAD and PLM data, and the adjusted CAD and PLM weights. The preprocessing script reads the files and finds the columns ART and function group. All unique combinations of ART and function group are identified, and the files are grouped according to this. This results in a data frame where each row, or sample, consists of all components in the vehicle sharing the same value for ART and function group. Several components share the same combination of these two features, and thus, in the merged data frame, nearly all feature values are arrays. The arrays are sorted on weights, in ascending order. A fictitious example of a dataset after the file merge is found in Table 4.1. One row in the table represents one data sample, which consists of n components sharing the same values for ART and function group (FG). The first data sample consists of

three components and the second of two components.

ART	FG	POS	CAD W.	PLM W.	Adj. CAD	Adj. PLM
1	10	[30, 15, 50]	[2, 30, 1755]	[2, 30, 1170]	[0, 30, 1170]	[0, 30, 1170]
2	20	[4s0, 25]	[30, 3000]	[30, 4000]	[30, 4000]	[30, 4000]

Table 4.1: An illustration of the merged data frame. CAD W. and PLM W. represent the component weights in the corresponding datasets. Adj. CAD and Adj. PLM. represents the adjusted weights. Note that only seven of the 150 data columns are represented in this fictitious example.

There are two main reasons for grouping components based on function groups. First, the goal of data harmonisation is to adjust the weights so that the datasets have a maximum difference of 500 grams per function group. This division is logical because it aligns with the measure we aim to minimise, the error per sample, that is, a group of components. Second, as previously explained, there is a discrepancy in the number of components between the datasets at this level. The CAD data, for example, often represents components in greater detail. Due to this discrepancy, it is not possible to treat each component as a separate sample, as it may not have a corresponding component in the other datasets.

By stacking several vehicle data frames, a merged data frame is created, which consequently includes data from multiple vehicle configurations. In this project, three sizes of data frames were used for model development, which included data from different configurations of three vehicle models. The car models are referred to as "Project 1", "Project 2" and "Project 3". One small data frame was created that comprises data from four car configurations of "Project 1", a medium-sized data frame with ten car configurations of "Project 1" and a large

one comprising data for 41 different car configurations of all accessible car projects were created. The small and medium datasets solely contain data of "Project 1", since this car model offers a substantial amount of data. The large dataset contains all the data that was available at the start of the project. The file merge resulted in the three training datasets, or frames, for model training with the following number of samples:

Dataset	**Samples**
Small	690
Medium	1,710
Large	6,835

Table 4.2: Sample size of each dataset.

Each of the datasets serves a different purpose. The small data set with few samples is useful for pilot runs, since it offers less training time. When model specifications with high performance are found on the small dataset, one can proceed to the medium dataset for additional analysis. When optimal model specifications are found, the model will be trained on the large dataset, since more data typically results in better model generalisation on unseen data.

As stated earlier in this section, there exists a variety of versions of these datasets preprocessed using different techniques. The specifics of the preprocessing techniques explored in this project will be discussed in further detail in the next subsections.

4.1.2 Data cleaning

The imputation of missing values in the data is an essential step to ensure data quality. The data values for important features might

be missing due to limitations in the data sources or as a consequence of inaccuracies in data management. The optimal way to resolve the issue of missing data was carefully examined, mainly since the imputation will affect the final calculations. A suitable approach was found to be imputing missing values with zeros. In addition, the arrays were padded with zeros such that each feature has values with the same dimensions. The length of the features was based on the sample in the dataset with the longest feature value. However, as the project progressed, it became clear that this was not the most optimal approach for padding. Rather, all features were padded to the same length, namely 1500. The sample with the largest number of components contained approximately 1,000, so the padding length had to exceed this. A value of 1,500 was chosen to provide a margin. The inference data was padded equally, which facilitates the model in detecting patterns in the inference data.

The data contains around 150 columns, so-called features. To reduce the dimensionality of the dataset and enhance the efficiency of training, it is preferred to exclude some features by feature selection. The goal was to find a suitable method for this among the three mentioned above: filter methods, wrapper methods and embedded methods. Filter methods require the computation of a covariance matrix, which is a task of significant complexity when handling array-type values. For this reason, the filtering technique was not implemented as a part of the final preprocessing pipeline. However, embedded feature selection methods were applied to ensemble models, which is discussed further in Section 4.3.

There exist samples in the dataset that include inaccurate adjustments of weights as labels. This refers to samples with a discrepancy between

the CAD and the PLM components' weight sums after adjustments greater than 500 grams, which can occur when the calculations were not intended to apply to all function groups. These samples were included in the datasets during the neural network model training, but were later filtered out as part of the data cleaning. Additionally, weights that do not require adjustments were filtered out. This was the case for the datasets used during the training of both neural network models and ensemble models. These are the components where the sums of the CAD and PLM weights align based on a threshold. The threshold was set to zero or 200 grams. The percentage of correct weights varies depending on the dataset and the threshold. The objective of the model is to adjust the weights where the two sums do not align, making this a suitable step to reduce the dimensionality of the data.

4.1.3 Feature transformation

For this project, scaling of numerical features and encoding of textual features are the steps included in the feature transformation. To achieve feature transformation, primarily, one has to decide which features are numerical and which are textual. This was mainly executed automatically, since the data contains around 150 features, except for three numerical features that were manually pre-determined to be encoded as textual features. This is owing to the features being numerical categorical values, such as "Component ID", which are not associated with a numerical magnitude. The function for the automated decision-making attempts to convert all the values in each feature to a numerical format. If this is achieved successfully without undefined values, the feature is considered a numerical feature. As a subsequent step, the most suitable method for feature scaling should be decided.

A significant number of the numerical components in the data contain some extreme outliers. Visualisations of the distribution of six key features are found in the histogram plots in Figure 4.1. In all of the features, one can observe a frequent occurrence of outliers, which motivates the use of StandardScaler and RobustScaler. Nevertheless, since the data is padded with zeros, this can affect the scaling with StandardScaler, since this method relies heavily on the mean and the standard deviation. This is, on the other hand, not an issue for MinMaxScaler or RobustScaler. For this reason, MinMaxScaler and RobustScaler were the methods used in this project. Primarily, model performance under MinMaxScaler preprocessing was evaluated, followed by an evaluation of model performance with RobustScaler.

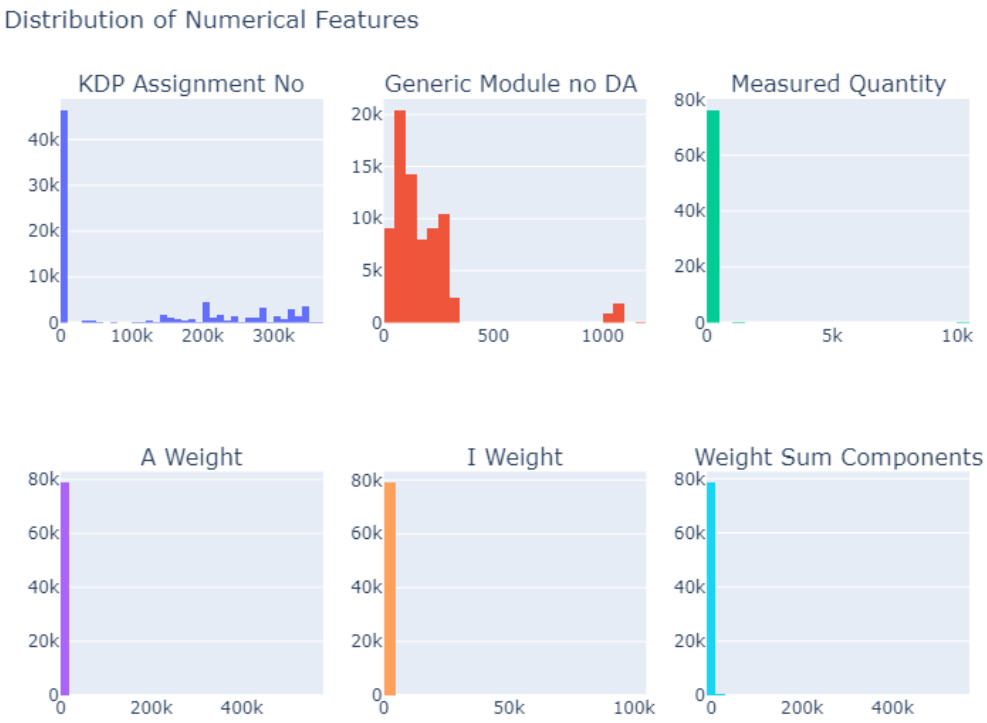

Figure 4.1: The distribution of the values of the numerical features KDP Assignment NO, Generic Module no DA, Measure Quantity, A Weight, I Weight and Weight Sum Components.

Scaling with RobustScaler is computed using the median and the interquartile range. Since the data is padded with a significant number of zeros, both the median and the interquartile range are occasionally zero. In this case, the values are not affected by scaling. For this reason, the performance of models with scaling applied before padding was investigated. This was limited to the training data for the ensemble models.

Regarding encoding of textual features, TfIdfVectorizer was the method of choice, since it is capable of capturing stronger relationships within textual data compared to many other encoders. Examples of the textual features in need of encoding are the component names as *"Flange Screw"*, *"Socket Screw"* and *"Sems Screw"*. Using LabelEncoder, OrdinalEncoder or OneHotEncoder, these components will be encoded as completely different, whereas the relationship between these components is markable and the model might benefit from recognising these. However, the TfidfVectorizer in Scikit-Learn is incompatible with the array-structured data utilised in this project. In addition, encoding with TfidfVectorizer increases the data dimensionality remarkably. Consequently, encoding with TfidfVectorizer was excluded from the preprocessing pipeline in this project. OneHotEncoder is not a suitable option, since the main part of the features are multi-class features, leading to OrdinalEncoder and LabelEncoder being the most viable options. One motivation for applying LabelEncoder to the textual features in the data, although it is more commonly applied for targets, is that the encoded feature remains in a one-dimensional format. This is of value since the dataset is already in a quite complex array-format. For this reason, LabelEncoder was used as the main encoding scheme for the training data. However, after evaluating models

trained on LabelEncoded data, OrdinalEncoder replaced the LabelEncoder in the preprocessing pipeline.

In addition, a handmade encoder was manually created to capture more of the relationships in the textual data. The handmade encoder combines tokenisation from TfIdfVectorizer with integer encoding from LabelEncoder by creating an array for each label, where each value in the array represents each word in the label. Table 4.3 illustrates an example of encoding with the handmade encoder. As shown in the example, the encoding represents relationships in the data in a superior way to LabelEncoder and OrdinalEncoder. This encoding scheme also results in significantly lower-dimensionality representations compared to using TfIdfVectorizer. However, the handmade encoding scheme still led to data with a structure that was cumbersome to manage. The complex format could not easily be used as input to the machine learning models, and for this reason, the handmade encoding scheme was never implemented as a part of the final preprocessing pipeline.

Component Name	**Encoding**
Flange Screw	[0, 1]
Socket Screw	[2, 1]
Sems Screw	[3, 1]

Table 4.3: An example of encoding with the handmade encoding scheme.

4.1.4 Preprocessing of inference data

The structure of the data for model inference will look slightly different, since no labels are available. For this reason, the pipeline for preprocessing deviates from the pipeline described above, mainly con-

cerning the initial file merge. The script preparing data for inference takes two raw data files as input: the CAD and PLM data. These are merged based on ART and function group similarly to the training and validation data, hence, inference data has samples corresponding to all components in the same function group, sharing the same value for ART. Apart from this, the data is preprocessed using the techniques mentioned above, which yielded the best model performance during training and validation. The preprocessing steps included: scaling numerical features with RobustScaler and encoding textual features with OrdinalEncoder. The function groups where the CAD and PLM components' weight sums aligned were filtered out.

4.2 Neural network models

Neural networks were believed to capture underlying patterns in the data. For this reason, an introductory feedforward neural network was implemented as a starting point, using a single hidden layer. The activations explored for the model were linear activation, ReLU and Leaky ReLU, where a feedforward neural network with linear activation is mathematically equivalent to performing linear regression [30]. A substantial part of this project involved experimenting with different configurations of neural networks to find the one achieving the highest accuracy. A visual representation of the initial model can be found in Figure 4.2.

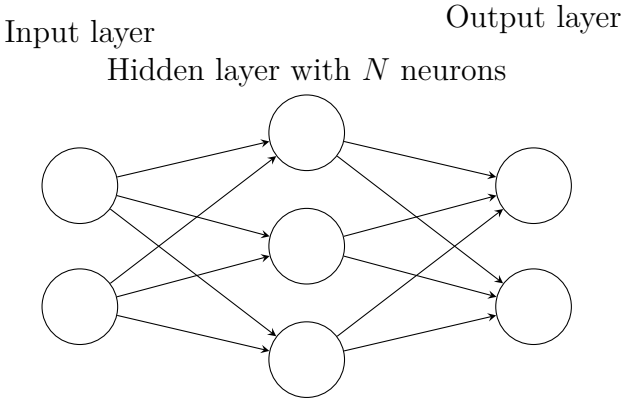

Figure 4.2: An illustration of the initial feedforward neural network. The network consists of two input neurons, three hidden neurons, and two output neurons, however, this is not aligned with the dimensions of the actual model. The input and output dimensions depend on the dataset employed for model training, and the width N of the network is tuned as a hyperparameter.

The neural network models were implemented using the PyTorch library, which is an open-source machine learning library that provides tools for effective implementation of neural networks [46]. Additionally, a DataLoader in PyTorch was created, using efficient management of batching, accessing, and shuffling data. The models were trained both locally and using Databricks.

4.2.1 Training and hyperparameter tuning

A following step in model implementation included finding the optimal model architecture for this specific area of application. To achieve this, several model configurations were investigated, with the model in Figure 4.2 as a basis. The training and validation loss for all model configurations were monitored, and decisions on network depth, width, or modifications of hyperparameters were guided by this. If oscillations in training or validation loss were observed, the learning rate was decreased, following standard machine learning practice. As men-

tioned previously, ADAM and RMSProp can be suitable alternatives if SGD is the optimiser algorithm causing oscillations in loss curves. If the loss appeared to continue to improve after the maximum number of epochs, this indicates that the model is capable of further improvements and that training was stopped prematurely. In these situations, the hyperparameters were saved and later reused in a prolonged training run.

A common practice when developing neural network models is to start by creating a simple network as a dummy model that serves as a baseline. Hence, experiments with the shallow network in Figure 4.2 were executed, both with linear output and including activation functions. ReLU and Leaky ReLU were believed to be appropriate choices, since the predicted weights are strictly positive numerical values. As previously mentioned, the decision of width in a network is highly relevant for the final performance of the model. General guidelines in machine learning development state that the optimal network width is most commonly found in the range between the number of input neurons and the number of output neurons. In this case, for the medium-sized dataset, which is most frequently used in model training, the input size is approximately 80,000, and the output size is around 1,000. If one follows these guidelines, the ideal network should have a hidden layer of size 1,000-80,000. Due to memory limitations, both in Databricks and locally, a neural network with more than 4,096 neurons was not feasible. For this reason, a hidden size in the range from 1,000 to 4,096 was applied in the hyperparameter tuning. Additionally, 512 was added to the potential neuron numbers, since a smaller network possibly provides more efficient training. This initial run with a single-layer network was carried out for different sets of hyperparam-

eters. For further information on the specific hyperparameters tested out, see Table 4.4.

Hyperparameters				
Loss function (Loss fn)	MSE	MAE	Huber loss	MAE Alt.
Learning rate (LR)	0.001	0.0005	0.0001	0.00005
Batch size	32	64	128	256
Number of neurons in hidden layers (N)	512	1,024	2,048	4,096
Optimiser (Opt.)	SGD	ADAM	RMSProp	

Table 4.4: A table presenting the hyperparameters evaluated in the neural network models.

Following the preliminary training, it became apparent that a more complex model was required to fit the data. This is reasonable since the target data - how the weights of specific components should be adjusted based on expert knowledge - in this project is considered quite complex. Consequently, additional layers were added to the model. The results of models with a range of two to five layers were examined, both with and without activation functions. A more detailed report of the final results observed in the implementation of neural network models can be found in Section 5.1.

Training the models demanded substantial time in this project. The number of epochs for each experiment was decided based on observations from early pilot runs with similar hyperparameters. Around 800-3000 epochs were required to obtain satisfactory results, which refer to either a low validation loss or a high validation accuracy, combined with a stagnated performance curve. To avoid unnecessary training, early stopping was implemented, which terminates the training if the validation loss has not improved in a certain number of epochs. The threshold for early stopping can be tuned as a hyperpa-

rameter. For reference, training a single-layer model with 512 neurons for 800 epochs required approximately 4 hours. The training time varied depending on the model architecture and chosen hyperparameters.

The tuning of hyperparameters was initially carried out using Bayesian hyperparameter optimization. Grid Search was considered an unfeasible method, partly due to the high dimensionality of the data. In addition, five parameters with four alternatives each result in 1,024 combinations to evaluate, which is not reasonable to process with an exhaustive method. A table with all hyperparameters that took part in the optimisation is found in Table 4.4.

For all model configurations examined, the training loss and validation loss were documented in the platform MLfLow, which is suitable for tracking and organising results in machine learning projects [21]. In addition, the training and validation accuracy were registered, and the definition of accuracy in this project will be further discussed in Section 4.4.

As an attempt to circumvent the issue of the padded zeros, a custom loss function was created, called MAE alternative (MAE Alt.). The function calculates the mean absolute error after excluding all zeros in the array. However, this means that even the zeros that belong to specific components are neglected. Nonetheless, this was an attempt to more accurately measure loss during model training, to help the model increase the accuracy of the predictions.

4.3 Ensemble models

In addition to exploring various neural network architectures, it was decided to employ simpler machine learning models. The initial desire

to use ensemble models was to be able to perform feature selection, which was deemed crucial to reduce the training time of the neural networks. Additionally, feature selection was considered to potentially improve the performance of the carried-out models. As a first step, the Scikit-Learn Random Forest Regressor was employed for feature selection. Using the estimator with its default parameters on the small training dataset, it outperformed all previously tested neural network versions in terms of average loss per function group and accuracy. At this stage, it was decided to explore more ensemble methods. To decide which models could be suitable for the data and the task, an experiment was conducted for the small dataset. The small dataset experiment included the following models:

- AdaBoost Regressor
- Extra Trees Regressor
- Gradient Boosting Regressor
- Random Forest Regressor
- XGBoost Regressor

The selection of regressors was an iterative process. As mentioned, the Random Forest Regressor was used as a starting point. The random forest algorithm is well-known and widely used in various machine learning applications, referred to as the most popular decision forest algorithm [47]. The algorithm has proven both speed and accuracy independently of the nature of the dataset [48]. Further, the random forest model has demonstrated success in various multi-output regression tasks [49], [50], [51]. In the process of seeking precious work

related to random forest multi-output regression, the Extra Trees Regressor was also mentioned as a potential candidate for this purpose [52], and thus chosen as a candidate model.

Given that the random forest and extra trees algorithms employ bagging to create their ensembles, it was deemed appropriate to also explore models based on boosting as an alternative approach. Therefore, the Gradient Boosting Regressor was the initial model from the boosting framework that was considered. Similar to the random forest algorithm, it has shown great success in a wide range of applications [53]. Furthermore, the gradient boosting algorithm has also succeeded in multi-output regression problems [54]. When further exploring models under the boosting framework, it was discovered that extended gradient boosting algorithms such as XGBoost have been shown to outperform random forest and gradient boosting models in some applications [48]. Thus, the XGBoost model was incorporated in the test as well. While reviewing the literature on multi-output regression, the AdaBoost model also appeared frequently in various applications [55], [56], [57]. Consequently, it was decided to also incorporate the AdaBoost Regressor within the small dataset experiment.

All the selected ensemble models require that each sample have the same input data length. This was solved by padding at the feature level. However, these models can not deal with feature values as arrays, and therefore, features are flattened before training. This implies a huge increase in features for the ensemble models, as each value in the list now serves as an individual feature. For example, consider one feature called feature_1, where the padded length of this feature is 20. After flattening, there are now 20 features made out of this one, feature_1_1, feature_1_2, ..., feature_1_20. Another crucial step in

using the ensemble method on this dataset was to wrap the regressors in the Multi-Output Regressor provided by Scikit-Learn, as the estimators do not support multi-output regression by default. Thus, after an ensemble estimator is initialised, it is wrapped by the Scikit-Learn Multi Output Regressor before training.

When training the ensemble models, it was decided to train two separate models. One for predicting the CAD output and one for the PLM output. Both models were provided with the same input data, all the features from the respective raw files. There are two main reasons why the models were separated. First, it was regarded as a measure to reduce complexity, potentially leading to improved prediction performance. Additionally, using separate models allows for independent feature selection, as the most important features may differ between CAD and PLM prediction.

The exploration of ensemble models consisted of three parts: the small dataset experiment, the medium dataset experiment, and the large dataset experiment. The methodology for each is presented in the following sections.

4.3.1 Small dataset experiment

The small dataset experiment was repeated with each newly developed preprocessing version, serving as a way to evaluate the impact of those changes. All applied preprocessing steps and versions can be found in Section 4.1. The small dataset initially consisted of 690 samples as presented in Table 4.2; however, due to filtering, the two preprocessing versions ultimately used for the small dataset experiment contained 455 and 133 samples, respectively. The difference

between the two lies in the filtering criteria applied. The first version contains all the function groups where a difference between the original CAD and PLM components' weight sums existed, and where the weights have been adjusted such that the difference is less than 500 grams. The second version contains all function groups where the difference between the original CAD and PLM components' weight sums is more than 200 grams, and where the weights have been adjusted such that the difference is less than 500 grams. Table 4.5 specifies the version differences. The motivation behind the various preprocessing changes can be found in Section 4.1. Both dataset versions contained 68,756 features once flattened, and a training/validation split of 80/20 was considered for the small test. The small test results for the five ensemble models can be found in Table 5.4 and Table 5.5.

Version	Samples	Original weight difference threshold
1	455	>0 grams
2	133	>200 grams

Table 4.5: Characteristics of the preprocessing versions used for the small dataset, the adjusted weight difference threshold is consistent across all the versions (<500 grams).

4.3.2 Medium dataset experiment

After obtaining the validation results from the small dataset experiment, two models were selected for further training and improvement. The chosen models were the Extra Trees Regressor and the XGBoost Regressor. By the time the selected models were ready to be trained on the medium dataset, new preprocessing changes had been introduced, leading to four separate test runs. At this stage, two major distinctions were identified as crucial for evaluation. The versions shown in

Table 4.5 remained, but a new modification was introduced: scaling before padding. This change was made after observing that some values did not seem to be affected by scaling at all. Again, more detailed motivation regarding the preprocessing approaches can be found in Section 4.1. The specifics of the four versions are displayed in Table 4.6. All versions flattened feature length is 79,773, and once again, a training/validation split of 80/20 was considered. The results for the selected ensemble models for these different versions of the medium dataset can be found in Table 5.6, Table 5.7, Table 5.8 and Table 5.9.

Version	Samples	Original weight difference threshold	Scaling
1	1191	>0 grams	before padding
2	1191	>0 grams	after padding
3	358	>200 grams	before padding
4	358	>200 grams	after padding

Table 4.6: Characteristics of the preprocessing versions used for the medium dataset, the adjusted weight difference threshold is consistent across all the versions (<500 grams).

After training both models on all versions of the medium dataset, feature selection was conducted using the XGBoost model trained on preprocessing version 2 from Table 4.6, as this combination of model and preprocessing version achieved the highest performance across all evaluation metrics. The feature selection process started with the top 10 features ranked by average importance scores. Since features are flattened during ensemble training, each feature's score represents the average importance of its flattened components. The model was then retrained iteratively, adding 5 more features in each round, to identify the point at which performance stagnates. Training with the top 10 features led to improved performance, prompting the decision

to further explore the impact of using even fewer than 10 features, and thus, using the top 2 and 5 features was considered as well. The results for training on selected features can be found in Table 5.10.

4.3.2.1 Testing of models trained on the medium dataset

In the small and medium tests, the models were trained on 80% of the data and validated on the remaining 20%. Validation results were promising, especially for the medium dataset, reaching an overall accuracy of 82% and a function group accuracy of 79%. However, when the same model was applied to predict on data from a car configuration that was not included in the medium dataset, its performance decreased significantly. The initial test data consisted of a dataset featuring all function groups of a single vehicle from the same car project as those found in the medium dataset. Given this similarity, the model was expected to perform well. Nevertheless, the results indicated that the model had very limited generalisation capacity.

To address the models' poor generalisation ability, new preprocessing methods were once again considered. Previously, datasets were padded at the feature level, ensuring that all samples had the same length for each feature. This length was determined by the sample with the longest value for a given feature. However, this approach resulted in varying lengths between features across datasets. Consequently, when creating a testing dataset using the same method, the model may have been trained on samples with feature lengths that did not match those of the test dataset. To be able to train the ensemble models, all features are flattened, and thus, this structuring of data may be misinterpreted by the model. To resolve this, it was decided to pad all features to the same length for both the training/validation

and testing datasets. Another important preprocessing adjustment involved ensuring that the features in both the training/validation and testing datasets were sorted in the same order. The motivation behind these two adjustments was to provide the model with more clarity and consistency, ultimately improving generalisation and performance.

When the changes above were implemented to preprocessing version 2 of the medium dataset, a new round of training was conducted for the XGBoost model. Since the training on selected features indicated that the model could perform well on 15 features, only these were considered. To test the model, it was decided to create three different testing frames, using the new preprocessing approach. Each of the testing frames consists of a full vehicle configuration, as in the initial testing. The first testing frame includes a vehicle configuration that is incorporated in the medium-sized dataset. This choice of testing does not align with general machine learning testing practice, where the data that the model is tested on shall not have been seen by the model before. However, testing with a complete vehicle configuration was deemed valuable as this is not the same as the validation procedure, which involved 20% of the data being randomly selected and does not necessarily represent a full vehicle. Furthermore, this first testing frame serves as a comparison for the other two. The second testing frame includes a vehicle configuration of the same car project that the medium dataset comprises. However, the exact configuration is not a part of the medium-sized dataset. The third testing frame is a vehicle configuration of a car project that is different from the ones in the medium dataset.

The testing results for the three different car configurations are presented in Table 5.11. These results indicate that the model is severely

overfitted to the training data. This can also be confirmed by the fact that the training accuracy is much higher than the validation accuracy. To reduce overfitting, hyperparameter tuning was performed on the XGBoost model. Table 4.7 presents the different model configurations that were tested. The configurations were applied in the same order as they appear in the table, and the changes were based on the results of the previous tuning. The default values for the hyperparameters that have been considered in the tuning follow:

- gamma = 0
- learning_rate = 0.3
- max_depth = 6
- min_child_weight = 1
- n_estimators = 100

Tuning	gamma	learning_rate	max_depth	min_child_weight	num_estimators
1	default	default	3	default	default
2	default	default	3	5	default
3	default	default	default	default	50
4	default	default	3	default	50
5	default	default	default	default	20
6	default	default	default	default	10

Table 4.7: Hyperparameter tuning of the XGBoost model trained on preprocessing version 2 of the medium dataset with new padding approach.

4.3.3 Large dataset experiment

After completing the medium training rounds and analysing the results, the XGBoost model was selected for training on the large dataset,

including data for all available car projects. The training was completed with the four different preprocessing versions presented in Table 4.8, and each run took between 6 to 19 hours to compute. The default parameters were used.

The results of the runs indicated room for improvement (see Section 5.2.3), and for this reason, additional training runs were executed, but this time with the dataset partitioned into subsets. One model was trained solely on data from a car project referred to as "Project 1", another was trained on data for "Project 2", and a third model was trained on data for "Project 3". All models were trained on the best-performing dataset from Table 4.6, namely version 4. The models were trained using the default hyperparameters, and the number of samples was 1,085 for "Project 1", 511 for "Project 2" and 534 for the dataset with the rest of the models. See Section 5.2.3 for the results.

Version	Samples	Original weight difference threshold	Scaling
1	4217	>0 grams	before padding
2	4217	>0 grams	after padding
3	1544	>200 grams	before padding
4	1544	>200 grams	after padding

Table 4.8: Characteristics of the preprocessing versions used for the large dataset, the adjusted weight difference threshold is consistent across all the versions (<500 grams)

The results from model training on the separated datasets remained unsatisfactory. For this reason, hyperparameter tuning of the XGBoost models was implemented. Since a training run with default parameters can require up to 19 hours of computational time, the possibility of investigating all hyperparameters was considered infeasible within the time limits, and the search was limited to the number of

estimators n_estimators. The tuning was performed with grid search, using different numbers of possible alternatives for n_estimators depending on the dataset. The number of estimators included in the grid search varied between 15 and 150.

Each of the three datasets were partitioned into two datasets: one for training/validation and another for testing. Subsequently, the training validation data was further split into one training set and one validation set. Both splits were performed using an 80/20 ratio. The test set was later used to create the data for evaluation, and the details of this are presented in the following subsection.

4.3.3.1 Testing of models trained on the large dataset

Following the model training, the three highest performing models were tested using a similar approach as in the medium dataset experiment. The reason for choosing three machine learning models for evaluation is that it was proven necessary to use one model for each of the car projects.

Two test sets were created to evaluate the performance of each of the three machine learning models. The two test sets contained the entire collection of components belonging to a specific vehicle configuration. The first test set contained components included in the training datasets, and the second one comprised components belonging to a vehicle of the same model, but where the components were not included in the training. Testing of the models on a vehicle from a different project can be viewed as preferable to truly evaluate the generalisation abilities of a machine learning model. However, this was infeasible due to a mismatch in dimensionality of the input data. To clarify the testing process, consider the final XGBoost model trained

on data from "Project 1". This model will be tested twice. The first test set will only include components from a vehicle configuration included in the training set, and the second test will be on data not included in the training set. Both test sets include vehicle configurations from the same car project. The reason for testing using two different datasets is to reveal potential over- or underfitting. A high accuracy for a vehicle included in the training data, combined with a low accuracy for the vehicle not included in the training data, can indicate poor model generalisation. The results of the tests are found in Section 5.2.3.

4.4 Evaluation metric

An important part when exploring various machine learning models is to find suitable evaluation metrics. The standard measure for the evaluation of regression models is the loss. However, the loss in this task is significantly affected by the numerous padded zeros in the label vectors. As an example, assume that the label vector has the format illustrated in Equation 4.1. The vector is of dimension (1, 1079) and consists solely of two weights, 50 and 28, followed by 1077 padded zeros.

$$y_n = \begin{bmatrix} 50 & 28 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \quad \in \mathbb{R}^{1\times 1079} \tag{4.1}$$

Using MAE as the loss function, a predicted vector consisting solely of zeros would result in an average loss of 0.0723 grams, which significantly underrepresents the true prediction error. To address this, the padded zeros are excluded when computing the loss, leading to a more meaningful and interpretable evaluation metric. This metric was used

when evaluating neural networks and is abbreviated MAE Alt. in the results.

If the model correctly predicts all component weights within a function group, the requirement of a maximum 500-gram difference between the CAD and PLM components' weight sums is implicitly fulfilled, as the adjusted weights serve as targets. Based on this, it was considered appropriate to include accuracy as an evaluation metric for this task, despite its unconventional use in regression tasks. The accuracy is the proportion of correctly predicted weights among all predicted weights. A weight is considered correctly predicted if its predicted value differs from the true (the adjusted weight value) by no more than 1 gram. Again, the padded values are excluded. This accuracy metric was used for the evaluation of neural networks and ensemble models and is abbreviated as Acc. in the results. However, in the sections related to the ensemble models, this metric is also referred to as the overall accuracy.

During the evaluation of the ensemble models, a third metric was introduced: the function group accuracy, abbreviated Acc. FG in the results. The function group accuracy is the proportion of correctly predicted function groups among all function groups. A function group is considered correctly predicted if all its component weights are correctly predicted. Table 4.9 demonstrates the function group accuracy. Where function group 1 is assessed as correctly predicted, whilst function group 2 is not, due to two of the CAD predictions not being within the 1-gram difference to the true value limit. Padded values are excluded.

FG	predicted component weights	true component weights	
1 (CAD)	[1, 2.5, 12.66, 13, 896]	[1, 3, 13, 13, 896]	correct
1 (PLM)	[1, 3, 12.5, 13, 896]	[1, 3, 13, 13, 896]	
2 (CAD)	[53.5, 57, 100, 100]	[55, 55, 100, 100]	not correct
2 (PLM)	[55, 55, 100, 100]	[55, 55, 100, 100]	

Table 4.9: Fictitious example of the Acc. FG metric.

Lastly, the mean absolute error per function group was also used in ensemble model evaluation, abbreviated as MAE/FG in the results. The absolute error for a function group is calculated by summing the errors of all its components, where the error for a single component is the difference between its predicted and true values. The absolute errors for all function groups are then averaged, with padded values excluded from the calculation.

4.4.1 Sufficient accuracy

The final model aims to output the correct adjusted weights, which means that the weights that the model did not predict correctly need to be manually adjusted if the function group difference is still more than 500 grams. Due to this, accuracy metrics were valued highly when analysing the results from the different models. The fewer weights that need to be adjusted afterwards, the better the model is for this application. However, the mean absolute error per function group was also deemed a crucial metric to consider within the tests. Ideally, the mean absolute error per function group would be zero, since this means that the requirement of a 500-gram difference on the function group level would be fulfilled, as the adjusted weights serve as the target values.

During model development, it was determined that a model must achieve a minimum function group accuracy of 70% during testing to be considered sufficiently accurate. The anticipated performance on inference data is significantly lower, since the inference data comprises new car configurations. Therefore, a high threshold for testing accuracy is required. This benchmark was established since a lower performance would not justify the extensive implementation and transition efforts required to move from the manual data harmonisation.

4.5 Final pipeline for data harmonisation

In parallel with model development, a pipeline for computing data harmonisation and calculations of axial weight distributions was built. The best-performing model was planned to be implemented as a part of a final pipeline to compute the adjusted weights necessary for the calculations. The pipeline for data harmonisation takes two raw data files as input: one CAD and one PLM file. The data is then preprocessed using the same steps as the training data. The samples with correct weights, where the sum of the component weights in CAD and PLM data align, are filtered out. Components that require adjustments are supposed to be handled by the machine learning model. After model inference, the weight predictions are visualised using Streamlit [58]. A subsequent and important step is for the user to manually handle the incorrect model predictions and overlook the correct ones to guarantee the quality of the weights. To simplify for the user, the predictions are evaluated using the difference between the predicted CAD and PLM components' weight sums. If the difference is greater than a set threshold, the predicted weights are highlighted as incorrect. After being reviewed and potentially adjusted, the weights

are saved in a CSV file. Following this step, the weights are used as input for axial weight distribution calculations and centre of gravity prognosis. The final pipeline is illustrated in Figure 4.3.

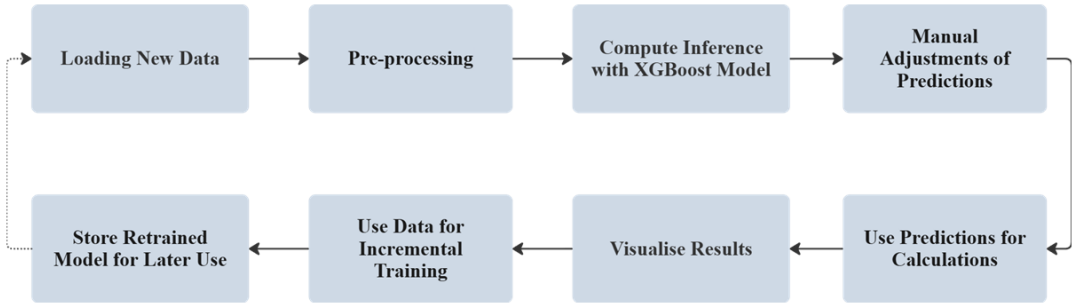

Figure 4.3: An illustration of the final pipeline for computing centre of gravity prognosis and axial weight distribution calculations.

Despite the numerous attempts to develop a machine learning model qualified for the task, the final decision was taken that no model met the requirements. The time and effort required for the deployment of the final tool was believed to exceed the benefits. For this reason, no model was implemented to be part of the final pipeline for axial weight distributions. The pipeline, however, was finalised with the exception of the machine learning model for weight predictions.

5

Results

This chapter presents the results of the thesis, including outcomes from neural networks and ensemble models.

5.1 Neural network models

As described in the Methods chapter, the first machine learning model implemented in this project was a single-layer feedforward neural network. The model was trained with several hyperparameter configurations, and the results of this can be found in Table 5.1. The results are sorted based on validation accuracy, in ascending order. The models were evaluated with linear activation function, ReLU and Leaky ReLU, and the potential activation functions were applied to all layers. The training trials are hereby referred to as experiments.

N	**B.S.**	**Opt.**	**Loss Fn**	**L.R.**	**Activation**	**Loss**	**Acc.**
2048	32	RMSprop	MAE	0.0005	Leaky ReLU	23094.7	0.00
4096	128	SGD	MAE Alt.	0.0001	Linear	403.4	0.01
4096	64	ADAM	MAE Alt.	0.0001	Linear	347.1	0.01
512	64	SGD	MAE Alt.	0.0005	Leaky ReLU	334.9	0.01
2048	256	ADAM	MAE Alt.	0.0005	Leaky ReLU	259.5	0.01
1024	32	ADAM	Huber	0.0001	ReLU	14.1	0.13
512	32	ADAM	MAE Alt.	0.0001	ReLU	205.2	0.13
1024	256	RMSProp	MAE	0.0005	ReLU	17.2	0.13
512	128	RMSProp	Huber	0.0001	ReLU	35.4	0.14

Table 5.1: Results for training of the single-layer feedforward neural network model. All experiments were executed for a maximum of 800 epochs, on the medium-sized training set with 1,368 samples. The dimensions for input and output were 79,961 and 1,260, respectively. N represents the number of hidden neurons in the network, B.S. is short for batch size, Loss Fn represents the loss function, L.R. is the learning rate, and Activation is the activation function applied to all layers. The loss and the accuracy (Acc.) correspond to the final validation loss and the final validation accuracy achieved during training.

As demonstrated by the results in Table 5.1, the model appearing last in the table achieved the highest accuracy, more specifically 14%. This means that 14% of the predicted weights corresponding to non-zero labels are correct, which does not meet the expected levels. With this, it became apparent that a more advanced model was required to fit the data successfully. Subsequently, experiments that included models with more hidden layers were executed, and the results of these can be found in Table 5.2 and Table 5.3. The first table presents the results for models that incorporate two hidden layers, and the latter models with five hidden layers. To reduce the number of possible combinations of model architecture, all layers in one network comprise the same number of neurons. The performance of the models with two hidden

layers was considerably higher - the maximum validation accuracy achieved in hyperparameter tuning was approximately 46 %. The accuracy curve of the best-performing model can be found in Figure 5.2.

N	**B.S.**	**Opt.**	**Loss Fn**	**L.R.**	**Activation**	**Loss**	**Acc.**
1024	32	ADAM	Huber	0.001	Linear	134.8	0.01
512	64	ADAM	MAE	0.001	Linear	23.0	0.04
4096	256	SGD	MAE Alt.	0.0001	Leaky ReLU	370.9	0.1
4096	256	ADAM	Huber	0.0001	Leaky ReLU	17.0	0.23
2048	128	RMSProp	Huber	0.0001	Leaky ReLU	16.9	0.32
1024	256	RMSProp	MAE	0.001	Leaky ReLU	15.7	0.39
2048	256	RMSProp	MAE Alt.	0.0005	ReLU	331.6	0.46

Table 5.2: Results for training of the feedforward neural network with two hidden layers. The results are sorted based on validation accuracy, in ascending order. All experiments were executed for a maximum of 800 epochs, on the medium-sized training set with 1,368 samples. The dimensions for input and output were 79,961 and 1,260, respectively. N represents the number of hidden neurons in the network, B.S. is short for batch size, Loss Fn represents the loss function, L.R. is the learning rate, and Activation is the activation function applied to all layers. The loss and the accuracy (Acc.) correspond to the final validation loss and the final validation accuracy achieved during training. The SGD optimiser, and the MSE loss functions were found to generate unsatisfactory results in pilot runs and were thereby neglected in the hyperparameter tuning.

N	**B.S.**	**Opt.**	**Loss Fn**	**L.R.**	**Activation**	**Loss**	**Acc.**
1024	256	RMSprop	Huber	0.0005	Linear	343808.9	0.00
1024	32	RMSprop	MAE	0.001	Leaky ReLU	16.8	0.13
1024	128	RMSprop	MAE Alt.	0.001	ReLU	366.2	0.14
512	128	ADAM	MAE	0.001	Leaky ReLU	17.5	0.15
4096	256	ADAM	MAE	0.0005	Leaky ReLU	17.4	0.17

Table 5.3: Results for training of a feedforward neural network, with five hidden layers. Each of the experiments was executed for a maximum of 800 epochs, on the medium-sized training set, containing 1,368 samples. The dimensions input and outputs were 79,961 and 1,260, respectively. The loss and the accuracy (Acc.) correspond to the final validation loss and the final validation accuracy achieved during training. The SGD optimiser, and the MSE loss functions were found to generate unsatisfactory results in pilot runs and were thereby neglected in the hyperparameter tuning.

When interpreting the results from the experiments, it is important not to focus too narrowly on the loss. The reason for this is explained in detail in Section 4.4, but in short, this is due to the extensive padding of the output vector. However, the loss can still serve as an indicator of the magnitude of the errors, especially when combined with accuracy. Since different loss functions are used for each model - MAE, MSE, Huber loss and a modification of MAE - one should also be careful when directly comparing loss values between the different models. All losses are calculated as the average discrepancy between the predicted output weights and the targets, per sample. Hence, an MAE loss of 17.2 stands for an average discrepancy of 17.2 grams between the output and the target. The modification of MAE, also referred to as MAE Alt., neglects all zero values when calculating the loss. This is also sub-optimal since some labels are zero.

To fully analyse the performance of a neural network model config-

uration, one must consider not only the final metrics but also the performance curves. One desirable characteristic to observe in model training is that the two loss curves align closely over time. In addition, the training should be terminated when the performance curves stagnate. Nonetheless, an occasionally observed phenomenon in the experiments in this project was a validation loss significantly smaller than the training loss. This was, for example, the case for the best-performing model, which can be seen in Figure 5.1. From the figure, one can tell that the model loss is reduced iteratively. However, the validation loss is almost exclusively lower than the training loss.

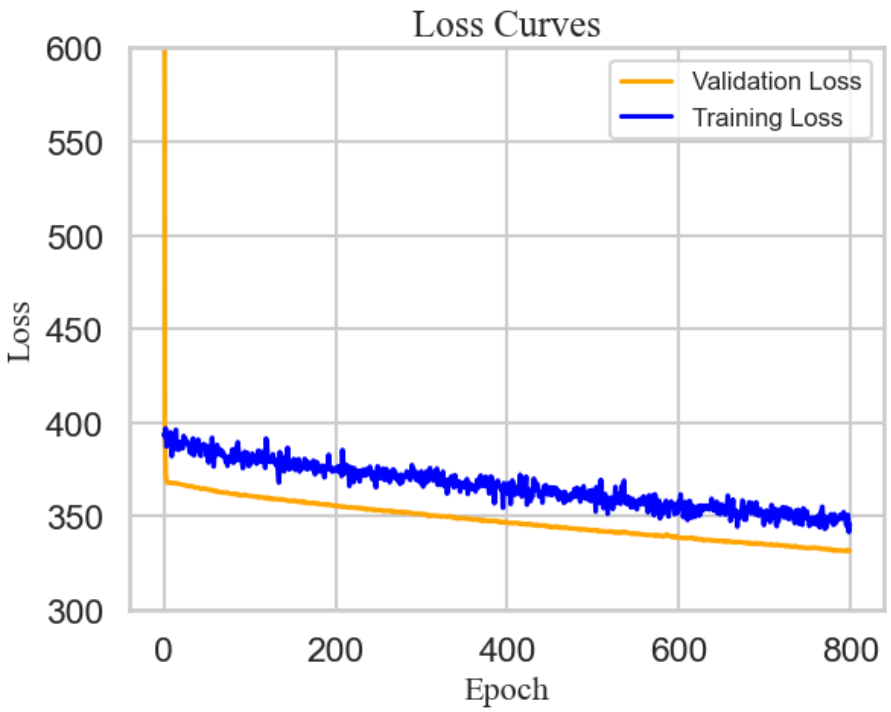

Figure 5.1: The loss curves of an experiment with a two-layer neural network.

Another observation that can be made from Figure 5.1 is that neither of the loss curves has reached a plateau, so further model improvements appear to be possible. For this reason, a prolonged run using the same hyperparameters was executed. The results from this run

showed that even though the loss continued to decrease until stagnation around 7,000 epochs, the accuracy did not increase. Rather, the accuracy decreased after reaching its maximum of approximately 40% at around 800 epochs (see Figure 5.3).

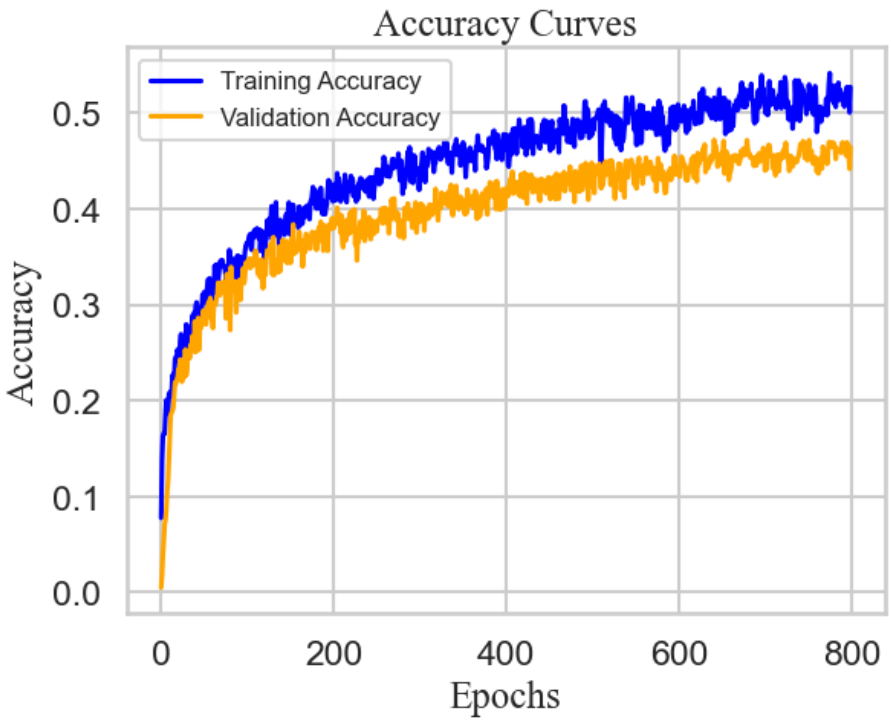

Figure 5.2: The accuracy curves of the experiment with the best performing neural network model, with two layers.

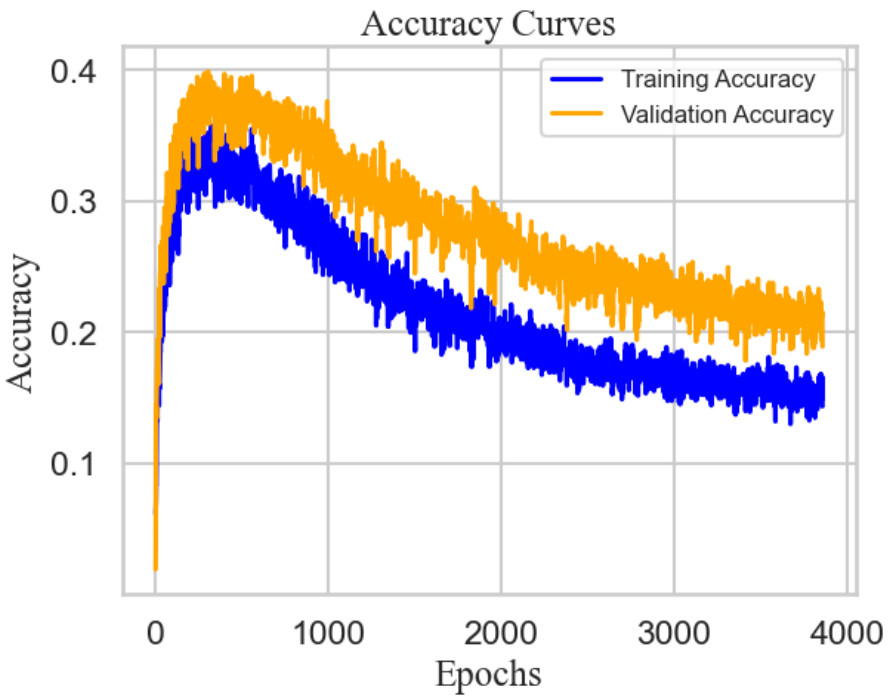

Figure 5.3: The accuracy of elongated run with optimal hyperparameters.

The results from the neural network models were not satisfactory, and instead, the focus was shifted toward ensemble models. The results from the implementation of ensemble models will be presented in the next section.

5.2 Ensemble models

This section presents the results of the various ensemble model experiments conducted throughout this thesis: the small dataset experiment, the medium dataset experiment, and the large dataset experiment.

5.2.1 Small dataset experiment

The final small dataset experiment was conducted for five different regressors, and two versions of preprocessing, described in detail in Section 5.2.1. The metrics evaluated are overall accuracy, function

group accuracy and mean absolute error per function group. An in-depth explanation of the metrics can be found in Section 4.4, and all provided metrics are for the validation data. Table 5.4 and Table 5.5 present the small dataset experiment results.

Model	Acc.	Acc. FG	MAE/FG
AdaBoost	0.200	0.000	9752.6
Extra Trees	0.784	0.714	8595.8
Gradient Boosting	0.721	0.159	8568.9
Random Forest	0.439	0.027	9881.1
XGBoost	0.807	0.703	9068.1

Table 5.4: Ensemble models results using preprocessing version 1 from Table 4.5.

Model	Acc.	Acc. FG	MAE/FG
AdaBoost	0.305	0.037	2975.0
Extra Trees	0.801	0.667	3283.5
Gradient Boosting	0.801	0.500	3252.0
Random Forest	0.260	0.000	6413.6
XGBoost	0.825	0.685	3124.0

Table 5.5: Ensemble models results using preprocessing version 2 from Table 4.5.

Based on the results of the small dataset experiment, it became evident that training the models with preprocessing version 2 enhanced performance. All models, except the Random Forest Regressor, achieved higher overall accuracy. For the function group accuracy, the results were mixed. However, the most significant improvement was observed in the mean absolute error per function group, where preprocessing version 2 reduced errors to between one-third and one-half compared to preprocessing 1.

The main purpose of the small dataset experiment was to decide which

models to use for further training and model improvement. Of the five models explored, two were selected for training on the medium-sized dataset, namely, the Extra Trees Regressor and the XGBoost Regressor. For both preprocessing versions, the AdaBoost Regressor and the Random Forest Regressor produced undesirable results, unable to predict any of the function groups from the validation data correctly. While there were some improvements with the second version of the dataset, these two models still performed significantly worse than the other three. The Gradient Boosting Regressor was not selected for further training due to its lower performance in function group accuracy, despite having similar overall accuracy and mean absolute error per function group as the Extra Trees Regressor and XGBoost Regressor. Another observation was the difference in training times among the various models. For preprocessing version 1, training times ranged from approximately one to five hours, while for preprocessing version 2, they varied between two and eleven hours. The AdaBoost Regressor and XGBoost Regressor were significantly faster, taking just over an hour for the first version and just over two hours for the second version. In contrast, the Extra Trees Regressor, Gradient Boosting Regressor, and Random Forest Regressor took more than four and ten hours, respectively.

5.2.2 Medium dataset experiment

As explained in Section 4.3.2, four different preprocessing versions of the medium-sized dataset were used to train the selected models: the Extra Trees Regressor and XGBoost Regressor. The results are presented in Table 5.6, Table 5.7, Table 5.8 and Table 5.9. The same evaluation metrics used in the small dataset experiment were applied, and again, all provided metrics are for the validation data.

Model	Acc.	Acc. FG	MAE/FG
Extra Trees	0.811	0.780	7295.9
XGBoost	0.819	0.780	6206.0

Table 5.6: Ensemble models results using preprocessing version 1 from Table 4.6.

Model	Acc.	Acc. FG	MAE/FG
Extra Trees	0.812	0.782	7310.0
XGBoost	0.822	0.789	6202.9

Table 5.7: Ensemble models results using preprocessing version 2 from Table 4.6.

Model	Acc.	Acc. FG	MAE/FG
Extra Trees	0.788	0.757	15079.8
XGBoost	0.795	0.750	13937.4

Table 5.8: Ensemble models results using preprocessing version 3 from Table 4.6.

Model	Acc.	Acc. FG	MAE/FG
Extra Trees	0.787	0.750	15738.8
XGBoost	0.798	0.750	13930.8

Table 5.9: Ensemble models results using preprocessing version 4 from Table 4.6.

When training the models on the small dataset, it was the preprocessing version with the original weight threshold of 200 grams that yielded significantly lower MAE/FG across all six models. However, when analysing the medium dataset experiment results, the trend is reversed, and it is the preprocessing versions with the original weight threshold of 0 grams that yield remarkably lower MAE/FG values. It is important to note, though, that the lowest MAE/FG achieved for the medium dataset is still considerably higher than the lowest MAE/FG observed with the small dataset. However, training on the

medium-sized dataset did increase the performance in terms of the two other metrics, that is, overall accuracy and function group accuracy. The most remarkable difference is observed for the function group accuracy, where the Extra Trees Regressor achieved at best 71% on the small dataset and 78% on the medium dataset, and the XGBoost Regressor achieved at best 70% on the small dataset and 79% on the medium dataset.

When comparing the results between the two models, there are no major differences. XGBoost performs marginally better across all metrics. However, considering the substantial disparity in training times, where the Extra Trees Regressor took 48-51 hours to train on the versions with 1,191 samples, compared to XGBoost's 7-10 hours, it made XGBoost the preferred model for use on the larger dataset.

The results from training the XGBoost Regressor on preprocessing version 2 of the medium-sized dataset with selected features can be found in Table 5.10. The results are visualised in Figure 5.4. All provided metrics are for the validation data.

Features	**Acc.**	**Acc. FG**	**MAE/FG**
Top 2	0.812	0.435	6272
Top 5	0.841	0.705	5872
Top 10	0.843	0.751	5907
Top 15	0.844	0.787	5871
Top 20	0.841	0.780	5900
Top 25	0.843	0.780	5913
Top 30	0.842	0.778	5926

Table 5.10: Results from training the XGBoost with selected number of features.

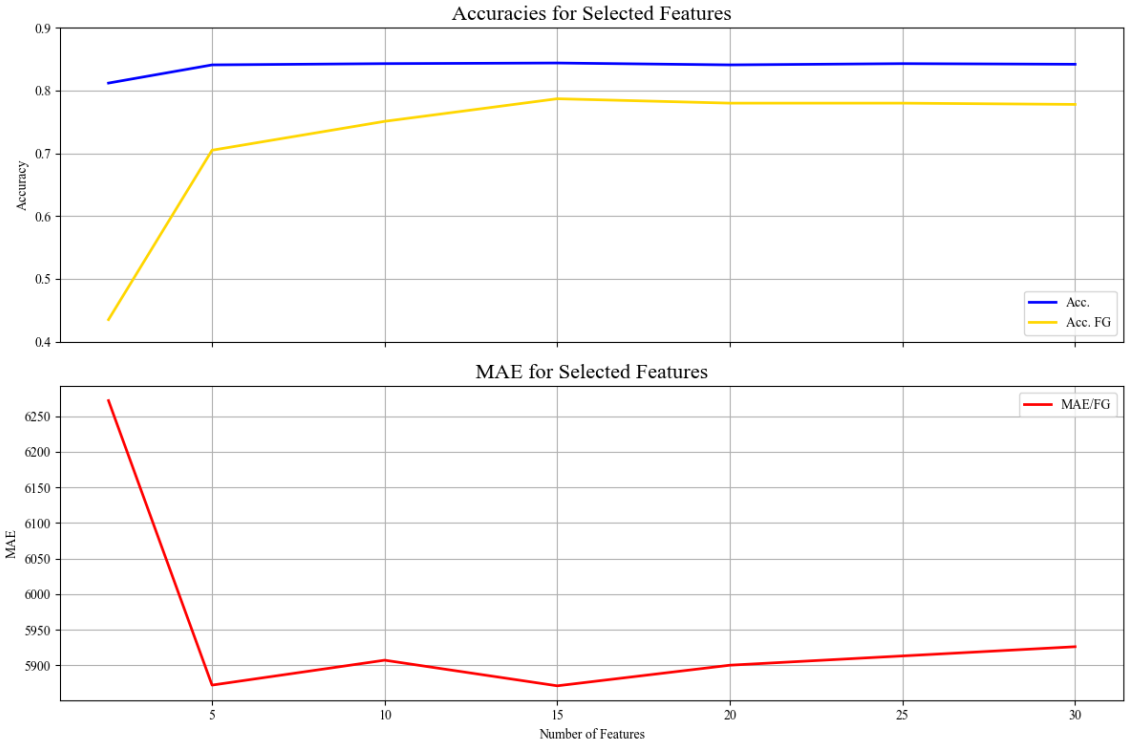

Figure 5.4: The first plot shows the overall accuracy (Acc.) and function group accuracy (Acc. FG), and the second plot shows the average mean absolute error per function group for training the XGBoost model on a selected number of features.

From the validation results of the trained model with selected features, it becomes clear that it does not need the full set of approximately 150 features; rather, its performance even improves with a much smaller proportion of features. Analysis of the plots in Figure 5.4 indicates that, for the medium dataset and preprocessing version 2, the optimal number of features for training the XGBoost model is 15. Beyond this point, both accuracy metrics stagnate, while the average mean absolute error per function group begins to increase.

5.2.2.1 Testing of models trained on the medium dataset

The results from testing the XGBoost model trained on preprocessing version 2 of the medium-sized dataset are presented below. The model

was trained using the top 15 features. Explanation and motivation for choices of testing frames are presented in Section 4.3.2.1.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.913	0.795	2116.9
"Project 1"	no	0.255	0.000	42989.1
"Project 2"	no	0.259	0.000	21489.9

Table 5.11: Results from testing the XGBoost model on three different full car configurations.

The testing results indicate that the model severely overfits the training data, as it performs very well on the full vehicle configuration included in the training set. This outcome is expected. However, there is a drastic performance drop when evaluating different car configurations, even though new preprocessing approaches were incorporated to combat this. It was expected that the model would perform best on the first configuration, slightly worse on the second, and worst on the third. However, the results did not follow this anticipated pattern, as the second configuration yielded the poorest performance.

Once the model was retrained with different hyperparameter configurations, the same testing was conducted. The tuned hyperparameters and their values can be found in Table 4.7. The testing results with tuned hyperparameters are presented in Table 5.12, Table 5.13, Table 5.14, Table 5.15, Table 5.16 and Table 5.17.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.781	0.114	2189.3
"Project 1"	no	0.199	0.000	42950.8
"Project 2"	no	0.231	0.000	21384.0

Table 5.12: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 1 from Table 4.7.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.448	0.010	2757.6
"Project 1"	no	0.095	0.000	48238.7
"Project 2"	no	0.118	0.000	22871.3

Table 5.13: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 2 from Table 4.7.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.884	0.395	2126.7
"Project 1"	no	0.255	0.000	42990.2
"Project 2"	no	0.260	0.005	21489.5

Table 5.14: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 3 from Table 4.7.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.573	0.014	2283.4
"Project 1"	no	0.202	0.000	42935.5
"Project 2"	no	0.233	0.000	21369.1

Table 5.15: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 4 from Table 4.7.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.641	0.033	2374.525
"Project 1"	no	0.266	0.000	43081.1
"Project 2"	no	0.280	0.000	21507.1

Table 5.16: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 5 from Table 4.7.

Car Model	In Medium Dataset	Acc.	Acc. FG	MAE/FG
"Project 1"	yes	0.283	0.019	3849.1
"Project 1"	no	0.144	0.000	44066.9
"Project 2"	no	0.174	0.000	21776.6

Table 5.17: Results from testing the XGBoost model on three different full car configurations, with hyperparameter tuning 6 from Table 4.7.

None of the hyperparameter tuning efforts successfully mitigated overfitting. Although a slight improvement in overall accuracy was observed for testing frames 2 and 3 in the fifth tuning configuration, the overall testing performance remains far below expectations.

5.2.3 Large dataset experiment

The results of model training on the large dataset, including all car projects, are found in Table 5.18. In addition, an attempt was made to train separate XGBoost models for the different car projects. These results are found in Table 5.19. In this table, the training accuracy was included to demonstrate the occurrence of overfitting, which was the case for all experimental runs. Due to the presence of overfitting, tuning of the number of estimators in the ensemble (n_estimators) was carried out to reduce the model complexity. The results from these experiments can be found in Tables 5.20, 5.21 and 5.22.

Version	Hyperparameters	Acc.	Acc. FG	MAE/FG
1	Default	0.287	0.119	14282.9
2	Default	0.318	0.141	17814.1
3	Default	0.483	0.240	19453.7
4	Default	0.487	0.250	19444.9

Table 5.18: Results from the XGBoost models trained on different preprocessing versions of the large dataset.

Dataset	Acc.	Acc. FG	MAE/FG	Tr. Acc.
"Project 1"	0.60	0.287	19420.4	0.767
"Project 2"	0.60	0.317	24622.8	1.000
"Project 3"	0.198	0.157	15720.2	0.918

Table 5.19: Results from the XGBoost models trained on data for different car models separated.

Dataset	n_estimators	Acc.	Acc. FG	MAE/FG	Tr. Acc.
"Project 1"	150	0.484	0.230	27466.2	-
"Project 1"	50	0.601	0.287	15330.8	0.761
"Project 1"	35	0.455	0.009	25951.6	0.650
"Project 1"	20	0.318	0.000	19692.6	0.380

Table 5.20: Results from training XGBoost models on all available data for "Project 1", using different numbers of estimators n_estimators. "Acc." corresponds to the achieved validation accuracy during evaluation, and "Tr. Acc" represents the training accuracy.

Dataset	n_estimators	Acc.	Acc. FG	MAE/FG	Tr. Acc.
"Project 2"	50	0.602	0.287	15330.8	0.900
"Project 2"	35	0.558	0.152	15343.0	0.996
"Project 2"	15	0.431	0.012	15680.6	0.996

Table 5.21: Results from training XGBoost models on all available data for "Project 2", using different numbers of estimators n_estimators. "Acc." corresponds to the achieved validation accuracy during evaluation, and "Tr. Acc" represents the training accuracy.

Dataset	n_estimators	Acc.	Acc. FG	MAE/FG	Tr. Acc.
"Project 3"	50	0.149	0.156	21354.1	0.930
"Project 3"	35	0.151	0.167	21350.3	0.803
"Project 3"	20	0.211	0.156	15695.2	0.516

Table 5.22: Results from training XGBoost models on all available data for "Project 3" using different numbers of estimators n_estimators. "Acc." corresponds to the achieved validation accuracy during evaluation, and "Tr. Acc" represents the training accuracy.

In the tables above, it can be observed that for both "Project 1" and "Project 2", the models with 50 n_estimators achieved the highest validation accuracy. In addition, overfitting can be observed in nearly all of the models. The model for "Project 1" was the sole model where training and validation accuracy moderately aligned. The best-performing model for "Project 3" was the model with 15 estimators, achieving a validation accuracy of 21%. These models were evaluated on test vehicles, and the results are presented below.

5.2.3.1 Testing of models trained on the large dataset

The best model for each of the car projects was tested on two full vehicle configurations, as described in Section 4.3.3.1. The results

from the evaluation of the "Project 1" model are found in Table 5.23, the results from "Project 2" in Table 5.24 and the remaining models in Table 5.25. To test the models on vehicles from another car project was not possible due to a mismatch in input dimensions.

Car Model	**In Large Dataset**	**Acc.**	**Acc. FG**
"Project 1"	yes	0.289	0.004
"Project 1"	no	0.332	0.008

Table 5.23: Results from testing the XGBoost model trained on the large dataset for "Project 1" on two different full vehicle configurations. "Acc." refers to testing accuracy.

Car Model	**In Large Dataset**	**Acc.**	**Acc. FG**
"Project 2"	yes	0.795	0.435
"Project 2"	no	0.410	0.222

Table 5.24: Results from testing the XGBoost model trained on "Project 2" on two different full vehicle configurations. "Acc." refers to testing accuracy.

Car Model	**In Large Dataset**	**Acc.**	**Acc. FG**
"Project 3"	yes	0.312	0.006
"Project 3"	no	0.167	0.000

Table 5.25: Results from testing the XGBoost model trained on the dataset from "Project 3" on three different full vehicle configurations. "Acc." refers to testing accuracy.

6

Discussion

From the conducted experiments, it can be concluded that the ensemble models generally outperformed the neural networks. The highest validation performance was achieved by the XGBoost model trained on preprocessing version 2 of the medium dataset, reaching an overall accuracy of 84% (see Table 5.10). Despite this, none of the models achieved satisfactory test results. The rationality of the results is discussed in the following sections, including the strengths and weaknesses identified during the process.

6.1 Results of neural network models

When the neural network models were implemented, two main issues were identified. First, the models did not achieve desirable results, and secondly, the validation loss was lower than the training loss for some model configurations. Plausible causes of these observations will be evaluated in the following subsections.

6.1.1 Unfavourable results

The unfavourable results of the neural network models can be due to several factors. One main concern is that the purpose of the model is to detect fine distinctions in the data. To demonstrate this, an example

is provided. Consider the unpadded vector with the input weights in equation 6.1, and the corresponding target values in equation 6.2.

$$y_n = \begin{bmatrix} 0 & 0 & 1 & 2 & 16 & 103 & 1264 & 30157 \end{bmatrix} \quad \in \mathbb{R}^{1\times 7} \tag{6.1}$$

$$y_n = \begin{bmatrix} 0 & 0 & 1 & 2 & 16 & 103 & 5264 & 38157 \end{bmatrix} \quad \in \mathbb{R}^{1\times 7} \tag{6.2}$$

The numerical difference between the two vectors is minimal; only two numbers differ. The magnitude of the difference is, by contrast, more significant. Neural networks are perceived as strong when it comes to detecting fine details. However, the model's performance depends heavily on the hyperparameters and the quality of the data. In this project, the data can most certainly be considered noisy, since several errors in the input data will occur, and this will worsen the model's performance. Furthermore, as frequently emphasised earlier in this report, neural network models are sensitive to hyperparameter tuning and the choice of feature engineering. Not all potential combinations of hyperparameters have been investigated, and there remains a possibility that the optimal configurations of hyperparameters for this use case have not yet been found. Regarding the feature engineering, a substantial amount of time and effort was put into finding the optimal methods for scaling, encoding and processing the features for model training, but also this was restricted to a finite number of options as a result of time constraints.

Convolutional Neural Networks (CNNs) are commonly more proficient in detail detection in regression tasks, compared to feedforward neural networks. Nevertheless, due to time limitations and the massive

dataset handled in this project, CNNs were not implemented. Another possible reason for the mediocre results of the neural network models is that an early version of the preprocessing pipeline was used. This pipeline did, for example, include LabelEncoder instead of OrdinalEncoder, and it did not exclude the samples with incorrectly adjusted label weights. In this context, incorrect refers to a discrepancy between the CAD and the PLM weights larger than 500 grams, which can occur when the calculations were not intended to apply to all function groups.

6.1.2 Validation loss exceeds training loss

The second issue worth discussing, the divergence in loss curves, indicates an error in model implementation, data leakage, or that the validation data is significantly less complex than the training data. In this project, the implementation of the model was carefully investigated and benchmarked against similar deployments, leading to the latter two being the most reasonable causes. Since the input weights are repeatedly similar to the target values, data leakage is possibly an issue. This is not data leakage in the conventional sense, where data that is inaccessible during inference is used as input to the model. Instead, this can be considered data leakage since, for some samples, the input data corresponds exactly to the target data. Additionally, there is a possibility that the validation data is occasionally less complex than the training data. The validation set is smaller than the training set, hence, there is a greater possibility for complex samples to appear in the training data. Samples with target weights that differ significantly compared to the input weights can be seen as complex in this case.

Regularisation techniques and data augmentation are additional factors that can cause the appearance of a validation loss lower than the training loss. However, this does not apply to this project since neither regularisation techniques nor data augmentation were applied to the data.

6.2 Results of ensemble models

The following section discusses the ensemble model results, covering topics such as hyperparameter tuning, the transparency of ensemble models, and the unsatisfactory test outcomes.

6.2.1 Hyperparameter tuning

When conducting the small and medium dataset experiments, all models were trained with their default parameters. Hyperparameter tuning was not performed until a single model and preprocessing version were chosen, meaning that only the XGBoost model underwent hyperparameter optimisation. The main motivation behind this choice was the extensive training times, and to perform grid search for hyperparameter tuning for all the models and preprocessing versions would not have been feasible within the time frame of this thesis. Due to this scoping, one can not exclude that there exist configurations of the other five ensemble models, excluded during the tests, that could have performed better than the final version of the XGBoost model.

6.2.2 Transparency

Generally speaking, ensemble models provide limited insights into the training process. Unlike neural networks, where training can be consistently monitored and evaluated for each epoch, ensemble models

do not offer the same level of transparency. As ensemble models rely on aggregating the results of weak learners, tracking their progression and understanding their behaviour becomes challenging. As a result, it becomes difficult to identify potential adjustments or improvements when unsatisfactory results are achieved. However, hyperparameter tuning and result analysis can provide some insight into ensemble models, ideally by hyperparameter optimisation methods, such as grid search. Nevertheless, due to time-intensive model training, only a limited set of hyperparameters and values was tuned during the medium and large dataset experiments.

6.2.3 Evaluation

During the validation phase of the medium dataset experiment, the XGBoost model trained on the preprocessing version 2 achieved the best results, with an overall accuracy of 84% and a function group accuracy of 78%. However, the testing results were less encouraging. Overall accuracy and function group accuracy for the three car configurations within the initial testing were 91% and 80%, 26% and 0% and 26% and 0%. This indicates that the model could not produce satisfactory results for vehicles it had not been trained on, independent of the believed similarities to the training data. Due to these results and the fact that the training accuracy was higher than the validation accuracy, it became evident that the model had been overfitted to the training data. Consequently, various attempts to reduce the overfitting were conducted by preprocessing changes and hyperparameter tuning. Regardless of these actions, testing results remained undesirable. If overfitting were the only issue, reducing model complexity would likely result in slightly lower performance on the car configuration that was a part of the training data, while the model's

performance on the two other car configurations, which were not part of the training data, would improve. However, this trend was not observed. Instead, all testing results worsened. This suggests that with reduced model complexity, the model could not even predict the instances of the training data, and generalisation had not been improved.

By analysis of the medium dataset experiment and its outcomes, it was believed that these results might have been a consequence of insufficient training data. Nevertheless, the models trained on the large dataset, the "Project 2" dataset and the "Project 3" dataset showed poor generalisation abilities too. A test accuracy for unseen data remarkably higher than the training accuracy was observed across nearly all training sessions. When less complex models were examined, it became evident that the simpler models were incapable of capturing data patterns at all. One can consider the models to be of high variance, meaning that they are sensitive to the specific training data. This implies that if the data changes slightly, the predictions become completely different. Gathering all these results, it can be concluded that the issue is not in the lack of training samples per se, but rather it can be explained by the large variation in samples. The issue is further elaborated in Section 6.4.

Even though the results of the ensemble models do not meet the expected performance, it can be concluded that ensemble models are preferred over neural networks for this specific task and data. Although neural networks were not explored with the final versions of preprocessing, they performed remarkably worse with consistency during all previous preprocessing versions. Ensemble models commonly outperform neural networks when the dataset is sparse and when the

data relationship is irregular; thus, the results are reasonable.

6.3 Limitations of the preprocessing

The presented results are based on a set of selected preprocessing versions that showed the most promising performance during development. Although many different approaches had been tested, the versions that ultimately were used were those that had generated the best performance so far. Since preprocessing strategies were evaluated iteratively, some approaches showed potential at different stages. This suggests that further exploration of ways to preprocess the data could still lead to improvement.

As discussed in the results section, the effectiveness of the different preprocessing strategies varied across datasets. Between the small and medium datasets for ensemble models, the results were reversed. For the small dataset, which includes four car configurations, the preprocessing version using a 200-gram original weight difference threshold performed better. In contrast, the 0-gram threshold yielded superior results for the medium dataset, which comprises ten car configurations. For the large dataset, comprised of all available car configurations, the results are more diverse. Preprocessing versions 1 and 2 yield lower losses, whilst versions 3 and 4 are better in terms of the accuracy metrics. Gathering these results, a definitive conclusion about how this specific difference in preprocessing affects the model's performance cannot be established. Instead, the impact appears to vary depending on dataset size and the degree of variability between samples. However, it can be concluded that the scaling before or after padding did not affect the results remarkably in any of the performed tests.

There is potential for improvement in the preprocessing pipeline. The final pipeline encodes textual features using OrdinalEncoder, which does not truly capture the relationships between certain components. This is illustrated in Table 2.1 and Table 2.2. Using a different data structure, the implementation of TfidfVectorizer could be feasible, which will most likely be a suitable encoding method for this problem. Furthermore, some categorical features are scaled as numerical features, even though the numerical values do not represent a magnitude, and some numerical features are encoded as textual features. This can happen if a column with numerical values in the CSV file contains an error, as "VALUE", resulting in the whole column being preprocessed as textual. To solve this issue, one can manually decide which columns should be scaled and which ones should be encoded. This was considered inadequate, since all future columns must be named equally to the ones in the training data, and the goal was to prevent hard-coding the preprocessing function to the fullest extent.

Another preprocessing adjustment that is believed to potentially improve performance is sorting the components based on position instead of weight. The original weight of the components may vary between the car configurations, which implies that the sequence of components in a function group will also differ between the car configurations. This may be less intuitive to the model than if components in a function group were consistently sorted by position. Additionally, an error in the input weight will affect the component order negatively.

6.4 General limitations of the results

The datasets are high-dimensional, which affects the training time and model alternatives. Consider, for example, the medium-sized dataset,

which comprises just under 80,000 values in one sample divided into 150 columns. In contrast to several other large-dimensional datasets for machine learning training, the medium-sized dataset is characterised by a high degree of diversity, meaning there is an absence of repeated instances within the dataset. For example, one version of the medium dataset applied to the ensemble models contained 358 samples, all of them being unique. In addition, the majority of the features exhibit exclusively unique values in the datasets. It is highly recommended in machine learning to have many similar samples in the dataset to improve generalisability and to better capture the relationships in the data. This was not possible to achieve in this project, due to a lack of data and the array format of the samples. For example, an array with the values ["Screw", "Steering Wheel", "Motor"] is not considered the same value as ["Screw", "Motor", "Steering Wheel"]. This, too, can be a reason for the suboptimal results.

When initially reviewing literature on multi-output regression, neural networks and ensemble models appeared as promising candidates. However, the large dimensionality of the target data in this project might impact the performance of models. The size of the CAD and PLM output varies depending on the dataset size and the preprocessing approach. In some cases, both output lengths are of size 1500. This means that the model shall be able to predict 1500 weights per sample, which is a complex task and may be one of the contributing factors to the unsatisfactory results.

6.5 Future outlook of the model development

The plan was to implement the best-performing model as a part of a pipeline for performing data harmonisation to enable axial weight dis-

tribution calculations and centre of gravity prognosis. Even the model that achieved the best test accuracy was considered inadequate, and consequently, no model was implemented in the pipeline. As explained in Section 4.5, a pipeline was constructed, nonetheless, where a machine learning model can be integrated if an optimal one is found in further studies. All relevant content developed in the course of this project, including experiments with machine learning models, the preprocessing pipeline and the final pipeline for data harmonisation, will be stored in Databricks. Suggestions regarding future steps to explore are presented in the following subsection.

6.5.1 Future work

Early in the process, it was decided that the most suitable way to structure the data was in an array format, where all components that share the same function group are arranged into an array per column. The motivation behind this is the possible discrepancy in the number of CAD and PLM components in a function group. More information about this is found in Section 3.1 about the data harmonisation process. The initial thought was to use a combined model for the output of CAD and PLM weights, but to separate the two outputs turned out to be a more advantageous approach. To structure the data in an array format, however, might not be the most optimal one for the machine learning model in retrospect. A possibility that could be explored in future work is to keep the components one by one, as separate samples, and separating the CAD and the PLM components in two models with one training dataset each. Challenges will arise with this; however, if a solution to this issue is encountered, it would be of great value. With another data structure, other feature engineering techniques, methods for feature selection and machine learning models

might be possible.

7

Conclusion

One of the main research questions this thesis sought to answer was whether it was possible to construct an automated pipeline in Python to perform data harmonisation, axial weight distribution, and centre of gravity prognosis. A substantial amount of time and effort during the thesis was devoted to the data harmonisation part of this question, and more specifically to answer the research question of whether a machine learning model can predict the correct weights with sufficient test accuracy and thereby facilitate the manual data harmonisation process.

Answering the latter question involved a thorough investigation of various machine learning models suitable for multi-output regression, focusing on two main categories: neural networks and ensemble methods. Prior to this and concurrently with model development, preprocessing approaches were designed and iteratively refined. Although improvements could be observed from the various actions, none of the explored models achieved an acceptable function group accuracy above 70% for testing.

In parallel with model development, the automated pipeline was constructed, including preprocessing, weight prediction, axial weight dis-

tribution and centre of gravity prognosis. However, when it was established that no model with acceptable performance had been found, it was decided not to take the development and implementation of the automated pipeline further. This decision was taken as the automated pipeline does not hold value in itself without the model that can perform automated data harmonisation. Nevertheless, if a machine learning model with desired performance is developed, it can easily be integrated within the automated pipeline. It is worth mentioning, though, that since the development of the pipeline was interrupted, the research question of how outputs from the pipeline can be visualised in an accessible way, with the possibility of making manual adjustments, was not fully examined. Thus, there are still potential improvements that can be made to the automated pipeline.

Although none of the models explored reached the expected level of performance during the testing, several conclusions can still be drawn. First, the ensemble models consistently outperformed the neural networks for the various datasets used in this thesis. Among the tested ensemble models (the AdaBoost Regressor, the Extra Trees Regressor, the Gradient Boosting Regressor, the Random Forest Regressor and the XGBoost Regressor), the XGBoost Regressor demonstrated the best validation performance. The second-best validation performance was achieved by the Extra Trees Regressor. Meanwhile, the AdaBoost Regressor, Gradient Boosting Regressor, and Random Forest Regressor were excluded following the first experiment due to their lower validation performance.

Even though some of the ensemble models showed promising validation results, their test performance fell short of expectations and requirements. It became evident that the models could not correctly predict

component weights with sufficient accuracy for vehicle configurations that they had not been trained on, despite the believed similarities between car configurations. Initially, it was believed that this was a result of overfitted models; nevertheless, after various attempts to reduce overfitting, it could be concluded that this was not the only issue. Following this, it was considered appropriate to use a larger training set to determine whether the poor performance was due to insufficient training data. Since the results remained unsatisfactory, it could be concluded that the issue was not due to a lack of training samples but rather the significant variation within the samples.

The primary suggestion for future work lies in the domain of preprocessing, and more specifically, to consider an alternative approach to represent the data. The current array structure, which is a consequence of grouping the components based on function group, might not have been the most optimal choice. Alternatively, representing the components individually may facilitate better interpretability by machine learning models and potentially lead to reduced sample variability.

The final aim of this thesis was to implement a machine learning model as a tool for data harmonisation to enable necessary computations. Although this was not achieved with desired success, this study can be considered a guideline for further attempts to create such a tool. There are other machine learning models to investigate, but the primary focus in further work should preferably be on ensemble models, since they demonstrated the ability to fit the training and validation data successfully. There remain numerous hyperparameters to be optimised and multiple ways of restructuring the data to enhance model performance. Hence, the results do not indicate that using ensemble models

for predicting vehicle component weights is unsuitable, but rather that the optimal way of formulating this problem was not found.

Bibliography

[1] V. M. Nesro, T. Fekete, and H. Wicaksono, "Leveraging Causal Machine Learning for Sustainable Automotive Industry: Analyzing Factors Influencing CO2 Emissions," in *Procedia CIRP*, 2024, vol. 130, pp. 161–166, doi: 10.1016/j.procir.2024.10.071.

[2] M. K. Msakni, A. Risan, and P. Schütz, "Using machine learning prediction models for quality control: a case study from the automotive industry," *Comput. Manag. Sci.*, vol. 20, no. 14, Mar. 2023, doi: 10.1007/s10287-023-00448-0.

[3] C. Wu et al., "Sustainable AI: Environmental Implications, Challenges and Opportunities," 2022, arXiv:2111.00364.

[4] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge, U.K.: Cambridge Univ. Press, 2022. [Online]. Available: https://smlbook.org/.

[5] S. Suthaharan, *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*, 1st ed. New York, NY, U.S.: Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-1-4899-7641-3.

[6] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, “A survey and taxonomy of loss functions in machine learning”, 2024, arXiv:2301.05579v2.

[7] X. Ying, “An Overview of Overfitting and its Solutions,” in *J. Phys.: Conf. Ser.*, 2019, vol. 1168, no. 2, doi: 10.1088/1742-6596/1168/2/022022.

[8] Rainio, O., Teuho, J. and Klén, R. "Evaluation metrics and statistical tests for machine learning," *Sci. Rep.*, vol 14, Mar. 2024, doi: 10.1038/s41598-024-56706-x.

[9] Scikit-Learn. "scikit-learn: Machine Learning in Python". Scikit-Learn. Accessed: 04/04/2025. [Online]. Available: https://scikit-learn.org/stable/.

[10] P. Bahad and P. Saxena, “Study of AdaBoost and Gradient Boosting Algorithms for Predictive Analytics,” in *Proc. Int. Conf. Intelligent Computing Smart Commun.*, 2019, pp. 221–230, doi: 10.1007/978-981-15-0633-8_22.

[11] Scikit-Learn. "1.10. Decision Trees". Scikit-Learn. Accessed: 20/03/2025. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree.

[12] Scikit-Learn. "DecisionTreeRegressor". Scikit-Learn. Accessed: 19/03/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

[13] Quantstart. "Bootstrap Aggregation, Random Forests and Boosted Trees". Quantstart. Accessed: 20/03/2025. [Online]. Available: https://www.quantstart.com/articles/bootstrap-

aggregation-random-forests-and-boosted-trees/.

[14] Scikit-Learn. "RandomForestRegressor". Scikit-Learn. Accessed: 19/03/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor

[15] S. Baladram. "Extra Trees, explained: A visual guide with code examples". Medium. Accessed: 04/04/2025. [Online]. Available: https://medium.com/@samybaladram/extra-trees-explained-a-visual-guide-with-code-examples-4c2967cedc75.

[16] Scikit-Learn. "ExtraTreesRegressor". Scikit-Learn. Accessed: 04/04/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.htm

[17] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Front. Neurorobotics*, vol. 7, Dec. 2013, doi: 10.3389/fnbot.2013.00021.

[18] Scikit-Learn. "GradientBoostingRegressor". Scikit-Learn. [Online]. Accessed: 19/03/2025. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegres

[19] B. Schölkopf, Z. Luo, and V. Vovk, *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, Berlin, Germany: Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-41136-6.

[20] XGBoost. "XGBoost Documentation". XGBoost. Accessed: 04/04/2025. [Online]. Available: https://xgboost.readthedocs.io/en/release_3.0.0/#.

[21] MLflow. "MLflow: A Tool for Managing the Machine Learning Lifecycle". Accessed: 02/05/2025. Available: https://mlflow.org/docs/latest/index.html.

[22] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artif. Intell. Rev.*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021, doi: 10.1007/s10462-020-09896-5.

[23] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD '16)*, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[24] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *WIREs Data Mining Knowl. Discov.*, vol. 5, no. 5, pp. 389–400, Jul. 2015, doi: 10.1002/widm.1157.

[25] Scikit-Learn. "MultiOutputRegressor". Scikit-Learn. Accessed: 19/03/2025. [Online]. Available: https://Scikit-Learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegress

[26] J. Li et al., "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–45, Dec. 2017, doi: 10.1145/3136625.

[27] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*, Cambridge, U.K.: Cambridge Univ. Press, 2021. [Online]. Available: https://doi.org/10.1017/9781108860604.

[28] A. Shainky and A. Ambhaikar, "A study of machine learning dy-

namics: algorithms, applications, and fundamental frameworks," in *Proc. of Int. Conf. on Communication and Computational Technologies (ICCCT 2024)*, 2025, vol. 1122, pp. 25–35, doi: 10.1007/978-981-97-7426-5_4.

[29] B. Ben Elallid, N. Benamar, A. S. Hafid, T. Rachidi, and N. Mrani, "A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 9, pp. 7366–7390, Oct. 2022, doi: 10.1016/j.jksuci.2022.03.013.

[30] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning: An MIT Press book". Deep Learning. Accessed: 28/03/2025. [Online]. Available:]https://www.deeplearningbook.org/.

[31] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Int. J. Eng. Appl. Sci. Technol.*, vol. 4, no. 12, pp. 310–316, Apr. 2020. Accessed: 04/04/2025. [Online]. Available: http://www.ijeast.com.

[32] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 4, Art. no. 15, pp. 1–21, Dec. 2012, doi: 10.1145/2382577.2382579.

[33] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., Sebastopol, CA, U.S.: O'Reilly Media, 2019. [Online]. Available: https://shashwatwork.github.io/assets/files/ml_ebook.pdf.

[34] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," 2012, arXiv:1206.2944.

[35] S. Krishnan. "How do determine the number of layers and neurons in the hidden layer?". Medium. Accessed: 10/04/2025. [Online]. Available: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3.

[36] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, Cambridge, MA, U.S.: MIT Press, 2022. [Online]. Available: http://probml.github.io/book1.

[37] Y. Shin, "Effects of Depth, Width, and Initialization: A Convergence Analysis of Layer-wise Training for Deep Linear Neural Networks," *Anal. and Appl.*, vol. 20, no. 01, pp. 73-119, Jan. 2022, doi: 10.1142/S0219530521500263.

[38] S. Prabhakaran. "Introduction to Feature Selection Methods with an Example". Analytics Vidhya. Accessed: 10/04/2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/-.

[39] Scikit-learn. "MinMaxScaler". Scikit-learn. Accessed: 03/04/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[40] Scikit-learn. "StandardScaler". Scikit-learn Documentation. Accessed: 03/04/2025. [Online]. Available: https://scikit-

learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.htm

[41] Scikit-learn. "RobustScaler". Scikit-learn. Accessed: 03/04/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html.

[42] Scikit-learn. "OrdinalEncoder". Scikit-learn. Accessed: 03/04/2025. [Online]. \endgrouphttps://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.htm

[43] Scikit-learn. "LabelEncoder". Accessed: 11/04/2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[44] T. D. Gillespie, *Fundamentals of Vehicle Dynamics*, Warrendale, PA, U.S.: SAE Int, 1992. [Online]. Available: https://www.sae.org/publications/books/content/r-506/.

[45] S. Taylor. "Model-agnostic feature importance through ablation". samueltaylor. Accessed: 11/04/3025. [Online]. Available: https://www.samueltaylor.org/articles/feature-importance-for-any-model.html.

[46] A. Paszke, S. Gross, S. Chintala, and G. Chanan. "PyTorch". Wikipedia The Free Encyclopedia. Accessed: 04/04/2025. [Online]. Available: https://en.wikipedia.org/wiki/PyTorch.

[47] L. Rokach, "Decision forest: Twenty years of research," *Inf. Fusion*, vol. 27, pp. 111-125, Jan. 2016, doi: 10.1016/j.inffus.2015.06.005.

[48] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative

analysis of gradient boosting algorithms," *Artif. Intell. Rev.*, vol. 54, pp. 1937-1967, Aug. 2021, doi: 10.1007/s10462-020-09896-5.

[49] C. González, J. M. Mira, and J. A. Ojeda, "Applying multi-output random forest models to electricity price forecast," Preprint at Preprints.org, Sep. 2016, doi: 10.20944/preprints201609.0053.v1.

[50] K. N. Das, J. C. Bansal, K. Deep, A. K. Nagar, P. Pathipooranam, and R. C. Naidu, Eds., *Soft Computing for Problem Solving*,

[51] K. Hoffman, J. Y. Sung and A. Zazzera, "Multi-Output Random Forest Regression to Emulate the Earliest Stages of Planet Formation," *Proc. 2021 Syst. Inf. Eng. Des. Symp. (SIEDS)*, 2021, pp. 1-6, doi: 10.1109/SIEDS52267.2021.9483749.

[52] L. Schmid, A. Gerharz, A. Groll, and M. Pauly, "Tree-based ensembles for multi-output regression: Comparing multivariate approaches with separate univariate ones," *Comput. Statist. & Data Analysis*, vol. 179, p. 107628, Mar. 2023. doi: 10.1016/j.csda.2022.107628.

[53] Z. Zhang and C. Jung, "GBDT-MO: Gradient-Boosted Decision Trees for Multiple Outputs," *Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 3156-3167, July 2021, doi: 10.1109/TNNLS.2020.3009776.

[54] X. Zhan, S. Zhang, W. Y. Szeto, and X. M. Chen, "Multi-step-ahead traffic speed forecasting using multi-output gradient boosting regression tree," *Int. J. Transp. Sci. Technol.*, vol. 8, no. 2, pp. 125-141, Mar. 2019, doi: 10.1080/15472450.2019.1582950.

[55] A. Najmoddin, H. Etemadfard, A. Hosseini.S and M. Ghalehnovi,

"Multi-output machine learning for predicting the mechanical properties of BFRC," *Case Studies in Construction Materials*, vol. 20, Jan. 2024, doi: 10.1016/j.cscm.2023.e02818.

[56] A. Dogan, D. Birant and A. Kut, "Multi-Target Regression for Quality Prediction in a Mining Process," *2019 4th International Conference on Computer Science and Engineering (UBMK)*, Samsun, Turkey, 2019, pp. 639-644, doi: 10.1109/UBMK.2019.8907120.

[57] C. Kucuk, D. Birant, and P. Y. Taser, “An intelligent multi-output regression model for soil moisture prediction,” in *Proc. Int. Conf. Intelligent and Fuzzy Techniques for Emerging Conditions and Digital Transformation (INFUS)*, Istanbul, Turkey, Aug. 2021, vol. 1552, pp. 475–481. [Online]. Available: https://doi.org/10.1007/978-3-030-85577-2_57.

[58] Streamlit. "Streamlit Documentation". Streamlit. Accessed: 02/05/2025. Available: https://docs.streamlit.io/.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY

CHALMERS UNIVERSITY OF TECHNOLOGY

CHALMERS
UNIVERSITY OF TECHNOLOGY