xAILAB
EXPLAINABLE MACHINE LEARNING

Component Identification for Geometric Measurements in the Vehicle Development Process Using Machine Learning

Master Thesis

Master of Science in Information Systems

Tobias Koch

December 15, 2023

Supervisor:

1st: Prof. Dr. Christian Ledig

[redacted]

Chair of Explainable Machine Learning
Faculty of Information Systems and Applied Computer Sciences
Otto-Friedrich-University Bamberg

Abstract

Geometric measurements are frequently performed along the virtual vehicle development chain to monitor and confirm the fulfillment of dimensional requirements for purposes like safety and comfort. The current manual measuring process lacks in comparability and quality aspects and involves high time and cost expenditure due to the repetition across different departments, engineers, and vehicle projects.

Thereby motivated, this thesis develops and implements automated solutions for the component identification within the geometric measurement process. The first goal is to classify the components of a vehicle as relevant and not relevant for the geometric measurements (binary classification task), and the second goal is to generate uniformly coded designations for the relevant car components (multi-class classification task). Artificial Intelligence (AI) is used in combination with rule-based filters to automate the component selection. Light Gradient-Boosting Machines (LightGBMs), eXtreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), and Feedforward Neural Networks (FNNs) are investigated for these tasks, which are compared regarding performance and training complexity. For the binary classification task, the highest average F_2-score of 95.465% using k-fold cross-validation is achieved by a LightGBM model. On the test set, this model reaches an F_2-score of 97.142%. For the multi-class classification task, the highest average F_2-score of 98.951% using k-fold cross-validation is achieved by a CatBoost model. On the test set, this model reaches an F_2-score of 100%.

As shown in this thesis, gradient boosting models can effectively be integrated into the geometric measurement process to increase comparability and reduce time by identifying relevant components and assigning uniformly coded designations. The code implemented for this thesis is open source and available in a Git-repository for further examination and utilization.[1]

[1] `https://github.com/koch-tobias/master-thesis`

Contents

List of Figures

List of Tables

List of Acronyms

Adam	Adaptive Moment Estimation
AdaGrad	Adaptive Gradient
AI	Artificial Intelligence
API	application programming interface
AdaBoost	Adaptive Boosting
AWS	Amazon Web Services
Bagging	Bootstrap aggregating
CatBoost	Categorical Boosting
CATIA	Computer Aided Three-Dimensional Interactive Application
EDA	exploratory data analysis
EFB	Exclusive Feature Bundling
FNN	Feedforward Neural Network
GOSS	Gradient-based One-Side Sampling
GPT	Generative Pre-trained Transformer
LeakyReLU	Leaky Rectified Linear Unit
LightGBM	Light Gradient-Boosting Machine
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RMSProp	Root Mean Square Propagation
SGD	stochastic gradient descent
SHAP	SHapley Additive exPlanations
SAE	Society of Automotive Engineers
XGBoost	eXtreme Gradient Boosting

1 Introduction

The automotive industry is the most important industrial sector in Germany and substantially outperforms other industries in terms of sales, investment, exports, as well as research expenditure (Müller, 2022). The steady striving to enhance safety, efficiency, and comfort in this area requires automobile manufacturers to develop vehicles that meet ever stricter specifications and quality standards.

In this context, the geometric measurement of functional dimensions is an essential step in the virtual vehicle development process to monitor and confirm the fulfillment of geometric targets. Examples of geometric targets are the loading sill height for the trunk or the viewing angle out of the vehicle. By repeating the measurements frequently along the whole development chain, potential deviations are identified at an early stage, and appropriate measures can be initiated directly to ensure that the vehicle meets the requirements. To make these repeated geometric measurements traceable, reproducible, and comparable along the development chain, identical procedural methods and dimensional definitions are crucial, especially as this task is performed across different departments, employees, and vehicle projects. However, despite these precise specifications, the manual geometric measuring process is subject to several limitations and is time consuming, which is why automotive manufacturers are searching for new, increased automated methods. These are expected to improve the quality of the performed measurements and increase the efficiency along the virtual development chain.

Thereby motivated, this thesis develops and implements an automated solution for the component identification within the geometric measurement process for functional dimensions. Artificial Intelligence (AI) is used in combination with rule-based filters to automate the component selection. Traditional machine learning methods (Light Gradient-Boosting Machine (LightGBM), eXtreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost)) and neural networks (Feedforward Neural Network (FNN)) are investigated for this task, which are compared regarding performance and training complexity. Additionally, an AI model is developed to generate uniformly coded designations for relevant car components. To make the results of the thesis directly and easily usable, the software solution is integrated into a Computer Aided Three-Dimensional Interactive Application (CATIA) pipeline via an application programming interface (API).

Chapter 1.1 first describes the process and method of the manual geometric measurements of functional dimensions and the related challenges and limitations. Subsequently, Chapter 1.2 presents the goal of the complete measurement tool, which is currently under development, and Chapter 1.3 concludes the introduction by detailing the structure of the thesis.

1.1 Current Situation

This chapter describes the current process and method for the manual geometric measurement of functional dimensions in the virtual vehicle development process. It also explains the challenges and limitations of this process, which motivate the need for more automated solutions.

1.1.1 Process and Method

The manual geometric measurement process consists of four consecutive steps, namely the identification of relevant car components, the upload of the relevant components into CATIA, the measurement execution, and the documentation and comparison of the results. These steps are described in more detail in the following. Since different lists and tools are necessary for these steps, Figure 1 gives a conceptual overview.

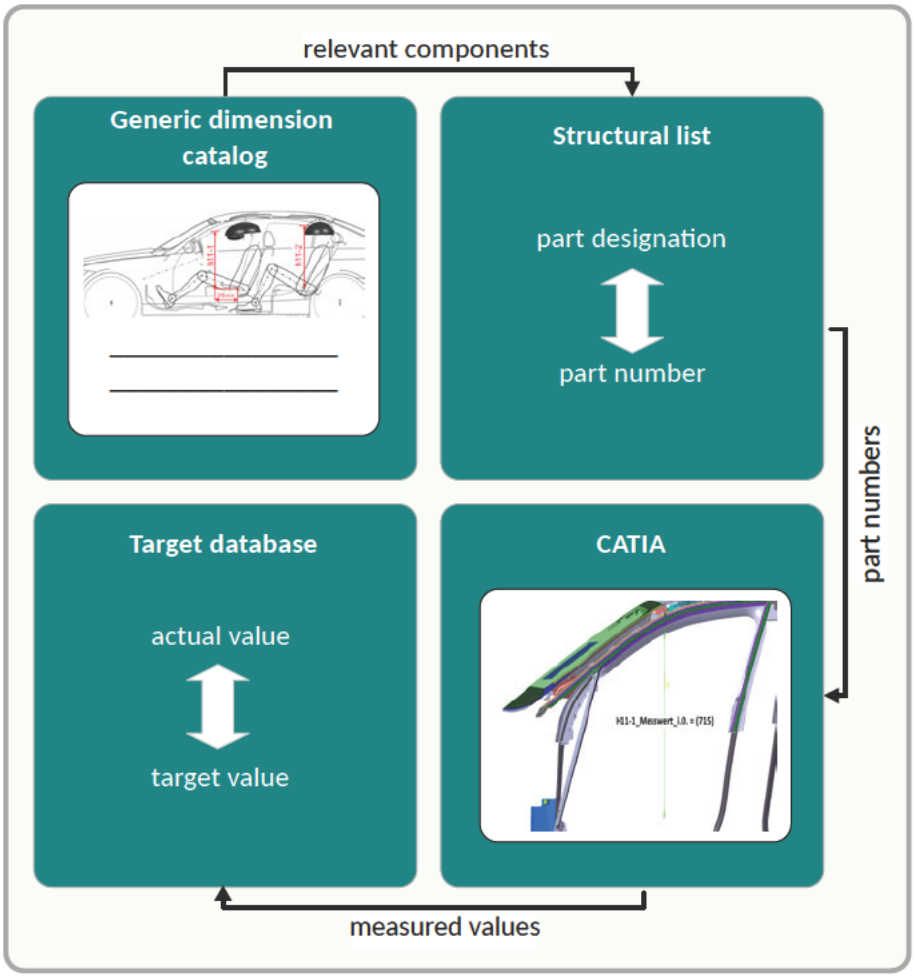


Figure 1: Overview of the lists and tools used in the manual measurement process.

Identification of relevant car components

The first step for determining the required dimensions of the vehicle is to identify the relevant car components for the measurements. These are not directly available but must be derived from a **generic dimension catalog** (see Figure 1) accessible on a Confluence page. The generic dimension catalog includes a subpage with a sketch and a textual description for each functional dimension that needs to be measured. From this information, the engineer needs to infer the necessary car components for the distance determination.

Figure 2 shows the sketch from the subpage for "headroom exit" of the generic dimension catalog, which is intended to graphically illustrate the structure of the dimensions "headroom exit front `h11-1`" and "headroom exit rear `h11-2`". The textual descriptions for these measures contain a definition of the dimension as well as further information about how components have to be cut in CATIA and which predefined planes are necessary for the construction in CATIA.

Using the example dimension "headroom exit front `h11-1`", the engineer should identify the components "front door edge protection" and "headliner" from the given information on the Confluence subpage.

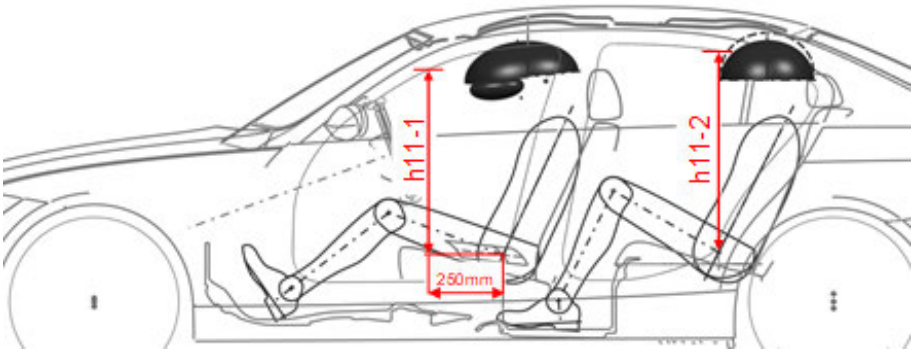


Figure 2: Visualization of the dimension "headroom exit front `h11-1`" and "headroom exit rear `h11-2`".

Loading of components into CATIA

After identifying the relevant components, they need to be loaded into CATIA. For this step, the **structural list** (see Figure 1) of the analyzed vehicle model is required, which contains all file paths and components. The common way is to navigate through the hierarchical structure of the parts until finding the designations matching to the required components identified in the previous step. From the structural list, the part numbers corresponding to the component designations can be extracted, and with them, the components can be loaded into CATIA. Loading the entire vehicle into CATIA is not recommended due to the required time and memory of the upload.

In the example of the dimension "headroom exit front `h11-1`", the engineer searches the required component designations for "front door edge protection" and "headliner" in the structural list, extracts the part numbers, and loads the components into CATIA using the part numbers.

Measurement execution

With the required components loaded into **CATIA** (see Figure 1), the measurements can be set up and performed. Therefore, the engineer sets the measurement points at the components in CATIA based on the textual definition given in the generic dimension catalog, whereby the geometrical distance between them is calculated in the tool.

Based on the textual definition of the dimension "headroom exit front `h11-1`", the engineer needs first to analyze whether the "headliner" or the "front door edge protection" has the lowest point and then needs to select the point. This can be realized in CATIA by generating cuts of the components. The second, bottom point for the measurement needs to be set on a predefined plane of the vehicle in CATIA. After selecting these two points, the geometrical distance, as shown in Figure 3, is calculated.

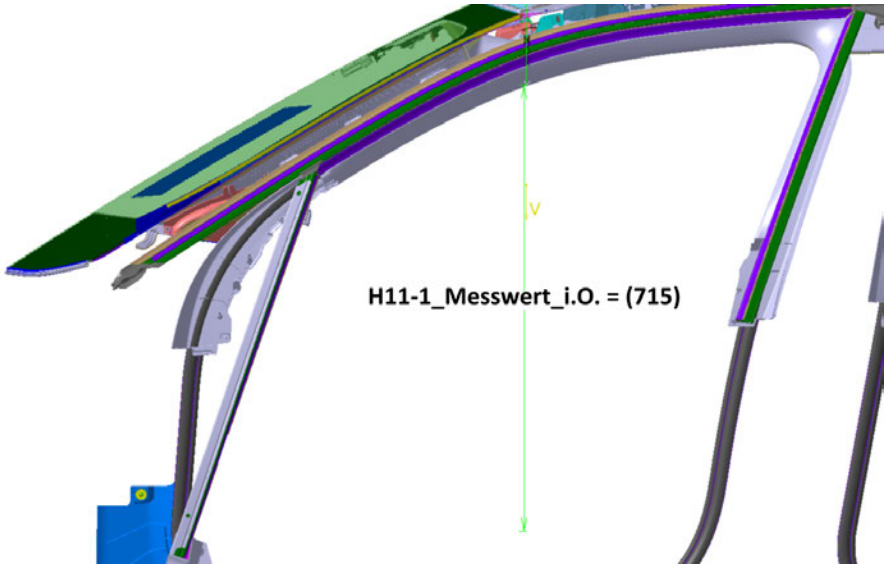


Figure 3: Measurement "headroom exit front `h11-1`" in CATIA.

Documentation and comparison of the results

In the last step, the measured values need to be transferred to a **target database** (see Figure 1) for the comparison with the target values. If all measured values of a vehicle are within the specified tolerances, the project phase can be released, and the current development status can be frozen. This means that no further adjustments

are made in the current step of the development chain to ensure that the achieved geometric targets are preserved. If the measured values deviate from the target values, adequate actions must be initiated to correct the discrepancies. This may involve revising certain car components, like adjusting the design.

1.1.2 Challenges and Limitations

This chapter describes the challenges and limitations of the manual process by focusing on the time and cost expenditure as well as the comparability and quality aspects.

Time and cost expenditure

The manual geometric measuring of functional dimensions in the virtual vehicle development process is time-consuming and costly. This is particularly generated by the frequent repetition of the measurements along the virtual vehicle development process. In addition, other factors increase the required time, for example, if the dimensions are complex or if engineers do not have a routine in measuring the vehicle dimensions. In detail, a thorough analysis of the descriptions and in-depth knowledge of the vehicle structure is necessary, especially to identify the relevant car components from the generic dimension catalog and to navigate to the component designation in the structural list. However, since different employees perform the measurements along the development chain, experience is often lacking, and efficiency is suffering. According to surveys with internal engineers and external service providers, the measurement of a complete vehicle requires about 40 working hours from inexperienced engineers and about 16 working hours from experienced engineers. The high time expenditure is directly linked to high costs. Either personnel costs are incurred if internal developers take over the task or costs for contracting a service provider if the task is outsourced.

Comparability and quality

Another limitation is the dependence of the measurement process on the generic dimension catalogs and their interpretation by the engineers because it can reduce the comparability and the quality of the measurements. If the dimensions in the generic dimension catalogs are too complexly defined, it can be challenging to identify the relevant car components and to accurately measure the required dimension. Since the measurements are conducted by different people along the development chain, there may also be variations in the interpretation of the required car components or the dimensional structure. As a result, the measured values are not exactly comparable, and the quality of the measurement process is limited. Overall, as with almost any manual task, it can be concluded that manual measurements are prone to errors and inconsistencies.

1.2 Goal Setting and Modules of the Measurement Tool

This chapter briefly describes the goal and structure of the overall measurement tool.

1.2.1 Goal of the Measurement Tool

Motivated by the described challenges and limitations of the current manual process for the geometric measurements in the virtual vehicle development, a new tool should be developed that automates the process to a large extent. By automating the process, the objective is to optimize and improve the virtual development process to reduce time and costs as well as to maximize the comparability and quality of the measurement results.

1.2.2 Modules of the Measurement Tool

Figure 4 describes the structure of the measurement tool, which is set up to perform the measurements largely automatically. The measurement tool consists of three main building blocks, specifically an AI system, a CATIA parametric model, and a CATIA macro between the AI system and the CATIA parametric model.

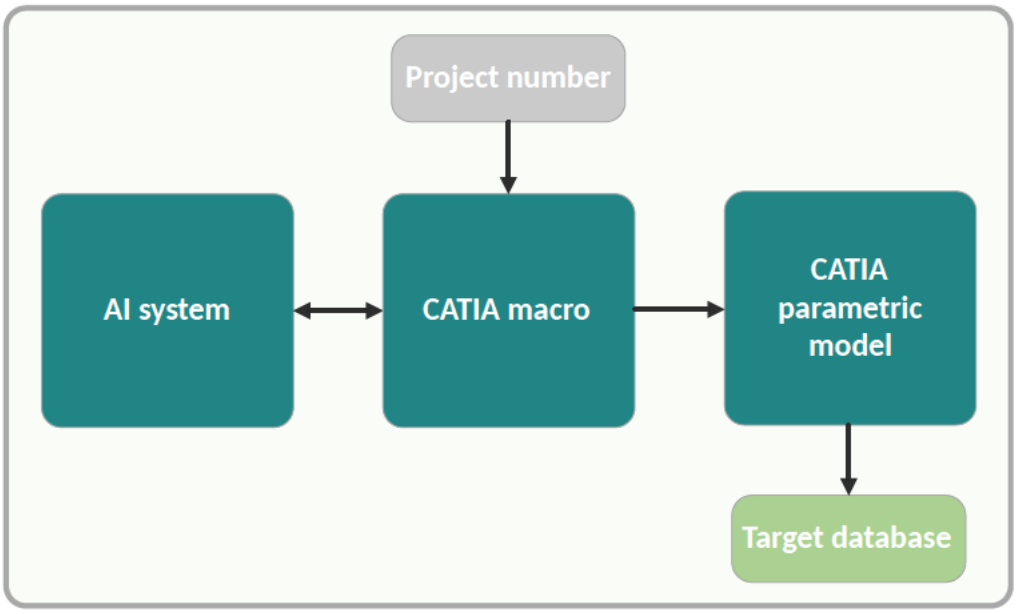


Figure 4: High-level architecture of the measurement tool.

CATIA macro

The CATIA macro enables the communication between the AI system and the CATIA parametric model. Here, the engineer can insert the project number of

the desired vehicle model, which the macro uses to get the structural list from the database. After downloading the structural list, it is transmitted to the AI system, which analyzes the data and returns a list of all relevant components for the measurements, including uniformly coded designations. Then, the CATIA macro loads the components in CATIA and creates planes and cuts that represent the components as required for precise geometric measurements.

AI system for the component identification

As apparent from the description of the manual process, the component identification is a crucial step for the automation of the measurement process and the core of this thesis. All measurement-relevant components are identified with the developed AI system by analyzing the metadata of the components in the vehicle's structural list. In addition, the AI system generates uniformly coded designations for the relevant components across all vehicle models so that they can be assigned later to the corresponding dimensions in the parametric model. Specifically, this is implemented with rule-based filters and machine learning methods.

CATIA parametric model

The CATIA parametric model performs the measurements on the selected components and compares them with the target values. It enables automated execution of the measurements and ensures reproducibility as well as high accuracy. The results are compared directly in the model to the target specifications from the general vehicle plan, which are displayed in color analogous to the traffic light logic. Afterward, the measured values are automatically transferred to another system, which is used for documentation purposes and enables further evaluations and analyses.

1.3 Structure of the Thesis

Chapter 2 explains the theoretical background of this thesis with a focus on the implemented machine learning methods. Chapter 3 introduces the data set by describing the raw data, the data labeling, the data preparation, and the data split. Chapter 4 deals with the method and includes the presentation of the development environment and general concepts, such as the metrics and the hyperparameter optimization techniques used for training the models. In addition, Chapter 4 gives an in-depth explanation of the implemented machine learning methods. The models from Chapter 4 are then evaluated based on the presented metrics in Chapter 5 and further interpreted and discussed in Chapter 6. After Chapter 7 shows the deployment architecture of the AI system, Chapter 8 addresses the limitations of this thesis and provides an outlook on possible future work. The thesis is concluded with Chapter 9, which gives a summary of the main findings.

2 Theoretical Background

This chapter provides the theoretical background knowledge that is important for understanding the AI methods implemented in this thesis. Figure 5 visualizes the AI subfields the used methods fall into, namely machine learning as well as deep learning. As a basis, Chapter 2.1 explains the fundamentals behind the cross-entropy loss and gradient descent optimization, before Chapter 2.2 describes gradient boosting, including three frameworks that can be classified as traditional machine learning techniques. The following Chapter 2.3.1 explains FNNs, which are considered as deep learning if they consist of multiple hidden layers, and a deep learning framework. Lastly, Chapter 2.3.2 briefly introduces Generative Pre-trained Transformer (GPT), which also belong to the subfield of deep learning as they involve neural networks with multiple hidden layers.

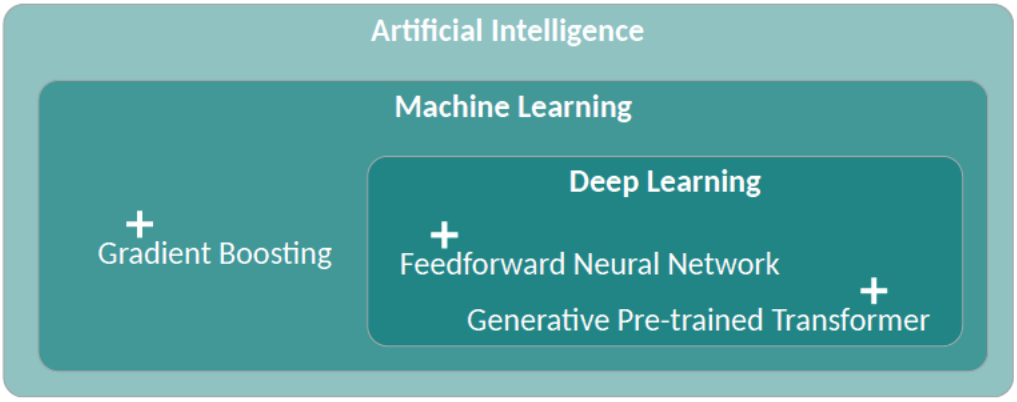


Figure 5: Categorization of the implemented methods in the common AI subfields.

2.1 Cross-Entropy Loss and Gradient Descent Optimization

The cross-entropy loss as a loss function and gradient descent as an optimization method are fundamental in the context of classification tasks in supervised learning and form the basis of many machine learning methods and, specifically, also of the methods investigated in this thesis. Hence, the following section explains both concepts, before the used machine learning methods are detailed.

For supervised learning tasks, optimization refers to the process of finding an optimal set of parameters $\boldsymbol{\theta}$ for a model that minimize a certain loss function $\mathcal{L}$. During the training process, the model learns to make predictions of the target output $(\hat{Y})$ based on the provided input features (X) by identifying patterns in the labeled training set, which consists of pairs of input features and corresponding target output values $(x_1, y_1), (x_1, y_2), \ldots, (x_N, y_N)$. For simplicity reasons, x_i and y_i are assumed as scalars in this chapter, but the formulas apply also for vectors.

With a loss function $\mathcal{L}$ the discrepancy between the predicted outputs $(\hat{Y})$ and the target outputs (Y) can be measured. The closer the loss function's value is to zero,

the better the model has learned to predict the output. Thus, minimizing the loss function $\mathcal{L}$ can improve the performance of the model.

Depending on the task of the model and the characteristics of the data, a loss function $\mathcal{L}$ has to be selected. For the classification tasks in this thesis, the cross-entropy loss, also called log-loss, is chosen. Entropy is rooted in information theory and goes back to Claude Shannon, where it is used to estimate the amount of uncertainty and information of a variable (Shannon, 1948). In the discrete case, the entropy H is calculated as follows, where I is the information in the variable X, and X takes the discrete values x_i with corresponding probabilities p_i (adapted from (Baddeley et al., 2000)):

$$H(X) = E[I(X)] = -\sum_i p_i \log(p_i) \tag{1}$$

The difference between two variables X_1 and X_2 can be assessed with the cross-entropy and is calculated in the discrete case according to the following equation, where the variables X_1 and X_2 take the values x_{1i} and x_{2i} with corresponding probabilities p_i and q_i (adapted from (Ramsundar and Zadeh, 2018)):

$$H(X_1, X_2) = -\sum_i p_i \log(q_i) \tag{2}$$

In the application for supervised learning, X_1 is the target variable Y and X_2 the prediction thereof $\hat{Y}$. In the binary case, p_i corresponds to the class label $y_i \in \{0, 1\}$ and q_i to the predicted probability either for the class with label zero $(1 - \hat{y}_i)$ or for the class with label one $(\hat{y}_i)$. The average cross-entropy loss over N predictions can then be simplified as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3}$$

Due to the logarithm in Equations 2 and 3, the cross-entropy loss penalizes especially those predicted outputs $\hat{y}_i$, which have high deviations from the target outputs y_i.

To minimize the value of the loss function $\mathcal{L}$, optimizers are used, which iteratively adjust the parameters $\boldsymbol{\theta}$ of a model. The most fundamental optimization algorithm used for machine learning tasks is gradient descent, which updates the parameters by moving them in the direction of the negative gradient of the loss function.

The gradient descent algorithm starts by initializing the model's parameters $\boldsymbol{\theta}$ with small random values. Then, in each iteration, it calculates the loss based on the predicted outputs $\hat{y}_i$, for example, according to Equation 3 in the binary classification task using the cross-entropy loss. The gradient of the loss is then calculated, which indicates the direction and magnitude of the steepest increase in the loss function. To minimize the loss, the parameters $\boldsymbol{\theta}$ are then updated according to the following equation by subtracting the gradient (steepest descent) multiplied by a learning rate (η), which controls the step size (adapted from (de Roos et al., 2021)):

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \nabla \mathcal{L}(\boldsymbol{\theta}_i) \tag{4}$$

The algorithm repeats the described steps for a predefined number of iterations or until convergence. To ensure convergence and stability in the optimization process, choosing an appropriate learning rate η is crucial. What is appropriate depends on various factors, for example the model complexity and the data set. A learning rate that is too small might result in a slow convergence of the model, while a learning rate that is too large might cause the model to overshoot the minimum.

Figure 6 illustrates the gradient descent algorithm for the two-dimensional function $f(x_1, x_2) = 2x_1^2 + 2x_2^2$, which is chosen as it is a simple, convex function and demonstrates how learning rates affect the direction changes for each iteration to find a minimum of a function. While the learning rate of 0.5 is too large to converge to the minimum due to high oscillations and overshooting, the learning rate of 0.1 is too small to converge to the minimum in the predefined number of iterations (4). The learning rate of 0.3 is appropriately selected and converges to the minimum.

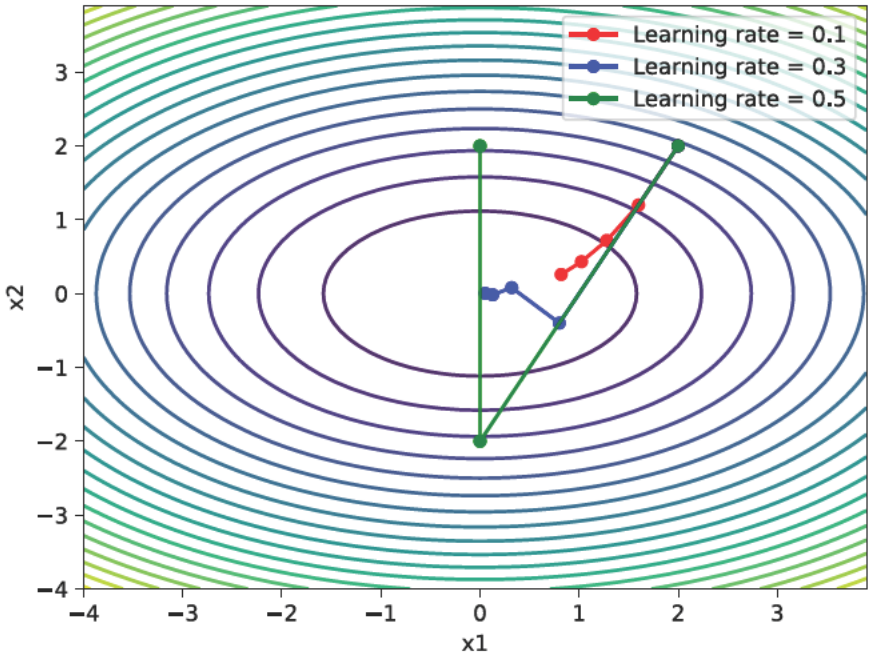


Figure 6: Gradient descent for $f(x_1, x_2) = 2x_1^2 + 2x_2^2$.

2.2 Ensemble Learning

With the fundamentals explained, this chapter focuses on the background of the implemented traditional machine learning techniques. Before it details gradient

boosting including three frameworks thereof, ensemble learning in general is briefly described.

In contrast to numerous other machine learning methods that generate predictions with a single model, ensemble learning focuses on increasing the prediction quality by combining predictions from multiple models. Two of the most widely used ensemble learning methods are Bootstrap aggregating (Bagging) (Breiman, 1996) and Boosting (Freund and Schapire, 1999). The key difference between both techniques is their workflow for training the individual models, which is visualized in Figure 7. Bagging algorithms train multiple models parallelly on randomly sampled data subsets and aggregate their predictions. The most prominent example that uses Bagging is a random forest (Breiman, 1996). Boosting algorithms train multiple models sequentially, such that the data set can be sampled based on the errors of the previous model and weight the individual predictions before aggregating them.

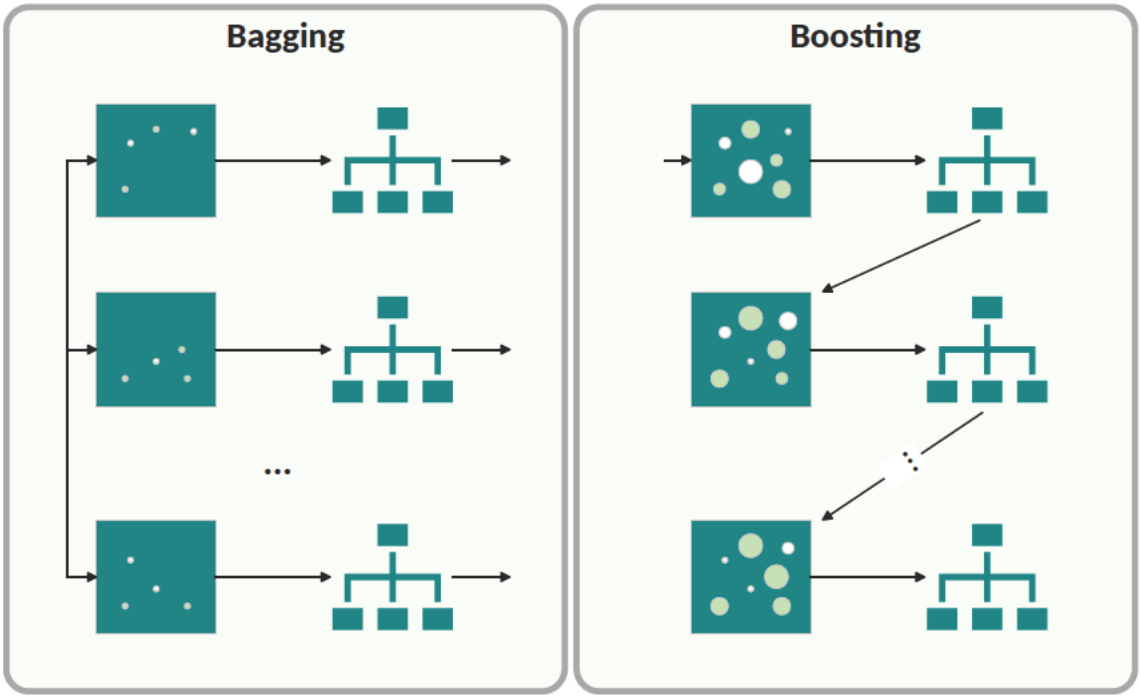


Figure 7: Workflow comparison between the ensemble learning techniques Bagging and Boosting (adapted from (Ismail et al., 2023)).

2.2.1 Boosting

The most widely used Boosting methods are Adaptive Boosting (AdaBoost) (Freund and Schapire, 1997) and different gradient boosting methods, which are a special form of boosting that leverage additional the principles of gradient descent in their learning process (Friedman, 2001).

Hence, before gradient boosting is described in the following chapter, this chapter focuses on the explanation and formulas of the more basic AdaBoost algorithm introduced in 1995 by Freund and Schapire (Freund and Schapire, 1997). Given the data set (x_1, y_1), (x_1, y_2), ..., (x_N, y_N), the goal is to find a performant predictive model $H(x)$ by combining a sequence of weak learners $h_1(x)$, $h_2(x)$, ..., $h_T(x)$ as follows, where α_t represents the weight assigned to each of the T weak learners (adapted from (Freund and Schapire, 1999)):

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{5}$$

A weak learner $h_t(x)$ is a learning algorithm with limited predictive power that might perform solely slightly better than random guessing (Freund and Schapire, 1999). For example, this can be a decision stump (one-level decision tree) or a linear classifier. The strength of the method lies in its ability to boost the performance of these weak learners by aggregating the predictions with weights α_t and adjusting data weights D_t as follows, where ϵ_t is the error of one weak learner (adapted from (Freund and Schapire, 1999)):

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t}) \tag{6}$$

$$D_{t+1}(i) = \frac{D_t(i) \exp -\alpha_t y_i h_t(x_i)}{Z_t} \tag{7}$$

The key idea here is to assign higher weights D_t to the misclassified examples and thus to increase their importance in the following iterations. Thereby, boosting algorithms effectively concentrate on the challenging examples to classify correctly. The final hypothesis produced by boosting is a weighted combination of these weak learners, where the weights α_t are determined based on their individual performance (see Equation 6).

2.2.2 Gradient Boosting

Gradient boosting was introduced in 1999 by Friedman and is a special form of boosting that leverages the principles of gradient descent in its learning process (Friedman, 2001). Again, the goal is to find a performant predictive model $H(x)$ by combining a sequence of weak learners $h_1(x)$, $h_2(x)$, ..., $h_T(x)$ given the data set (x_1, y_1), (x_1, y_2), ..., (x_N, y_N). In contrast to AdaBoost, where the weights of data points are adjusted before the following weak learner is trained, in gradient boosting, the following weak learner focuses on the difference between the prediction and the ground truth of the previous weak learner (called residuals or residual error) (Bentéjac et al., 2021). Hence, for gradient boosting, a loss function $\mathcal{L}$ needs to be selected, for example, the previously explained cross-entropy loss, which measures the discrepancy between the predicted values and the target values. A weak learner

$h_t(x)$ in gradient boosting is usually a slightly larger model than in AdaBoost and is often a decision tree (gradient boosting tree).

The gradient boosting algorithm starts by initializing the model $H(x)$ with a constant approximation, where α is set as a fixed value (Bentéjac et al., 2021):

$$H_0(x) = \underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(y_i, \alpha) \tag{8}$$

In each gradient boosting iteration t, pseudo-residuals, denoted as r_{it} for the data record i, are calculated as follows using the negative gradient of the loss function, where $H_{t-1}(x)$ represents the ensemble model of the iteration $t-1$ (Bentéjac et al., 2021):

$$r_{it} = -\left[\frac{\partial \mathcal{L}(y_i, H(x_i))}{\partial H(x_i)}\right]_{H(x)=H_{t-1}(x)} \tag{9}$$

The weak learner $h_t(x)$ is trained to minimize the pseudo-residuals r_{it} by fitting a decision tree to the training data. The resulting weak learner $h_t(x)$ is then added to the ensemble model of the iteration $t-1$ weighted by the learning rate (η) that controls the contribution of each weak learner. The updated model can be represented as (Bentéjac et al., 2021):

$$H_t(x) = H_{t-1}(x) + \eta * h_t(x) \tag{10}$$

This process is repeated until a predefined stopping criterion or iteration number T is reached, and results in the final model, which is the sum of all weak learners (Bentéjac et al., 2021):

$$H(x) = H_0(x) + \eta * h_1(x) + \eta * h_2(x) + ... + \eta * h_T(x) \tag{11}$$

By iteratively updating the model and training further weak learners, gradient boosting effectively improves the model's predictive performance.

After presenting the theoretical fundamentals of gradient boosting, three state-of-the-art gradient boosting frameworks (XGBoost, LightGBM, and CatBoost) are described in the following chapters. While they all share the principles of gradient boosting and use decision trees as the default choice for the weak learners, they differ in several additional aspects that are discussed in more detail. To allow a better comparison, each method is analyzed based on sampling characteristics, leaf growth, handling of sparse data, and handling of categorical features. Table 1 provides an overview of the comparison of these aspects in the three frameworks. The comparison shows that all three gradient boosting frameworks have different strengths due to their slightly different methods and algorithms. Which framework performs best depends heavily on the individual use case and cannot be generalized. A benchmarking for gradient boosting methods shows that all frameworks can be deployed very promisingly, with a slight tendency towards LightGBM as the best performing framework across different data sets (Florek and Zagdański, 2023).

Table 1: Comparison of selected aspects in the three gradient-boosting frameworks.

	XGBoost	**LightGBM**	**CatBoost**
sampling	no	yes	yes
leaf growth	level-wise	leaf-wise	symmetric
handling sparsity	yes	yes	yes
handling categorical features	no	yes	yes

2.2.2.1 eXtreme Gradient Boosting

XGBoost is an efficient, flexible, and scalable gradient boosting framework introduced in 2014 (Chen and Guestrin, 2016; Chen and He, 2023). In the Kaggle challenges published in 2015, XGBoost was already the most frequently used solution among the winners (Chen and Guestrin, 2016).

In contrast to the other two frameworks, in the default setting, XGBoost uses no **sampling** of the training data, which means that all data is used in one boosting iteration (Chen and Guestrin, 2022).

XGBoost uses a **level-wise** leaf growth strategy for its decision trees and generates them with a greedy algorithm for determining the splits (Alshari et al., 2021). A level-wise leaf growth means that the decision tree is expanded level by level, where a level refers to a specific tree depth, as shown in the center box of Figure 8. The greedy algorithm for splitting the leaves involves sorting the training instances based on the values of each feature, searching for potential split points, and calculating the information gain for each split point (Chen and Guestrin, 2016). Subsequently, XGBoost selects the split point that maximizes the gain for each feature, thereby determining the splitting feature for the corresponding tree node. The data is then divided into two subsets based on the selected feature and the split point, with the instances that fulfill the split condition being assigned to the left child node and the remaining instances to the right child node. This recursive splitting process continues level-wise until a stopping criterion is fulfilled, like reaching the maximum tree depth or a minimum number of instances in a leaf node. By iteratively training decision trees and optimizing splits based on the gain criterion, XGBoost gradually improves the model's predictive capabilities while minimizing the loss function. This process continues until a predefined number of trees is reached, or the model's performance converges.

Another feature of XGBoost is its ability to **handle sparsity patterns** in the input data without requiring manual preprocessing. This is achieved by assigning a default direction to each tree node so that even when the feature required to determine the split decision is missing, it can be classified (Chen and Guestrin, 2016).

XGBoost is built to work with numerical data and has no common option to **handle categorical features**, instead, the user should encode the features in a preprocessing step (Chen and Guestrin, 2022). An in-built option for handling categorical

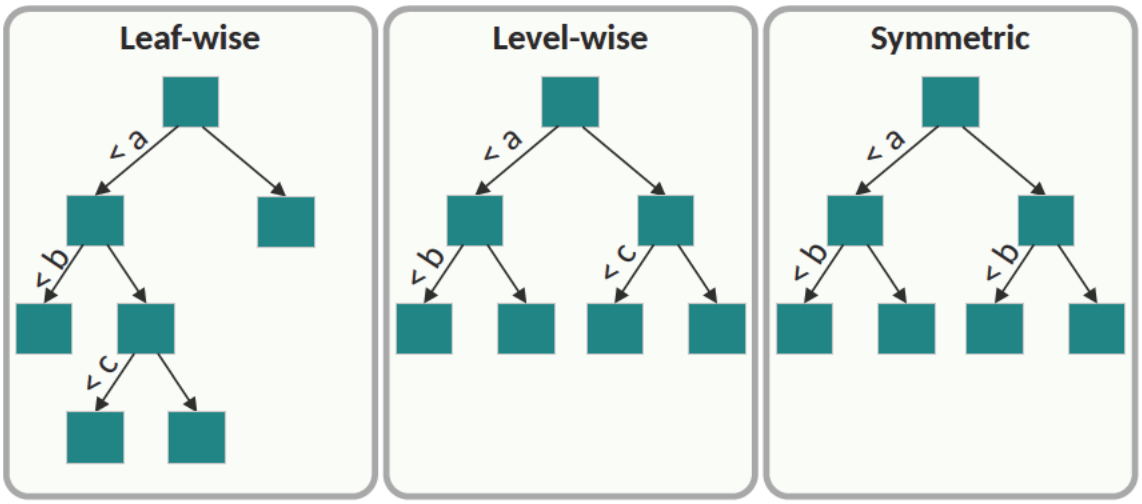


Figure 8: Example structures of decision trees, which are built using leaf-wise growth (left box), level-wise growth (center box), and symmetric growth (right box).

features is in an experimental stage and is planned to be advanced in the future (Chen and Guestrin, 2022).

2.2.2.2 Light Gradient-Boosting Machine

LightGBM is another gradient boosting framework introduced in 2016 that shares similarities with XGBoost, such as the way to **handle sparse data** (Ke et al., 2017). However, LightGBM differs from XGBoost mainly in the three aspects of sampling, leaf growth, and the handling of categorical features (Zhang and Gong, 2020).

LightGBM uses two unique **sampling** techniques to reduce the dimensionality, namely Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) (Ke et al., 2017). These have contributed to the popularity of LightGBM and have shown substantial improvements in computational speed and memory consumption compared to XGBoost, especially when processing large data sets (Ke et al., 2017). GOSS is a data sampling strategy that aims to reduce the number of records used for gradient calculation while maintaining the overall quality of the gradient estimation (Ke et al., 2017). It achieves this by focusing on the records with larger gradients and discarding the records with smaller gradients. In this way, GOSS significantly reduces memory and computation time requirements for gradient calculation. EFB groups features with similar values into bundles, which are then treated as single entities during the tree construction process (Ke et al., 2017). This allows the bundles of features with similar values to be considered as a single unit rather than treating each feature value as a potential split point. Hence, EFB also reduces the dimensionality and allows more efficient memory usage and computation.

As XGBoost, LightGBM uses a greedy algorithm for determining the splits in the decision tree generation, but contrary to XGBoost, LightGBM adopts a **leaf-wise** growth strategy, which grows the tree in depth and selects at each step the leaf for splitting that maximizes the gain (Zhang and Gong, 2020). This method is illustrated in the left box of Figure 8. It can lead to faster convergence and better performance but can also lead to overfitting if it is not properly controlled (Zhang and Gong, 2020).

LightGBM can **handle categorical features** as it employs a strategy for splitting directly also categorical features (Ke, 2023). This is achieved by creating a histogram of the categorical features, sorting it according to its accumulated values, and then finding the best split into two subsets based on the sorted histogram (Ke, 2023). With this strategy, LightGBM effectively handles categorical features while avoiding unbalanced and deep trees that can result from one-hot encoding, especially for categorical features with a large number of distinct categories.

2.2.2.3 Categorical Boosting

The third gradient boosting framework compared in this thesis is CatBoost, which was introduced in 2017 (Prokhorenkova et al., 2017). CatBoost is a gradient boosting framework, which is specifically designed to process categorical features and to address the so-called "prediction shift" (Prokhorenkova et al., 2017).

For **sampling**, CatBoost does not rely on traditional techniques but introduces randomization with a technique called "ordered boosting" to enhance the robustness and generalization of the model (Prokhorenkova et al., 2017). Ordered boosting generates multiple random permutations of the training data and uses them in such an order that the model is always evaluated on other permutations than being trained, which avoids the "prediction shift" (Prokhorenkova et al., 2017).

CatBoost incorporates a **symmetric** leaf growth strategy for its decision trees, which allows an efficient computation and reduces the complexity of the algorithm (Alshari et al., 2021). The tree is built level-wise, and each split in one level is based on the same criterion, which means that the decision of whether to go left or right in the tree is determined solely by the current level and the corresponding feature value. By using this symmetric structure, CatBoost can vectorize the decision-making process, which can substantially improve the computational efficiency.

In general, CatBoost can **handle sparse data** as it assigns the minimum value of a feature to the missing values in the default implementation (Gulin, 2023). However, this simplistic approach might be inappropriate for some data sets (Florek and Zagdański, 2023).

One of the key features of CatBoost is its in-built algorithm for **handling categorical features** using their target statistics (Prokhorenkova et al., 2017). This algorithm calculates the statistics of the target variable for each category and takes these statistics to convert them into numerical values (Prokhorenkova et al., 2017).

2.3 Deep Learning

Deep learning is a subfield of machine learning, which has gained much attention and success, especially in fields that involve large amounts of complex data like images, audio, and text (Taye, 2023). With increasing efforts to adapt deep learning models for tabular data, deep learning also gains popularity here besides traditional machine learning methods (Joseph, 2021). Hence, in addition to the presented gradient boosting methods, FNNs are implemented in this thesis with the PyTorch Tabular framework. In addition, GPT is explained briefly in this chapter, as they are investigated in this thesis for data augmentation and can also be classified as deep learning.

2.3.1 Feedforward Neural Network

An FNN is the simplest structure of a neural network and consists of multiple computational units, also called perceptrons or neurons, organized in consecutive layers, as schematically illustrated in Figure 9. A single neuron computes its output y according to the following equation, where σ denotes the activation function, w the weight, x the input of a neuron, and b the bias:

$$y = \sigma(wx + b) \tag{12}$$

An FNN passes information only in one direction from the input layer through hidden layers to the output layer. It defines a mapping f of an input x to an output $\hat{y}$ and adapts the parameters $\boldsymbol{\theta}$ iteratively during the training process to approximate the function f^*, where x and $\hat{y}$ can also be vectors like in Figure 9 depending on the task (adapted from (Goodfellow et al., 2016)):

$$\hat{y} = f(x; \boldsymbol{\theta}) \tag{13}$$

FNNs are considered as deep learning, when they consist of multiple hidden layers, and hence f can also be interpreted as a chain, where f^1 denotes the first layer and f^l the last layer (adapted from (Goodfellow et al., 2016)):

$$f(x) = f^l(...f^2(f^1(x))) \tag{14}$$

The input data is passed through the layers of the network to compute its prediction. During the training process, a method called backpropagation is used to update the parameters $\boldsymbol{\theta}$ of the FNN, for which the propagation direction is reversed and the gradient of the loss according to Chapter 2.1 with respect to the weights w is calculated. With the gradients, the weights in the network are updated, for example, by using gradient descent as described in Chapter 2.1. These steps are repeated for several epochs until the model's performance converges.

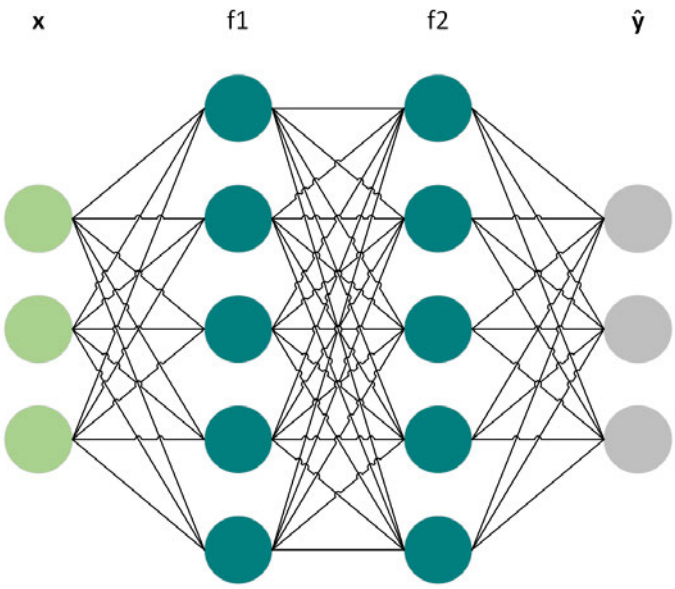


Figure 9: Schematic visualization of an FNN.

PyTorch Tabular

PyTorch Tabular is an open-source framework introduced in early 2023 to simplify deep learning for tabular data (Joseph, 2021). It builds on PyTorch, PyTorch Lightning, and Pandas and provides a comprehensive toolkit to work with tabular data (Joseph, 2021). The framework adopts a "config-driven" approach built around five core configuration files that define the data, the model, the training, the optimization, and the experiments. It enables customizing the machine learning pipelines depending on the data and application requirements (Joseph, 2021). The library also contains options for integrated preprocessing and feature engineering.

PyTorch Tabular has several state-of-the-art model architectures implemented, including the standard FNN used in this thesis (Joseph, 2021).

2.3.2 Generative Pre-trained Transformer

GPT is a method in the field of Natural Language Processing (NLP) that gains increasing popularity in language-related tasks such as text generation, language translation, and sentiment analysis (Yenduri et al., 2023). GPT is based on two concepts, namely pre-training and fine-tuning, to predict the next word in a sequence based on the previous words.

The first stage is to pre-train the language model with unsupervised learning techniques, which means it does not require labeled data for training. Instead, it uses large amounts of unlabeled text data (for example from books, articles, or web pages) to learn the statistical properties of languages, including syntax, semantics, and grammar (Radford et al., 2018). For this, a transformer architecture is investi-

gated. It is a neural network architecture that relies on self-attention mechanisms to capture contextual relationships between words in a sentence (Yenduri et al., 2023). Unlike traditional recurrent neural networks that also process sequential data, transformers can process sentences parallelly, making them highly efficient. The core components of the transformer architecture are an encoder and a decoder. The encoder processes the input sequence while the decoder generates the output sequence. Each component consists of multiple layers, and each layer contains two sub-layers (Radford et al., 2018): a multi-head self-attention mechanism and an FNN. The self-attention mechanism enables the language model to weigh the importance of different words or segments in a sentence (Kardakis et al., 2021).

After pre-training the model, it is fine-tuned for tasks such as text classification, named entity recognition, or machine translation. In fine-tuning, the parameters of a model are adapted in a training process on labeled data for a specific task (Radford et al., 2018). This transfer learning approach allows GPT to utilize its pre-trained knowledge and adapt it to specific applications.

3 Data Analysis and Data Preparation

Data analysis and data preparation are important parts in machine learning tasks as they provide the foundation for building models and extracting meaningful insights. Hence, this chapter is also an integral part of the work in this thesis.

In order to use a data set effectively, it is important to understand its characteristics and adapt processing techniques so that the data is suitable for the respective tasks. Therefore, this section focuses on the investigated data sets and the methods shown in Figure 10, which are applied to prepare them for training the machine learning models.

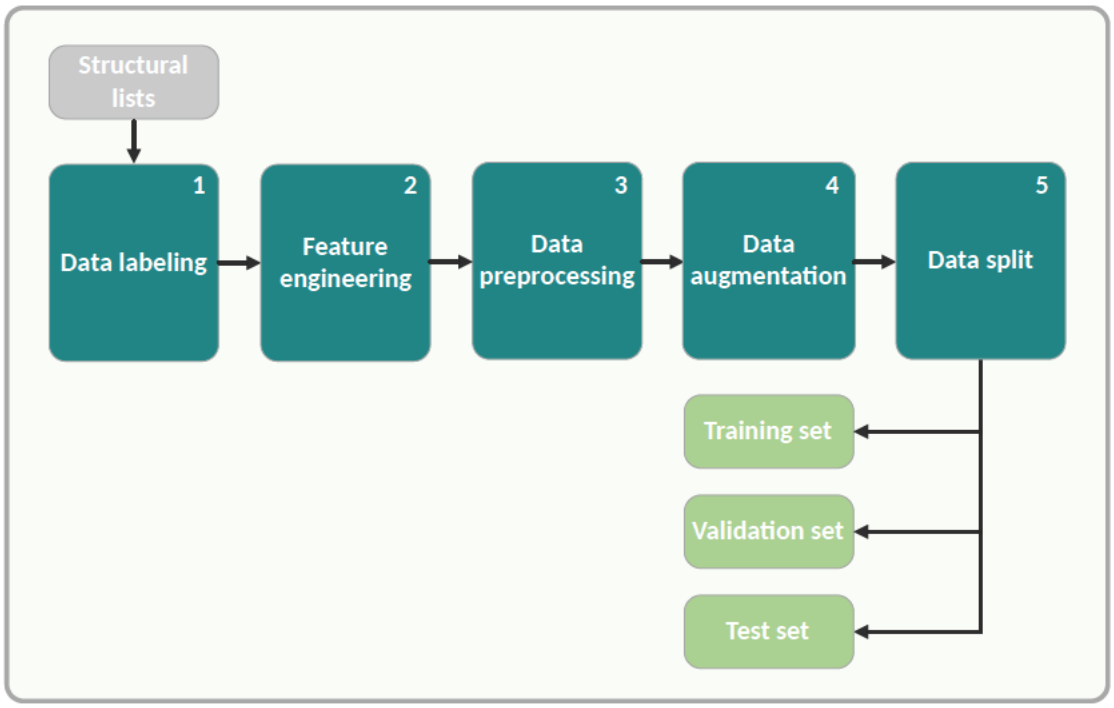


Figure 10: Methods used to prepare the data.

Chapter 3.1 provides a detailed description of the data sets, which includes the type of data, the source, its size, and other relevant characteristics. After that, Chapter 3.2 describes the labeling process (Method 1 in Figure 10) and discusses the classes employed for the machine learning tasks. Subsequently, Chapter 3.3 gives a detailed description of the data preparation, ensuring its suitability for training machine learning models. This includes the feature engineering, which covers the feature selection, transformation, creation, and extraction, the data preprocessing, and the data augmentation (Methods 2-4 in Figure 10). In addition, the class distributions of the labels are analyzed to tailor the preprocessing steps. The section concludes with Chapter 3.4, which covers the strategies for splitting the data (Method 5 in Figure 10) into training, validation, and test sets.

3.1 Data Description

The data used in this thesis consists of structured, tabular data. Tabular data is usually organized in rows and columns, resembling spreadsheets or databases. Each column represents a characteristic or attribute, such as age, income, or categorical variables, like gender or occupation. Each row represents a record, which could be a person or an observation.

The data used in this thesis consists of structural lists of various vehicle models, which are downloaded as Excel files from a database (see Figure 1). Each list contains all components of a specific vehicle model. These components are organized into main modules, including body, exterior, interior, and various submodules. Therefore, the rows of the data sets represent individual car components or their associated hierarchical module structure, while the columns contain information about each record, including the part designation, the part number, a functional key, and additional metadata. The metadata includes attributes like bounding box information, the weight, and the version. An extract from a selected structural list is shown in Table A1 to illustrate the hierarchical structure of the data set. However, the values have been modified for information security reasons.

In total, the combined data set consists of 33 data sets from different vehicle models, each containing exactly 104 attributes and, on average, 5903 records. The record number ranges from 4139 to 8175 and depends mainly on the model and its development stage. The selection of the vehicles includes earlier models, current models as well as models in a development stage to ensure a comprehensive representation. Also, the models are selected such that at least two examples of each vehicle type are included to enhance the generalizability. Table 2 provides an overview of the vehicle types and their distribution within the combined data set before data preparation.

Table 2: Distribution of the car types in the combined data set before data preparation.

Vehicle type	**Number of vehicles**	**Number of records**
Limousine	5	33416
Compact	2	9974
Cabrio	3	18288
Coupé	5	29335
SUV	13	72309
Estate	3	19423
Sport	2	12070
	33	**194815**

3.2 Data Labeling

For the supervised learning tasks implemented in this thesis, labeled data is necessary. As this is not already available, the data needs to be labeled first.

Each record of the data set gets two labels assigned that represent the target outputs for the two tasks. The first label "Relevant fuer Messung" contains the binary classes "Ja" (Yes) and "Nein" (No) and is used to classify whether a given record (car component) is relevant for the measurement process or not. The second label "Einheitsname" assigns uniformly coded designations for the relevant components across all vehicle models. Figure 11 illustrates two examples of the uniformly coded designation labels for the front and back cover window frame. The original designations of the components are shown in grey, and the uniform names in green. This uniform naming allows the CATIA parametric model to accurately match components to their associated dimensions. Depending on the vehicle model, the vehicle contains between 20 and 35 relevant components.

The labeling process is divided into two stages. In the first stage, eight out of 33 data sets are manually labeled. These are directly used to train two machine learning models, one for each label. LightGBM models are utilized for this task, similar to those described later in Chapter 4.2.2, but in an earlier development stage and without implemented hyperparameter tuning. In the second stage, these models are used to pre-label the other 25 data sets, which only require manual confirmation and correction of misclassifications. This process substantially reduces the time required to label the remaining data sets from over five hours to 30 minutes per data set.

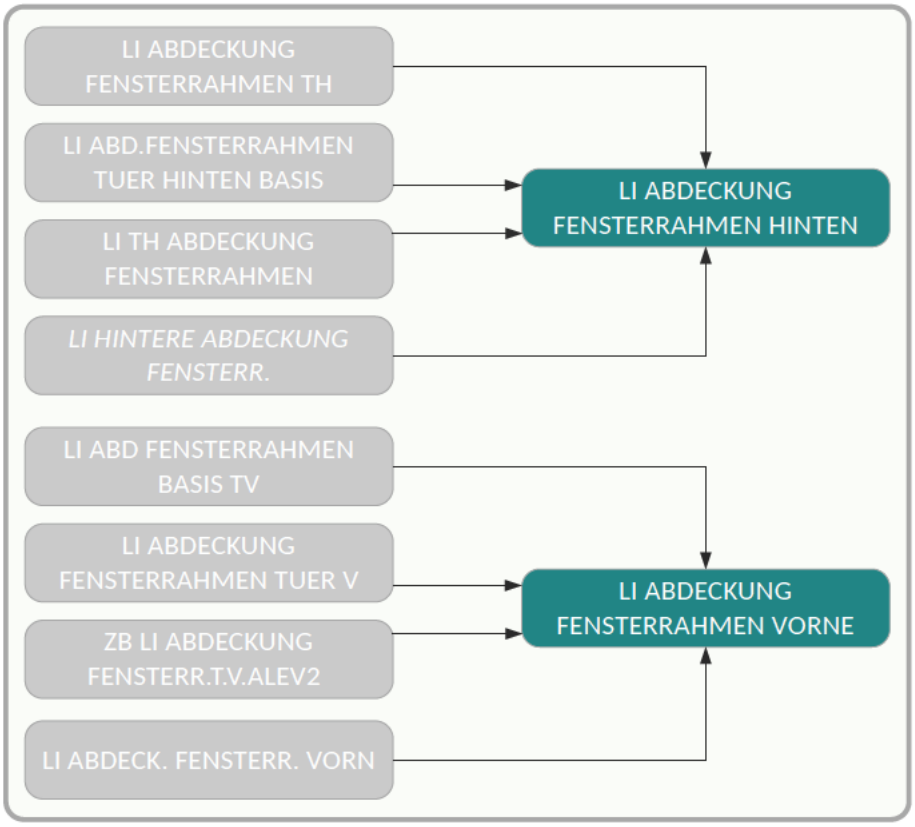


Figure 11: Uniformly coded designations (green) for the original designations (grey) of the cover window frame front (top) and rear (bottom).

3.3 Data Preparation

This chapter details the investigated methods for data preparation, namely feature engineering, data preprocessing, and data augmentation (Methods 2-4 in Figure 10), and additionally provides an analysis of the classes that guides the data preprocessing and data augmentation.

Before the data sets are further analyzed and prepared for training the machine learning models, the records are pre-selected based on simple criteria to standardize the data sets and reduce their size.

Since the dimensional measurements of the CATIA parametric model are based on specific modules, modules that do not contain any relevant component can be excluded. This is achieved with the attributes "Modul (Nr.)" and "Ebene", where the attribute "Ebene" indicates the level of each record in the hierarchical structure, and the attribute "Modul (Nr.)" assigns each record to the corresponding submodule. This combination can be used to keep solely the records from modules with relevant components. Once the not relevant modules have been excluded, all records that are not components are removed. This is achieved with the "Doc-format" attribute, which categorizes each record as a car part or file path to realize the hierarchical structure.

3.3.1 Feature Engineering

Feature engineering is the process of transforming the raw data into features, which enables machine learning algorithms to accurately model the data. This process is critical for the performance of machine learning models, as it identifies the potential relevant features for a given task, creates new features to extend the existing data, and transforms features to a more suitable representation, which highlights relevant information and removes noise.

The feature engineering described in this chapter is demonstrated on the data set of one randomly selected vehicle model but is applied to the other vehicle models as well and includes the feature selection, the feature transformation, the feature extraction, and the feature creation.

3.3.1.1 Feature Selection

For the feature selection, it is crucial to understand the characteristics of the data set and their relevance for a respective task. Therefore, all 104 attributes in the data set, which could be used as potential features for the machine learning models, are assessed with domain experts. Out of the 104 attributes, 31 are identified as potential relevant features and 38 as not meaningful, while 35 are eliminated due to a high rate of missing values.

Table 3 briefly explains the 31 pre-selected features, which are analyzed in more detail in the following. The numerical pre-selected features are specifically assessed

by conducting an exploratory data analysis (EDA). Motivated by Occham's Principle of Parsimony (also known as Occham's Razor), which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything" (Tolin, 2016), only those features that are expected to add substantial information value are selected as input features to limit complexity.

Table 3: List of the 31 pre-selected features with a brief translated description.

Feature	**Description**
Sachnummer	Unique key to reference the record, f.e part number
Zeichnungsindex	Forms the functional key of the version
Doku-Teil	Forms the functional key of the version
Alternative	Forms the functional key of the version
Dok-Format	Forms the functional key of the version
Bennennung (dt)	Car component designation
Kurzname	Short name of the component
L/R-Kz.	Left/right markers for symmetrical component
Modul (Nr)	Module number
Ebene	Level of the record in the hierarchy of the structural list
Code	Development code to which the vehicle model belongs
X-Min	X-Min of the bounding box
X-Max	X-Max of the bounding box
Y-Min	Y-Min of the bounding box
Y-Max	Y-Max of the bounding box
Z-Min	Z-Min of the bounding box
Z-Max	Z-Max of the bounding box
ox	x-value of the shift vector
oy	y-value of the shift vector
oz	z-value of the shift vector
xx	xx-value of the rotation matrix
xy	xy-value of the rotation matrix
xz	xz-value of the rotation matrix
yx	yx-value of the rotation matrix
yy	yy-value of the rotation matrix
yz	yz-value of the rotation matrix
zx	zx-value of the rotation matrix
zy	zy-value of the rotation matrix
zz	zz-value of the rotation matrix
Wert	Weight of the car component
Einheit	Weight unit

Functional key: The features "Sachnummer", "Zeichnungsindex", "Doku-Teil", "Alternative", and "Dok-Format" are categorical and form the functional key for referencing records. While not suitable as an input to the machine learning models,

these features play an important role in loading the relevant components into the CATIA parametric model. Therefore, they are used in a later stage of the AI system but are not considered as potential input features for the machine learning models.

Designation: "Benennung (dt)" is a textual feature representing the German designation of the component, which contains relevant information for the classification task. The challenge with this feature stems from the lack of consistency in the component naming across different vehicle models, as engineers are free to choose the name of the components. Nevertheless, there are similarities in the naming, and the feature contains no missing values, which is why the feature is selected as an input feature but requires extensive further preparation.

Short name: The categorical feature "Kurzname" represents a short name (one word) of the German designation and could serve as a valuable feature for training purposes. However, engineers are not required to fill this field during the virtual vehicle development, leading to potential missing input values and inconsistencies also during inference. Additionally, the short name does not provide sufficient details about the position of the part in the vehicle, such as front or back. Hence, this feature is not selected as an additional input feature to the complete designation.

Symmetrical components: The "L/R Kz." feature indicates whether a component is symmetrical and used on both the left and right sides of a vehicle. After consulting with experts, it was determined that only the left-hand versions of these components are necessary for the measurements, as they can be mirrored in the parametric model if needed. Therefore, this feature is used for preprocessing (see Chapter 3.3.3) but not as an input feature for the machine learning models.

Module number and level: As mentioned at the beginning of this section, the "Modul (Nr)" and "Ebene" features are used to filter the data sets for relevant modules. Therefore, the data sets are already filtered by this feature, and the expected additional information for the machine learning models is considered low. Hence, this feature is not selected as an input feature for the models.

Development code: The feature "Code" specifies the vehicle model and is unique for each vehicle model. This means that the feature has the same value for all components of one vehicle model and is therefore not considered as a valuable feature for the machine learning models. However, this feature is used as a reference in the final, combined data set over all 33 vehicle models to be able to assign the records to the initial vehicle models.

Bounding box: The bounding box of a component is defined by the numerical features "X-Min", "X-Max", "Y-Min", "Y-Max", "Z-Min", and "Z-Max", where the difference in x-direction represents the length, in y-direction the width, and in z-direction the height of the component's bounding box as it is aligned with the coordinate axes. The coordinate system for these values is set equally across all vehicle models according to the Society of Automotive Engineers (SAE) convention, having the origin at the midpoint of the front axle at ground level.

By investigating the boxplots of the bounding box features shown in Figure 12, one value for each feature is considerably distant from the other values, including the

outliers, which all stem from one component. Depending on the vehicle model, up to nine components are affected. Following consultation with domain experts identified that the anomalies are system errors, and hence, the respective values are set to zero for these components.

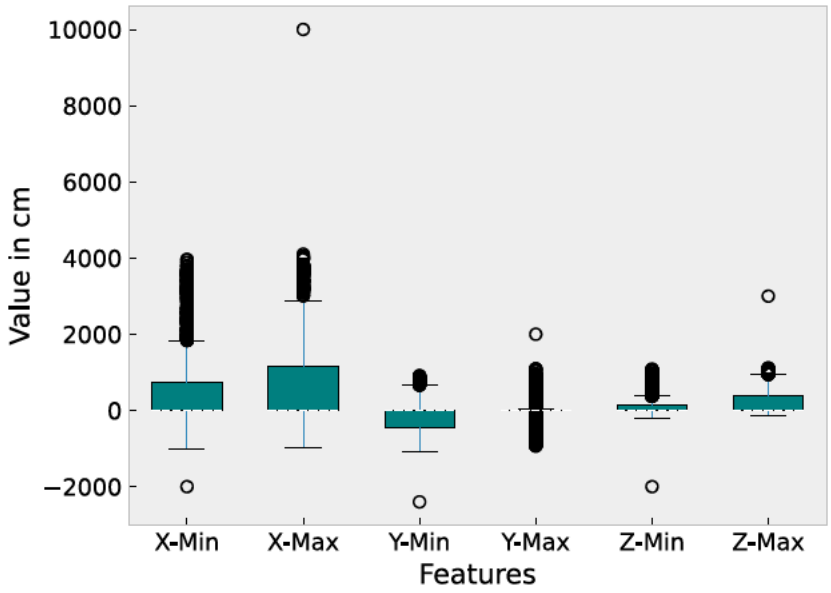


Figure 12: Boxplot of the features "X-Min", "X-Max", "Y-Min", "Y-Max", "Z-Min", "Z-Max" from a randomly selected vehicle model.

Shift vector and rotation matrix: In addition, a shift vector ("ox", "oy", "oz") and a rotation matrix ("xx", "xy", "xz", "yx", "yy", "yz", "zx", "zz") provide further options to adjust and position the bounding box in the coordinate system. These features do not have any missing values but are often zero. For the anomalies found in the previous step, it was verified that the values are still considerably distant after shifting and rotating, and hence, for these errors also the features for the shift vector and the rotation matrix are set to zero.

By calculating the correlation between the bounding box features, it is obvious that some features are positively correlated, which shows that these variables provide similar information and could be redundant to an extent. Therefore, the idea is to represent the 18 bounding box features with fewer features as described in Chapter 3.3.1.3.

Weight: The last two analyzed features "Wert" and "Einheit" specify the weight and the corresponding unit of the component. The categorical column for the unit has two unique values "g" and "g/m", which stand for grams or grams per meter. More than 98% of the components have the unit grams, which is why no distinction is made by the unit. For the numeric value feature, the boxplot in Figure 13 shows that the components from relevant modules consist of many light components but

also many components with a zero value. Domain experts deduced that weights are truncated, and components such as small screws or springs with a weight of less than one gram are assigned to zero. In addition, the weights for components that are at an early stage of development are not yet available and are therefore set to zero. Both features do not have missing values, however, only the value feature for the weight is utilized as an input feature for the machine learning models.

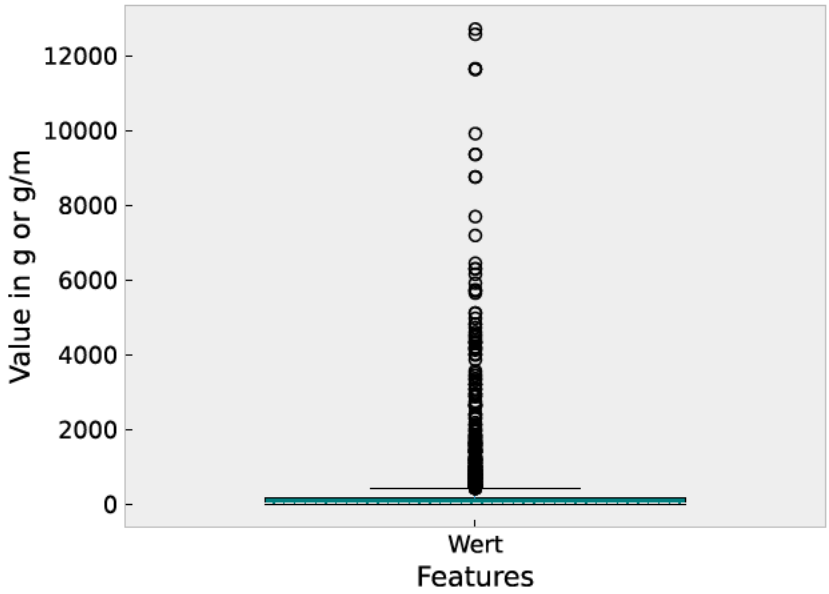


Figure 13: Boxplot of the feature "Wert" from a randomly selected vehicle model.

Based on the previously explained analysis, "Benennung (dt)", the features corresponding to the bounding box, and "Wert" are selected for further preparatory steps as they have the potential to be used as input features for the machine learning models. "Ebene", "Sachnummer", "Zeichnungsindex", "Doku-Teil", "Alternative", "L/R-Kz.", and "Code" are not used as input features but are kept in the data set as they are required for referencing purposes and preprocessing steps.

As outlined in the subsequent chapters, a feature transformation is applied to the textual feature "Benennung (dt)", a feature extraction and feature creation is carried out for the features corresponding to the bounding box, and no additional modification is performed for the "Wert" feature.

3.3.1.2 Feature Transformation

This chapter focuses on the process of transforming the textual feature "Benennung (dt)". Firstly, it undergoes a series of cleaning steps to prepare it for further analysis. This involves converting all letters to uppercase and removing punctuation as well as numerical digits. Additionally, manually predefined words and abbreviations

that occur often but have no information, like “AF” or “ZB” are deleted. By applying these cleaning techniques, the feature is made more suitable for subsequent transformations, and additionally, the number of unique words is reduced.

Following the cleaning process, the feature is transformed further using the CountVectorizer technique, which is part of the scikit-learn library in Python (Pedregosa et al., 2011). The CountVectorizer is a commonly used text preprocessing technique in NLP to convert text data into a numerical representation that machine learning algorithms can use. The main idea behind CountVectorizer is to convert text into a vector of word frequencies. Therefore, it creates a vocabulary of unique words from the given text and assigns an index to each word. Then, it counts in the text of the record the number of occurrences of each word and transforms it into a vector, where the count for each word is denoted at the corresponding index. Instead of words, character n-grams with a range between three and eight characters are used to account for minor variations in the words, abbreviated versions of words, or spelling or typographical errors. The char-analyzer and the range for the n-grams enable to capture both short and long character sequences. This is particularly beneficial as car component designations can vary in length and may contain important information at different character levels. Considering a wide range of n-grams increases the chance of capturing relevant patterns and relationships within the data. As it is more relevant for the task which n-grams are present, it is appropriate that CountVectorizer does not capture semantic relationships between n-grams or take their order into account.

Since the number of features increases exponentially with the amount of unique n-grams in the text, the vectorizer is fitted solely with the designations of the components that are labeled as relevant. This approach substantially reduces the number of features to represent the designations and ensures that the vectorizer learns particularly from the relevant records.

3.3.1.3 Feature Extraction

Feature extraction is an essential technique for transforming raw data into a more meaningful representation that machine learning algorithms can effectively utilize. In the context of the given problem, feature extraction is performed on the features with the bounding box information (“X-Min”, “X-Max”, “Y-Min”, “Y-Max”, “Z-Min”, “Z-Max”), the shift vector (“ox”, “oy”, “oz”), and the rotation matrix (“xx”, “xy”, “xz”, “yx”, “yy”, “yz”, “zx”, “zy”, “zz”). Out of these features, the length, width, and height, the center point, and the orientation of the bounding box are calculated to reduce the number of features for representing the bounding box.

Dimension: The length, width, and height of the bounding box are calculated from the minimum and maximum values before the bounding box is shifted and rotated as it is aligned with the coordinate axes in this step:

$$length = X_{\max} - X_{\min} \tag{15}$$

$$width = Y_{\max} - Y_{\min} \tag{16}$$

$$height = Z_{\max} - Z_{\min} \tag{17}$$

For generating further features, the edge points of the rotated and shifted bounding box are calculated. This transformation is performed as follows: Firstly, a matrix of corner points C is defined using the minimum and maximum values of the bounding box:

$$\text{Corners} = C = \begin{bmatrix} X_{\min} & Y_{\min} & Z_{\min} \\ X_{\min} & Y_{\min} & Z_{\max} \\ X_{\min} & Y_{\max} & Z_{\min} \\ X_{\min} & Y_{\max} & Z_{\max} \\ X_{\max} & Y_{\min} & Z_{\min} \\ X_{\max} & Y_{\min} & Z_{\max} \\ X_{\max} & Y_{\max} & Z_{\min} \\ X_{\max} & Y_{\max} & Z_{\max} \end{bmatrix} \tag{18}$$

Then, the rotation matrix R is applied to the corner points, resulting in rotated corner points $C_{rotated}$:

$$C_{rotated} = C \cdot R = C \cdot \begin{bmatrix} \text{xx} & \text{xy} & \text{xz} \\ \text{yx} & \text{yy} & \text{yz} \\ \text{zx} & \text{zy} & \text{zz} \end{bmatrix} \tag{19}$$

To get the transformed corner points $C_{transformed}$, the shift vector σ is added to the rotated corner points $C_{rotated}$:

$$C_{transformed} = C_{rotated} + \sigma = C_{rotated} + \begin{bmatrix} \text{ox} & \text{oy} & \text{oz} \end{bmatrix} \tag{20}$$

With the transformed corner points, the following features are calculated:

Center points: The center points of the transformed bounding box are calculated by taking the mean of the x-, y-, and z-coordinates of the transformed corner points:

$$center_x = \frac{\sum_{i=0}^{7} C_{transformed}[i][0]}{8} \tag{21}$$

$$center_y = \frac{\sum_{i=0}^{7} C_{transformed}[i][1]}{8} \tag{22}$$

$$center_z = \frac{\sum_{i=0}^{7} C_{transformed}[i][2]}{8} \tag{23}$$

Euler angles: The orientation of the transformed bounding box is determined by calculating the Euler angles θ_x, θ_y, and θ_z from the rotation matrix R according to the equations by Slabaugh (2020).

By performing the previously explained calculations, the number of features to represent the bounding box information is reduced from 18 to nine features for the length, width, and height, the center point, and the orientation of the bounding box.

3.3.1.4 Feature Creation

From the length, width, and height of the bounding box, the volume is derived, which provides a comprehensive measure of the spatial extent. The volume of a bounding box can be calculated by multiplying its length, width, and height.

The correlation matrix for the features length, width, height, and volume is shown in Figure A2. It reveals a moderate positive correlation between the volume and the length and a weaker positive correlation between the volume and the width and the volume and the height. However, it cannot be concluded that the volume can be used as a substitute for the dimensional features during model training. Hence, all four values are used as input features.

All resulting bounding box features (length, width, height, center point, volume, and orientation) are further analyzed with a correlation matrix. However, they do not show substantial correlations and, therefore, are all retained as input features for model training.

To conclude this chapter, Table 4 shows the resulting features from feature engineering, which are used as inputs for the models.

Table 4: List of the selected features after feature engineering.

Features used as input for the models
"Bennennung (dt)" (vectorized)
$center_x$
$center_y$
$center_z$
θ_x
θ_y
θ_z
$length$
$width$
$height$
$volume$
"Wert"

3.3.2 Label-based Analysis

Now that the features are prepared and selected, this chapter analyzes the labels of the data set. This helps to understand the distribution and characteristics of the

different classes within the data set and provides insights to make informed decisions for proceeding with data preprocessing and data augmentation.

For this, the combined data set from all vehicle models is investigated. This consolidation allows to have a comprehensive view of the data and perform consistent preprocessing steps across all records. The combined data set contains 101140 records out of 33 structural lists of various vehicle models.

The distribution of the label "Relevant fuer Messung" in Figure 14 shows that the class "Ja", which represents the relevant components, is highly underrepresented in the data set. This indicates an imbalance in the data set, which can potentially affect the performance of the machine learning models. To address this issue, data preprocessing methods, as described in Chapter 3.3.3, are implemented such that the number of records labeled as not relevant is further reduced.

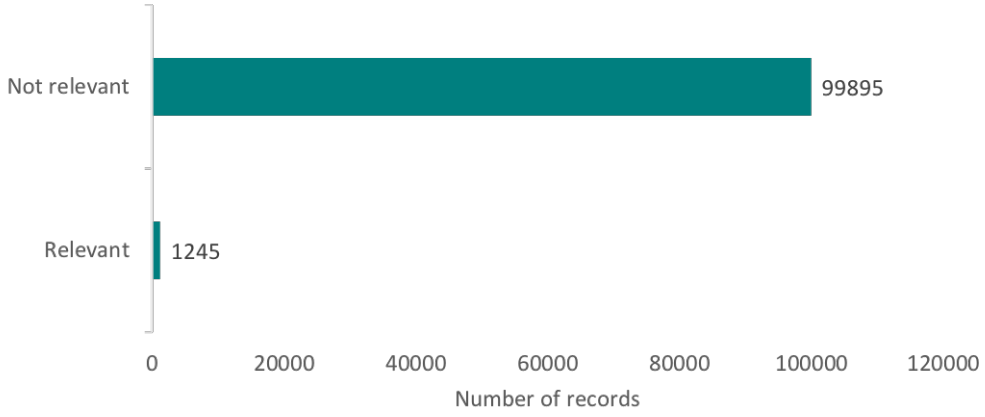


Figure 14: Distribution of the label "Relevant fuer Messung".

The distribution of the uniformly coded designations ("Einheitsname") of relevant components reveals that components appear in the data set with varying frequency. For example, due to the low number of convertibles, the convertible top ("VERDECK") is less represented in the data set than others. Therefore, when dividing the data into a training, validation, and test set, it is necessary to ensure that the relevant components are distributed evenly by the uniform names. However, this is not possible if a class is too rare in the data set. Therefore, methods have been implemented as described in Chapter 3.3.4 to generate synthetic data for the affected components.

3.3.3 Data Preprocessing

As mentioned in the previous Chapter 3.3.2, the data set includes substantially more components labeled as not relevant than relevant and is therefore highly imbalanced. To counteract this, methods for identifying and excluding not relevant components are presented in this chapter.

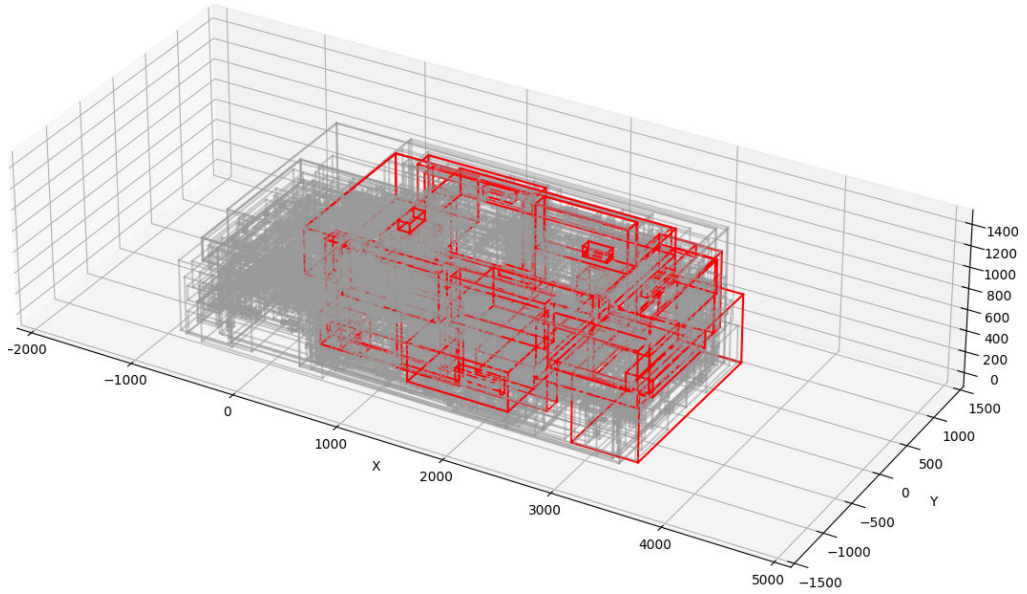


Figure 15: Bounding box visualization of a selected vehicle with relevant components highlighted in red.

Figure 15 shows all bounding boxes of the selected modules of an example vehicle model. The bounding boxes highlighted in red are the relevant labeled components, and the bounding boxes shown in grey are the not relevant components. This plot is the starting point to illustrate further effects of the methods explained in the following.

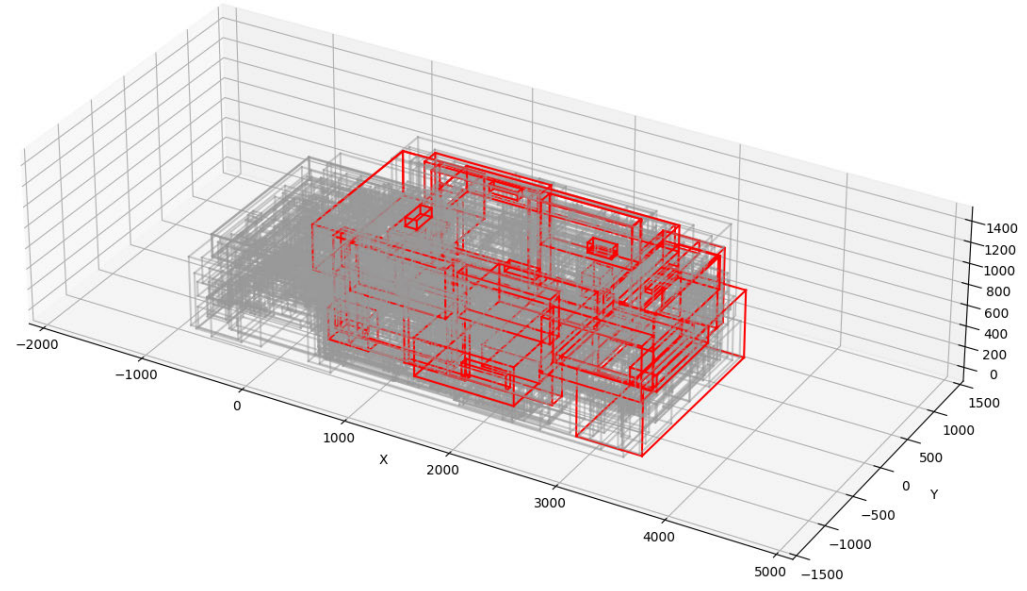


Figure 16: Bounding box visualization of a selected vehicle with relevant components highlighted in red after filtering by volume.

Filtering by volume: Looking at the relevant components in Figure 15, it seems that they all have a relatively large size. This is supported by analyzing the minimum and maximum volume values of the relevant components from 503 cm^3 to 3.31 m^3 respectively compared to the not relevant components from 0.00 mm^3 to 11.0 m^3 in the combined data set. The highest bounding box volume of the relevant components refers to the convertible top, and the not relevant ones are PVC seams and vehicle lacquers. Due to this finding, it is implemented that all components with a volume smaller than 450 cm^3 and components with a higher volume than 4.00 m^3 are removed. These values contain a buffer to compensate for changes in the size of relevant components. Figure 16 shows the remaining components for the example vehicle. The difference to Figure 15 is not easily visible, but in general, it can be observed that the grey from the not relevant components is lighter. In the combined data set the record size of the not relevant components is reduced from 99895 to 34134.

Filtering by position: The data set is further filtered by the position of the records. As observable from Figure 15, no relevant component is located in the front part of the vehicle. The first component that is relevant from the front is the windshield. Therefore, all components located in the first 10% of the length of the vehicle are removed, which further reduces the number of records of the not relevant components from 34134 to 29712 records in the combined data set. Figure 17 shows the remaining components after this filter criterion. The difference to Figure 16 is clearly visible as no bounding box remains at the front part of the vehicle.

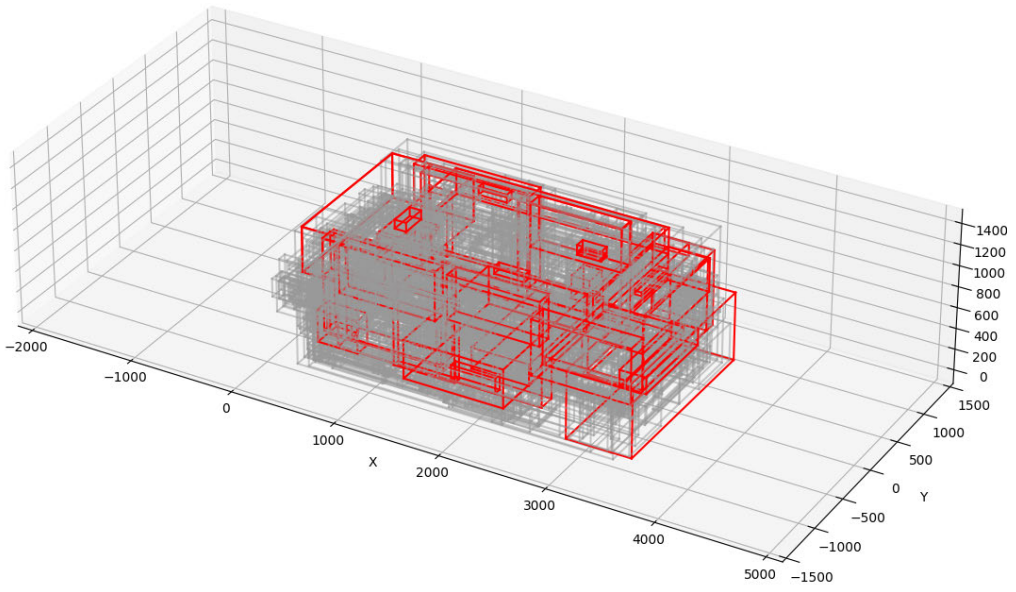


Figure 17: Bounding box visualization of a selected vehicle with relevant components highlighted in red after filtering by position.

Filtering symmetric components: As mentioned in Chapter 3.3.1.1, only the parts in the left half of the vehicle are necessary if they are identical on both sides. The reason is that the components can be mirrored in the CATIA parametric model

based on the "L/R-Kz" feature. By removing the mirrored components on the right side, the number of records of the not relevant components in the combined data set is further reduced from 29712 to 19673, which is shown in Figure 18. Compared to Figure 17, the right side of the vehicle contains fewer bounding boxes.

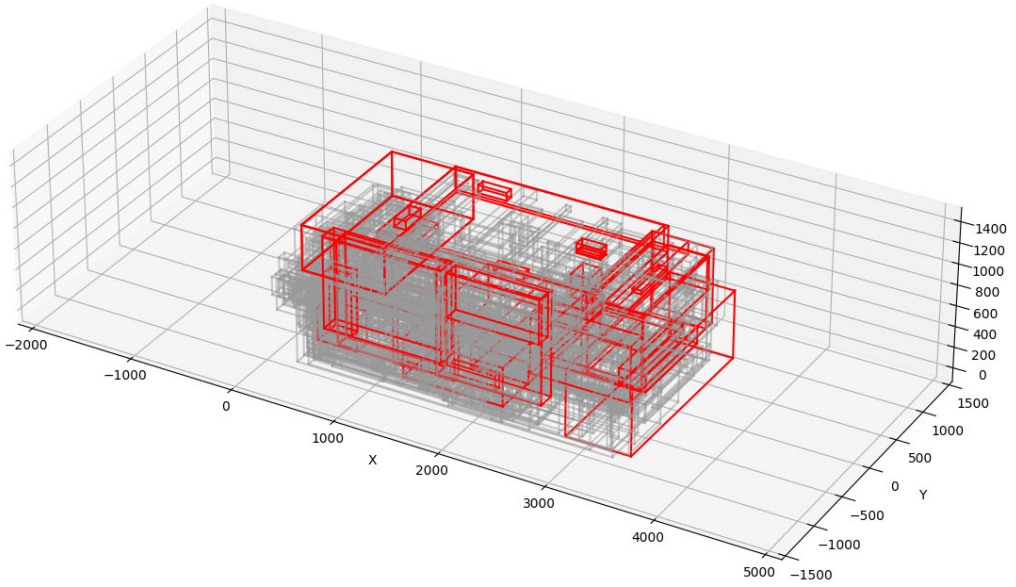


Figure 18: Bounding box visualization of a selected vehicle with relevant components highlighted in red after deleting mirrored components.

Dropping duplicates: The last preprocessing step is to remove all duplicates in the data set such that no component is used twice across vehicle models, which reduces the number of records of the not relevant components from 19673 to 19478 in the combined data set.

By the preprocessing performed with the described filter criteria, the number of not relevant components is reduced from 99895 to 19478 records, while the number of relevant components is reduced from 1245 to 919. The new distribution is substantially more balanced than before. However, the data set is still highly imbalanced, which must be taken into account in splitting the data into training, validation, and test sets to train and evaluate the models.

3.3.4 Data Augmentation

Chapter 3.3.2 also discusses the imbalanced and underrepresentated classes within the label "Einheitsname". This chapter aims to generate synthetic records for the underrepresented components so that these can also be distributed to the training, validation, and test sets with the same split ratio. The process of splitting the data as described later in 3.4 requires a minimum of seven records such that in the validation and test set, the number of records per class can be one. With the explained split, five of the seven records (70% of seven records rounded up to

full records) are assigned to the train set, and one of the seven records each to the validation and test set. For a number below seven records, assigning one record each to the validation and test set would not be possible with the performed split. Hence, for the combined data set, the convertible top ("VERDECK") with a record number of six requires one synthetically generated record, and in the stage where solely eight data sets out of 33 are used, several more components require synthetically generated records.

For this purpose, synthetic records are created with the augmentation techniques described in the following until the required number of records is reached. However, only the designation and the bounding box information are generated. All other features are copied from an original component.

Generating designations: To generate a synthetic designation, one of the three following methods is randomly selected: One method randomly reorders the words of an original designation to create a new one. This method introduces variation while preserving the original designation. A second method swaps two characters in an original designation to generate a spelling error, further diversifying the data set. The third method utilizes GPT-3.5 to generate a new designation similar to an original designation with the same uniformly coded designation label.

The GPT model, hosted on Azure, is configured with a temperature of 0.4 and a maximum token limit of 15. The temperature parameter controls the randomness of the model's output, where 0 gives an almost deterministic and 1 a highly varied response (Yenduri et al., 2023). Here 0.4 is chosen, which allows moderate variation in the generated designations. The maximum token limit of 15 specifies the maximum length of the generated text, where each token is about four characters. By configuring the GPT model with these settings, the aim is to achieve a balance between adding variations and keeping key information in the designation.

The prompt given to GPT is structured using the elements task, context, instructions, and examples, as illustrated in Table 5. The task involves generating modified car component designations based on the input while ensuring that the output differs slightly from the input. The context of the prompt emphasizes that the original designations are written in German and may contain abbreviations and technical terms. It also includes a list of common abbreviations found in the data set, such as "HI" for "HINTEN" (rear), "VO" for "VORN" (front), "HKL" for "HECKKLAPPE" (rear trunk), and others. These abbreviations serve as a reference for creating the modified designations.

The instructions given to GPT guide its behavior during the generation process. GPT is instructed to carefully analyze the input and comprehend its meaning. It is then tasked with modifying the designation to create a new version while maintaining the same meaning. Additionally, GPT is instructed to avoid generating a modified designation identical to the input data.

To further enhance the performance of GPT, a few-shot prompting technique is employed. This technique facilitates contextual learning by incorporating relevant examples in the prompt.

Table 5: GPT prompt to generate synthetic component designations.

Element	**Description**
Task	Write a modified car component designation based on the given input. The output must be slightly different to the input while retaining the same meaning.
Context	The original designation is written in German. It may contain abbreviations and technical terms. Do not add or remove words. Abbreviations that often appear in the data set are: “HI” for “HINTEN”, “VO” for “VORN”, “HKL” for “HECKKLAPPE”, “GPR” for “GEPAECKRAUM”, “VKL” for “VERKLEIDUNG”, “ISP” for “INNENSPIEGEL”, “TV” for “TUER VORNE”, “TH” for “TUER HINTEN”, “HBL” for “HOCHGESETZTE BREMSLEUCHTE”, “STF” for “STOSSFAENGER”, “KST” for “KOPFSTUETZE”, “ND” for “NORMALDACH”, “DAHAUBE” for “DACHANTENNENHAUBE”.
Instructions	1. Carefully examine the input and understand its meaning. 2. Modify the designation to create a new version which is very close to the original. 3. Make sure that the meaning of the designation remains the same. 4. Make sure that the modified designation is not equal to the input data.
Examples	Original: ABDECKLEISTE EINSTIEG HI Modified: ABDECKLEISTE EINSTIEG HINTEN Original: KANTENSCHUTZ TUER VORN Modified: KANTENSCHUTZ TV Original: MD KST VERSTELLBAR AUSSEN MAT Modified: MD KST AUßENMATTE VERSTELLBAR Original: SEITENVERKLEIDUNG Modified: SEITENVERKL Orignal: ABDECKUNG FENSTERRAHMEN TUER VORNE Modified: ABDECKUNG FENSTERRAHMEN TUERE VORN
Input	Original designation

Calculating bounding box information: In addition to generating a designation, the bounding box information for the synthetic records is calculated. Firstly, a valid space is defined based on the other components with the same class label. Secondly, a random bounding box is generated in this space, which has similar dimensions to the original components. This means that the components from the data set of the same class are analyzed to get the range of minimum and maximum values in x-, y-, and z-direction. A new bounding box with a similar volume is then generated within this range.

Once the necessary number of synthetic records is generated, the data can be split among the training, validation, and test set, which is described in more detail in the following chapter.

The resulting distributions after data preparation of the label Relevant fuer Messung are shown in Figure 19 and of the label "Einheitsname" of relevant components in Figure 20.

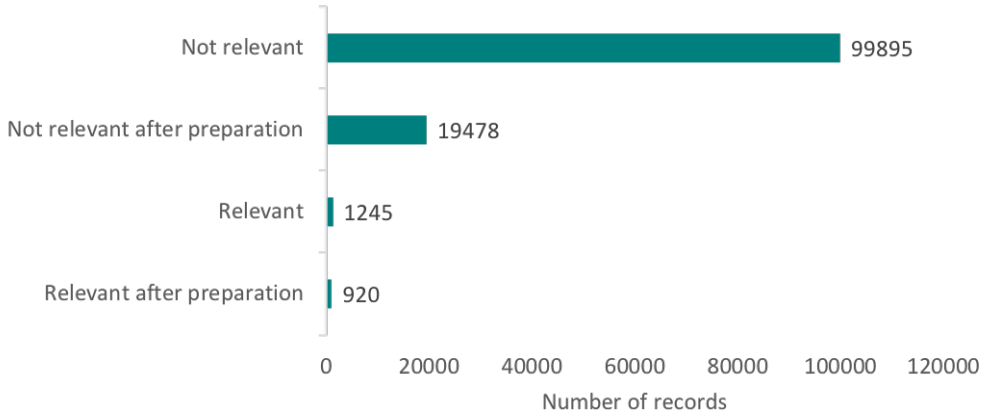


Figure 19: Distribution of the label "Relevant fuer Messung" after preparation.

3.4 Data Split

This chapter discusses the process of dividing the data set into training, validation, and test sets for the two different classification tasks. One task is a binary classification task with the label "Relevant fuer Messung", while the other is a multi-class classification task with the label "Einheitsname". To ensure representative and unbiased training, validation, and test sets, a stratified split approach is implemented. This approach considers the class labels and ensures that the distribution of classes remains consistent across the splits. Maintaining the proportional representation of each class reduces the risk of introducing bias into the model during training. The initial split of the data set involves allocating 70% of the data to the training set, 15% to the validation set, and 15% to the test set. This split allows the model to learn from a substantial part of the data while reserving a separate validation set for hyperparameter optimization and model selection. Additionally, the test set enables evaluating the model's performance on unseen data and assessing its generalization capabilities. For splitting the data for the binary task, the entire preprocessed data is used. The resulting class distribution across the splits is visualized in Figure A1. For the multi-class task, only the records labeled as relevant are investigated.

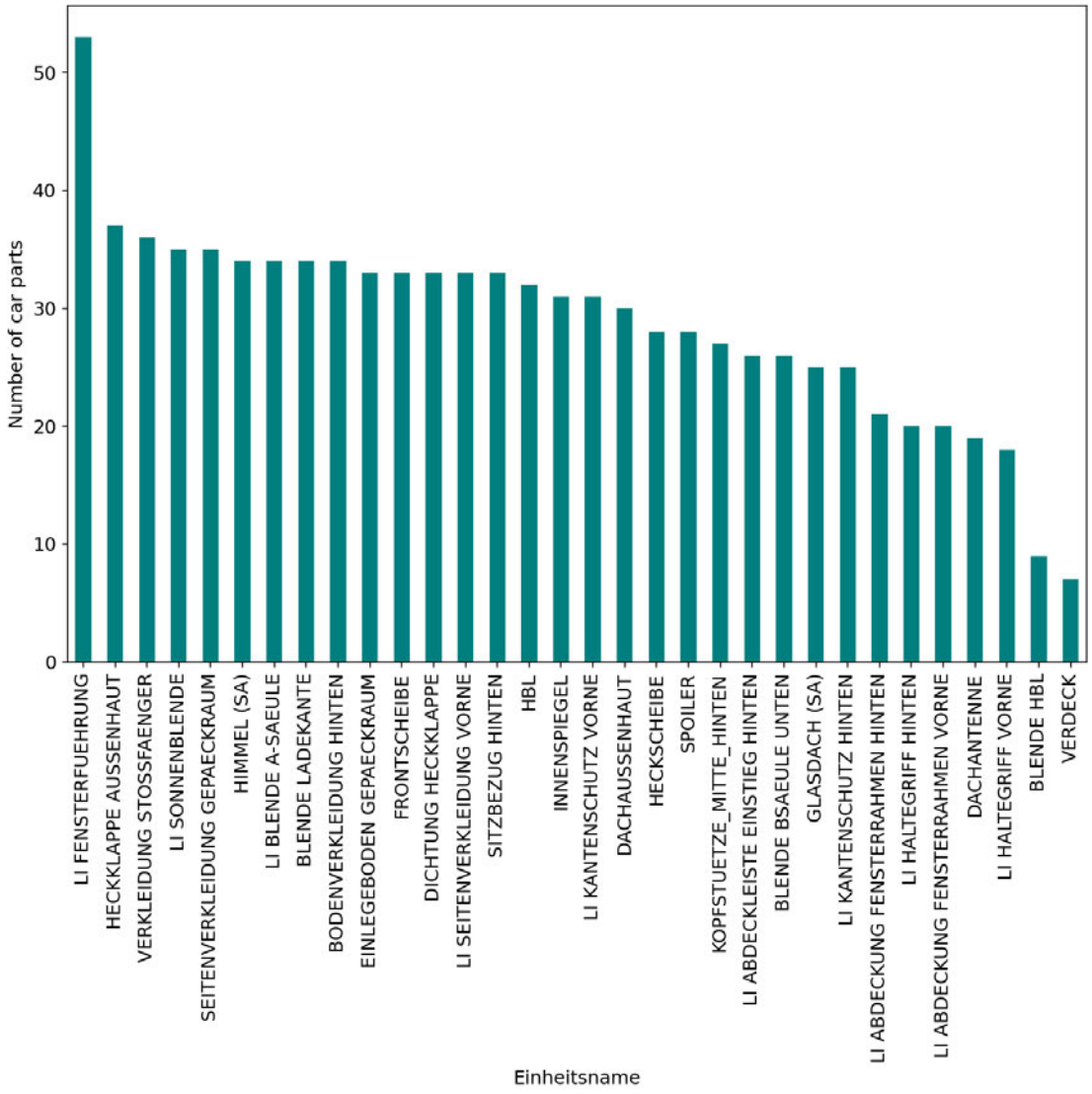


Figure 20: Distribution of the label “Einheitsname” of relevant components after preparation.

4 Method

This chapter explains the method for selecting suitable inference models and details the implementation of the compared models. As explained, two different tasks should be addressed with the models, which are briefly summarized in the following.

Binary classification task - Component relevance: The first goal is to derive a model that can classify the pre-filtered components as relevant and not relevant for geometric measurements. This is realized as a binary classification task. For an input feature vector $\boldsymbol{x}_i$ consisting of the features for one component from Table 4, the target output value y_i is either one ("Ja" (Yes) - relevant component) or zero ("Nein"(No) - not relevant component) according to the labeling performed in Chapter 3.2 and the label encoding. The goal is that the prediction $\hat{y}_i$ of a model is equal to y_i for all N components in the data set that should be classified.

Multi-class classification task - Uniformly coded designations: The second goal is to derive a model that generates uniformly coded designations for relevant components for the geometric measurements. This is realized as a multi-class classification task. For an input feature vector $\boldsymbol{x}_i$ consisting of the features for one component from Table 4, the target output value y_i is a number between zero and 31 according to the labeling performed in Chapter 3.2 and the label encoding for the 32 possible "Einheitsnamen". The goal is that the prediction $\hat{y}_i$ of a model is equal to y_i for all M components classified as relevant in the data set, which should get a uniformly coded designation.

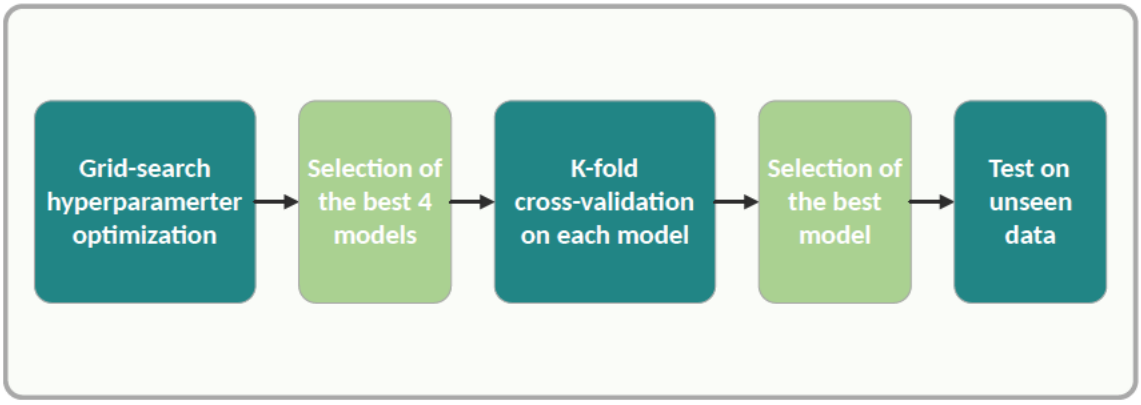


Figure 21: Process to find the optimal method and model for each task.

Figure 21 illustrates the process that is set up to develop the models for both tasks. In total, the visualized process is executed once for each implemented framework and task. The process begins with hyperparameter tuning of the respective model using grid-search, which systematically trains the model with different combinations of hyperparameters and evaluates them to optimize the parameters. Based on the performance on the validation set, the best four models (top 5%) are selected. For a more robust estimate of the models' performances, these models are further evaluated using k-fold cross-validation. The model that scores highest here is chosen as

the best model of a framework and is further evaluated on the test set. To use all data for the inference model, the best model selected from comparing the methods is then trained on a data set, which combines the training and test set.

This chapter begins with a brief description of the development environment in Chapter 4.1, which includes the hardware and the most important frameworks used in this thesis. The first step of the process is presented in Chapter 4.2 and shows the implementations of the different machine learning methods, the used parameters, and how the hyperparameters are optimized using grid-search. The other steps of selecting and evaluating the models are explained afterward in Chapters 5 and 6.

4.1 Development Environment

This thesis is implemented in Python on a local machine with the following hardware:

- Processor: Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz, 2.30 GHz
- RAM: 16 GB
- Operating System: Windows 10 Enterprise (64-bit operating system, x64-based processor)

According to IEEE Spectrum, Python is currently the world's most widely used programming language (Cass, 2023) and offers numerous advantages, particularly for developers in machine learning and AI, due to its extensive collection of open-source libraries. In this thesis, the gradient boosting models are built with the frameworks XGBoost (Version 1.7.6), LightGBM (Version 3.3.5), and CatBoost (Version 1.2.1). For comparison, Pytorch Tabular (Version 1.0.2) is used as a deep learning framework, along with Pytorch (Version 1.13.1), for building and training an FNN. Additionally, the SHapley Additive exPlanations (SHAP) framework (Version 0.41.0) is employed for interpreting the machine learning models, and Pytest (Version 7.4.0) is utilized to write and execute unit tests. Besides, docker and Amazon Web Services (AWS), FastAPI (Version 0.97.0) and Streamlit (Version 1.23.1) are used for the deployment of the AI system.

4.2 Model Implementation

This chapter focuses on the implementation of the machine learning models. To limit the description per framework, Chapter 4.2.1 presents general settings implemented identically for the tasks and methods. Afterward, Chapter 4.2.2 gives the implementation details for the ensemble learning methods, followed by Chapter 4.2.3 for the deep learning method.

4.2.1 General Concepts

The following concepts are implemented for all frameworks and models of this thesis. This includes the metrics, the hyperparameter optimization technique, and the parameters, which are used across all methods.

4.2.1.1 Metrics

In addition to the cross-entropy loss, which is described in Chapter 2.1 and used for optimizing all models, the models for the binary classification task additionally calculate the F_2-score in each training iteration, which is explained in the following. For the multi-class classification task, the F_2-score is only calculated for the fully trained models as XGBoost has no option yet to implement a loss in combination with an additional custom metric.

Sensitivity

Sensitivity, also known as recall or true positive rate, measures the percentage of correctly classified positive records out of the total number of actual positive records:

$$\text{Sensitivity} = \text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{24}$$

It quantifies the model's ability to correctly identify the positive records. In tasks where the focus is on identifying all positive records, sensitivity is an important metric. For the binary classification task in this thesis, a high sensitivity score indicates that the model effectively classifies relevant components and especially classifies a few relevant components incorrectly as not relevant. However, this metric does not consider the not relevant components incorrectly classified as relevant.

Precision

Precision measures the percentage of correctly classified positive records out of the total number of records predicted as positive:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{25}$$

It quantifies the model's performance on the positive predictions. In tasks where the focus is on the accuracy of the positive predictions, precision is an important metric. For the binary classification task in this thesis, a high precision score indicates that the model provides accurate predictions for the relevant components and especially classifies few not relevant components as relevant. However, it does not consider the relevant components incorrectly classified as not relevant.

F_1-score

To combine the sensitivity and precision, the F_1-score is a commonly used metric for evaluating a model's performance, especially in imbalanced tasks, where both metrics are important:

$$F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} \tag{26}$$

It is the harmonic mean of precision and sensitivity, which provides a balanced measure of the model's ability to correctly classify both positive and negative records. The F_1-score ranges from zero to one, where a value of one indicates perfect precision and sensitivity. In imbalanced tasks, where the focus is often on correctly identifying the minority class, the F_1-score can be a valuable metric.

F_β-score

To further adjust the balance between sensitivity and precision, the F_β-score is a modified version of the F_1-score. It is often used in evaluating imbalanced classification tasks to get more focus on either minimizing the false positives or the false negatives. The F_β-score is calculated as follows:

$$F_\beta\text{-score} = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Sensitivity}}{(\beta^2 \times Precision) + Sensitivity} \tag{27}$$

The parameter β controls the trade-off between precision and sensitivity. When β is larger than one, more emphasis is on the sensitivity, when β is smaller than one, more emphasis is on the precision. The F_β-score ranges from zero to one, with a value of one indicating perfect precision and sensitivity. In this thesis, the F_β-score with β equal to two is chosen to prioritize minimizing sensitivity, focusing on capturing all relevant components.

For the multi-class classification task, the F_β-score is weighted across the different classes. This includes calculating the F_β-score separately for each class and then averaging it based on the number of true instances for each class.

4.2.1.2 Hyperparameter Optimization Method

Hyperparameters mainly define the performance of a machine learning model and must be adapted to a specific task. Hyperparameter optimization aims to find a tuple of hyperparameters yielding to a minimal loss or another extremum of a metric. For this purpose, several optimization algorithms can be applied in machine learning. This thesis uses grid-search as an optimization approach, where each combination of hyperparameters specified in the search space is trained and evaluated. All models are trained with a grid of four selected hyperparameters and three values each, resulting in 81 combinations and, thus, 81 models per method and task.

4.2.1.3 Parameters

Seed: By setting a seed, the same results are generated when the model is trained with the same parameters and data, which ensures reproducibility.

Number of estimators/ epochs with early stopping: For the models in this thesis, the number of estimators/ epochs a model is trained is set to a high value of 10000, which gets limited due to incorporating early stopping. Early stopping is a regularization technique used to prevent overfitting by monitoring the model's performance on the validation set and by stopping the training when no improvement on the validation set is achieved after a defined number of estimators/ epochs, called patience. Hence, computational resources and training time are saved. The stopping criterion, patience, is set for all models to 25, which means that the training stops after the model's performance on the validation set does not increase for 25 iterations.

4.2.2 Ensemble Learning

This chapter describes the implementation of the ensemble learning methods XGBoost, LightGBM, and CatBoost for the binary and multi-class classification tasks, focusing on addressing class imbalance and optimizing hyperparameters for these methods. For the grid-search hyperparameter optimization, the hyperparameters tree depth, learning rate, L2 regularization, colsample by tree, and bagging temperature are selected, which enhance the generalization capabilities.

The **class weight** parameter is used in all implemented ensemble methods to address the class imbalance in the data set. It assigns different weights to classes based on their frequencies. For the binary task, the class weight is calculated using the formula shown in Equation 28, while for multi-class tasks, it is computed by Equation 29. By adjusting the class weights, the model can effectively reduce the impact of imbalanced class distributions and improve its ability to classify underrepresented classes.

$$class\_weight_{binary} = \frac{Number\ of\ records\ labeled\ as\ not\ relevant}{Number\ of\ records\ labeled\ as\ relevant} \tag{28}$$

$$class\_weight_{Einheitsname} = \frac{Number\ of\ records\ of\ the\ majority\ class}{Number\ of\ records\ of\ the\ considered\ class} \tag{29}$$

Grid-search hyperparameter optimization

Table 6 lists the hyperparameter combinations used for grid-search optimization, where each ensemble method is optimized with four hyperparameters and three values each. The hyperparameters and their effects are explained in more detail in the following.

Table 6: Values for hyperparameter tuning.

Hyperparameter	**XGBoost**	**LightGBM**	**CatBoost**
Tree depth	[3, 6, 9]	[6, 9, 12]	[3, 6, 9]
Learning rate	[0.1, 0.2, 0.3]	[0.05, 0.1, 0.2]	[0.1, 0.2, 0.3]
L2 regularization	[0.05, 0.1, 0.2]	[0.05, 0.1, 0.2]	[0.05, 0.1, 0.2]
Colsample by tree	[0.5, 0.7, 1.0]	[0.5, 0.7, 1.0]	x
Bagging temperature	x	x	[0.5, 1.0, 1.5]

The **tree depth** utilized in all ensemble models determines the maximum depth of the individual trees within the ensemble. Controlling the depth of the trees is an important parameter to manage the model's complexity and potential overfitting. While a deeper tree can capture more complex patterns in the training data but may lead to overfitting, a shorter tree may generalize better but might not capture all relevant patterns. Therefore, careful tuning this hyperparameter can lead to a balance between model complexity and generalization performance.

The **learning rate** is selected as a hyperparameter for optimization in all gradient boosting methods. Chapter 2.1 shows the importance of selecting an appropriate learning rate.

L2 regularization, also called weight decay, is a technique used to prevent overfitting. It adds a penalty term to the loss function, which penalizes large weights by adding the sum of the squared magnitude of all the weights to the cost or loss function (Marin et al., 2020).

Colsample by tree is a hyperparameter, which controls the fraction of features to be randomly sampled for each tree. Introducing randomness in feature selection for each tree can help to reduce overfitting and increase the variety of the trees, leading to better generalization performance.

Bagging temperature is a hyperparameter used in CatBoost and defines the sampling of the weights. The higher the temperature, the more aggressively the bagging is performed (Dorogush et al., 2018).

4.2.3 Deep Learning

This chapter addresses the implementation of the deep learning method with PyTorch Tabular, focusing on the used optimizer and the hyperparameter optimization. PyTorch Tabular is utilized solely for the binary classification task as it has currently no support for multi-class classification tasks. For the grid-search hyperparameter optimization, the hyperparameters layers, activation function, batch size, and dropout are discussed, enhancing the performance and generalization capabilities.

Optimizer

The Adaptive Moment Estimation (Adam) optimizer is used to adjust the weights and search for the optimum. This optimizer is common for classification tasks because it achieves high performance in a comparatively short time. It combines the advantages of the Adaptive Gradient (AdaGrad) algorithm and the Root Mean Square Propagation (RMSProp), which are both extensions of the stochastic gradient descent (SGD) algorithm (Ruder, 2016).

Learning rate

PyTorch Tabular has a learning rate finder implemented in its framework, which should enable the usage of an optimal learning rate for the tabular data models (Smith, 2017). However, as conceptual implementations in this thesis indicated a better performance with a fixed learning rate, 0.0001 is chosen as a learning rate for the deep learning models.

Grid-search hyperparameter optimization

The hyperparameter combinations for the grid-search hyperparameter optimization used to optimize the PyTorch Tabular models are shown in Table 7. Similar to the ensemble methods, hyperparameter optimization is performed with four hyperparameters and three values each. The hyperparameters and their effects are explained in more detail below.

Table 7: Values for hyperparameter tuning.

Hyperparameter	**PyTorch Tabular**
Layers	[16-8-4, 512-256-128, 2028-1014-512]
Activation function	[LeakyReLU, ReLU, Sigmoid]
Batch size	[16, 256, 1024]
Dropout	[0, 0.1, 0.3]

The **layers** refer to the architecture of the neural network, specifically the number of hidden layers and the number of neurons in each layer. The different configurations represent the arrangement of neurons in each layer and can significantly impact the model's capability to learn complex patterns from the input data.

Activation functions can introduce nonlinearities into the neural network to model complex relationships within the data. In this paper, different activation functions such as Leaky Rectified Linear Unit (LeakyReLU), Rectified Linear Unit (ReLU), and Sigmoid are considered.

The **batch size** defines the number of training samples, which are propagated through the neural network during a single iteration before adjusting the parameters. The choice of batch size can impact the model's convergence speed, memory requirements, and the stability of the training process.

Dropout is a regularization technique used to prevent overfitting. It works by randomly deactivating a fraction of neurons during each training iteration. In this study, no dropout and dropout values of 0.01 and 0.1 are considered, representing the proportion of neurons dropped out during training. For example, a dropout value of 0.1 means that during each training iteration, 10% of the neurons in the network are randomly deactivated by setting their outputs to zero. Hence, the network learns from more robust features and distributes the decision competence across many neurons.

5 Evaluation

This chapter evaluates the performance of the models developed in Chapter 4 by covering the steps of Figure 21, beginning with the model selections based on the described hyperparameter optimizations. As a basis, this chapter first describes the general concepts for the evaluation.

5.1 General Concepts

This chapter introduces the metrics and the resampling technique used for all models of both tasks.

5.1.1 Metrics

In addition to the sensitivity, the precision, and the F_2-score presented in Chapter 4.2.1.1, the confusion matrix is utilized for evaluating the performance of the models. It is a valuable tool for understanding the model's performance in differentiating between classes and identifying areas of improvement. In this chapter, the confusion matrix is used to analyze the performance of the models on the test set.

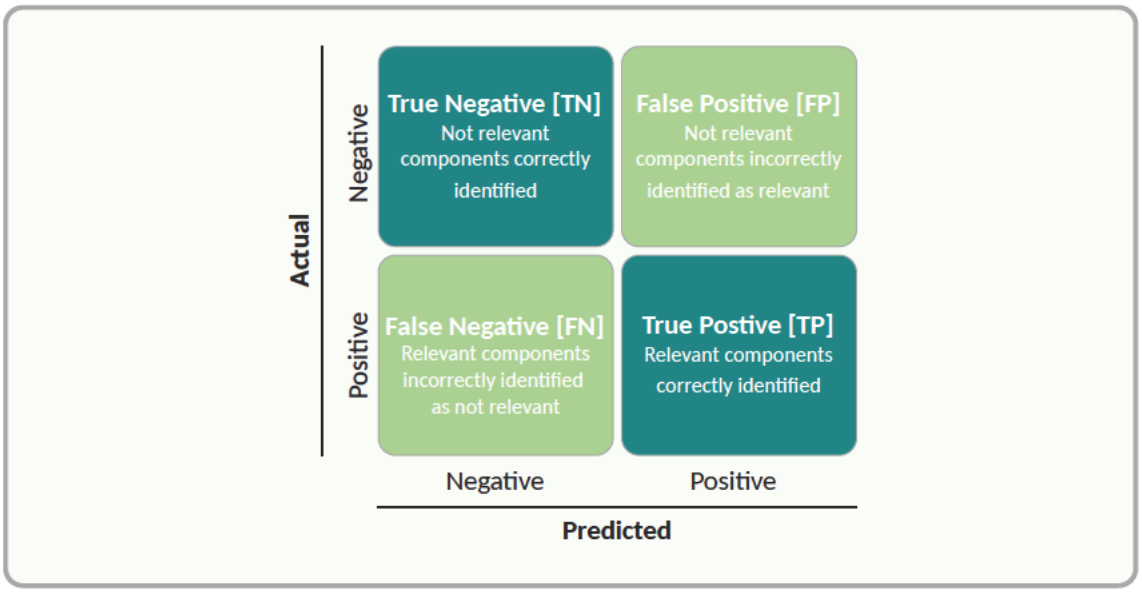


Figure 22: Confusion matrix with description for the binary classification task.

The confusion matrix provides a tabular representation of the model's predictions compared to the actual labels. Figure 22 shows the meanings of the values in the confusion matrices for the binary classification task. In the confusion matrices for the multi-class classification task, the diagonal elements correspond to the true positive predictions for each class, and off-diagonal elements represent misclassifications.

5.1.2 K-fold Cross-Validation

K-fold cross-validation is a method that enhances the robustness for estimating the performance of a machine learning model and is particularly important for smaller data sets. The basic idea behind k-fold cross-validation is to split the data set into k subsets, also called "folds", which have a similar size. The model is k times trained and evaluated, where always one of the k folds is used as the validation set, and the other k-1 folds as the training set. With k-fold cross-validation, a more reliable estimate of the model's performance can be obtained compared to a single split, especially for small data sets. This thesis employs the k-fold cross-validation technique with $k = 4$ for all methods. The same value for k allows a fair comparison of the performance of various models, as they are all evaluated using the same cross-validation setup.

5.2 Model Evaluation

As explained in Chapter 4 and Figure 21, the models of each method are trained using a grid-search hyperparameter optimization, which results in 81 models per method and task. Based on the validation F_2-score, the top four models (top 5%) from hyperparameter optimization for each task and method are selected. These models are further evaluated using the 4-fold cross-validation technique. The best model of each method and task is chosen based on the average validation F_2-score from 4-fold cross-validation, which is evaluated in more detail. In Chapter 6, the final binary and multi-class models are determined by comparing the best models across the investigated methods.

5.2.1 Ensemble Learning

This chapter details the evaluation of the three ensemble learning methods XGBoost, LightGBM, and CatBoost, starting with the binary classification task and followed by the multi-class classification task.

5.2.1.1 XGBoost

Binary classification task - Component relevance

For the XGBoost models of the binary classification task, Table 8 shows the results of the four models with the highest F_2-scores on the validation set. The corresponding hyperparameters and the training complexities for these models are listed in Table A2.

Table 8: Results of the top four XGBoost models for the binary classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
23	**0.99752**	**0.00086**	**0.96607**	**0.01735**
11	**0.99752**	0.00093	0.96377	0.01802
26	**0.99752**	0.00089	0.96377	0.01810
51	**0.99752**	0.00205	0.96269	0.01843

These four models are further evaluated using 4-fold cross-validation, which leads to the results in Table 9. When comparing these results with the results after hyperparameter tuning, the ranking of the models based on their validation F_2-score has changed. Here, Model 11 is selected as the best XGBoost model after cross-validation, as it has the highest average validation F_2-score of 95.324%. For the XGBoost Model 11, the average validation F_2-score from k-fold cross-validation of 95.324% is lower than the validation F_2-score of 96.377% using one fold.

Table 9: Results of the top four XGBoost models for the binary classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Average F_2-score	Average loss	Average F_2-score	Average loss
11	0.99660	0.00238	**0.95324**	**0.01526**
51	**0.99864**	0.00145	0.94690	0.01707
23	**0.99864**	0.00156	0.94600	0.01630
26	**0.99864**	**0.00107**	0.94470	0.01690

To assess the generalizability of XGBoost Model 11, it is further analyzed on the test set, also by calculating the sensitivity and precision, which are especially interesting metrics for the binary classification task with imbalanced data. On the test set, the XGBoost Model 11 achieves a F_2-score of 96.237%, which is slightly lower than the validation F_2-score and higher than the average F_2-score from 4-fold cross-validation. Detailing the metric further, the model reaches on the test set a sensitivity of 95.652% and a precision of 98.507%. The resulting confusion matrix in Figure A3 shows that six out of 138 relevant components are incorrectly classified as not relevant, while two out of 2784 not relevant components are incorrectly classified as relevant.

Multi-class classification task - Uniformly coded designations

For the multi-class classification task, the results of the four models with the highest validation F_2-score are given in Table 10, and the corresponding hyperparameters

and training complexities are listed in Table A3. To further evaluate the top four models after hyperparameter optimization, 4-fold cross-validation is used, which leads to the results listed in Table 11. Contrary to after hyperparameter tuning, Model 46 is here identified as the best XGBoost Model for the multi-class classification task with an average F_2-score of 98.169% on the validation set. For the XGBoost Model 46, the average validation F_2-score from k-fold cross-validation of 98.169% is slightly lower than the validation F_2-score of 98.532% using one fold. The best XGBoost Model 46 achieves an F_2-score of 99.256% on the test set, which is higher than the validation F_2-score and the average F_2-score from 4-fold cross-validation. The confusion matrix shows only one incorrectly classified component, which is a component of class "SITZBEZUG HINTEN" that is classified as "VERKLEIDUNG STOSSFAENGER".

Table 10: Results of the top four XGBoost models for the multi-class classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
40	**1.0000**	0.02435	**0.98532**	**0.10879**
43	**1.0000**	0.02409	**0.98532**	0.11990
44	**1.0000**	0.02462	**0.98532**	0.12295
46	**1.0000**	**0.02229**	**0.98532**	0.10254

Table 11: Results of the top four XGBoost models for the multi-class classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Avgerage F_2-score	Avgerage loss	Avgerage F_2-score	Avgerage loss
46	**1.0000**	0.02665	**0.98169**	**0.11452**
44	**1.0000**	0.02622	0.97795	0.12543
43	**1.0000**	**0.02580**	0.97787	0.12308
40	**1.0000**	0.02669	0.97780	0.11913

5.2.1.2 LightGBM

Binary classification task - Component relevance

For the LightGBM models of the binary classification task, Table 12 shows the results of the four models with the highest F_2-scores on the validation set. The corresponding hyperparameters and the training complexities for these models are listed in Table A4.

Table 12: Results of the top four LightGBM models for the binary classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
0	0.99721	0.00574	**0.95514**	0.02099
13	0.98561	0.01743	0.95442	0.03068
21	0.99413	0.00827	0.95409	0.02489
9	**0.99814**	**0.00442**	0.95376	**0.02034**

These four models are further evaluated using 4-fold cross-validation, which leads to the results in Table 13. Also here, the ranking of the models based on their F_2-score has changed. The highest average validation F_2-score of 95.465% is achieved by Model 13, which is therefore selected as the best LightGBM model after cross-validation. For the LightGBM Model 13, the average validation F_2-score from k-fold cross-validation of 95.465% is comparable to the validation F_2-score of 95.442% using one fold.

Table 13: Results of the top four LightGBM models for the binary classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Average F_2-score	Average loss	Average F_2-score	Average loss
13	**0.99534**	0.00808	**0.95465**	**0.02003**
0	0.99168	0.01150	0.95284	0.02336
9	0.99501	**0.00737**	0.95045	0.02047
21	0.98698	0.01649	0.94584	0.02827

On the test set, the LightGBM Model 13 achieves a higher F_2-score of 97.142% compared to the validation F_2-score and the average F_2-score from 4-fold cross-validation. Detailing the metric further, the model reaches on the test set a sensitivity of 98.551% and a precision of 91.892%. The resulting confusion matrix in Figure A5 shows that two out of 138 relevant components are incorrectly classified as not relevant, while 12 out of 2784 not relevant components are incorrectly classified as relevant.

Multi-class classification task - Uniformly coded designations

For the multi-class classification task, the results of the four models with the highest validation F_2-score are given in Table 14, and the corresponding hyperparameters and training complexities are listed in Table A5. 4-fold cross-validation on these

top four models leads to the results listed in Table 15. The highest average validation F_2-score of 98.155% is achieved with Model 17, which is selected as the best LightGBM model for the multi-class classification task. For the LightGBM Model 17, the average validation F_2-score from k-fold cross-validation of 98.155% is higher than the validation F_2-score of 97.797% using one fold. The best LightGBM Model 17 achieves an F_2-score of 99.256% on the test set, which is higher than the validation F_2-score and the average F_2-score from 4-fold cross-validation. The confusion matrix shows that the incorrectly classified component is again a component of class "SITZBEZUG HINTEN" that is classified as "VERKLEIDUNG STOSSFAENGER".

Table 14: Results of the top four LightGBM models for the multi-class classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
16	**1.00000**	**0.00000**	**0.97808**	0.08344
7	**1.00000**	0.00050	0.97797	0.08631
17	**1.00000**	**0.00000**	0.97797	**0.07447**
26	**1.00000**	**0.00000**	0.97797	0.07957

Table 15: Results of the top four LightGBM models for the multi-class classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Avgerage F_2-score	Avgerage loss	Avgerage F_2-score	Avgerage loss
17	**1.00000**	0.00038	**0.98155**	0.07849
16	**1.00000**	**0.00016**	0.98152	**0.07406**
7	**1.00000**	**0.00016**	0.98152	**0.07406**
26	**1.00000**	0.00066	0.98020	0.07584

5.2.1.3 CatBoost

Binary classification task - Component relevance

For the CatBoost models of the binary classification task, Table 16 shows the results of the four models with the highest F_2-scores on the validation set. The corresponding hyperparameters and the training complexities for these models are listed in Table A6.

Table 16: Results of the top four CatBoost models for the binary classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
78	**0.99660**	**0.01337**	**0.97122**	0.06059
79	**0.99660**	**0.01337**	**0.97122**	0.06059
80	**0.99660**	**0.01337**	**0.97122**	0.06059
24	0.99567	0.01958	0.96983	**0.05719**

These four models are further evaluated using 4-fold cross-validation, which leads to the results in Table 17. Also here, the ranking of the models based on their F_2-score has changed. The highest average validation F_2-scores of 95.000% is achieved by Model 24, which is therefore selected as the best CatBoost model after cross-validation. For the CatBoost Model 24, the average validation F_2-score from k-fold cross-validation of 95.000% is lower than the validation F_2-score of 96.983% using one fold.

Table 17: Results of the top four CatBoost models for the binary classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Average F_2-score	Average loss	Average F_2-score	Average loss
24	0.99560	0.01933	**0.95000**	**0.08678**
78	**0.99923**	**0.00753**	0.94046	0.10899
79	**0.99923**	**0.00753**	0.94046	0.10899
80	**0.99923**	**0.00753**	0.94046	0.10899

On the test set, the CatBoost Model 24 achieves a F_2-score of 96.705%, which is lower than the validation F_2-score, but higher than the average F_2-score from 4-fold cross-validation. Detailing the metric further, the model reaches on the test set a sensitivity of 97.826% and a precision of 92.466%. The resulting confusion matrix in Figure A7 shows that three out of 138 relevant components are incorrectly classified as not relevant, while 11 out of 2784 not relevant components are incorrectly classified as relevant.

Multi-class classification task - Uniformly coded designations

For the multi-class classification task, the results of the four models with the highest validation F_2-score are given in Table 18 and the corresponding hyperparameters, and training complexities are listed in Table A7. 4-fold cross-validation on these top four models leads to the results listed in Table 19. The highest average F_2-score of

98.951% is achieved with Model 76, which is selected as the best CatBoost model for the multi-class classification task. For the CatBoost Model 76, the average validation F_2-score from k-fold cross-validation of 98.951% is lower than the validation F_2-score of 99.254% using one fold.

The best CatBoost Model 76 achieves an F_2-score of 100% on the test set, which is higher than the validation F_2-score and the average F_2-score from 4-fold cross-validation. Hence, the confusion matrix shows no incorrectly classified component.

Table 18: Results of the top four CatBoost models for the multi-class classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
73	**1.00000**	0.00122	**0.99254**	0.06132
79	**1.00000**	**0.00072**	**0.99254**	**0.04628**
76	**1.00000**	0.00099	**0.99254**	0.05006
63	**1.00000**	0.00310	0.99252	0.04982

Table 19: Results of the top four CatBoost models for the multi-class classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Avgerage F_2-score	Avgerage loss	Avgerage F_2-score	Avgerage loss
76	**1.00000**	**0.00234**	**0.98951**	**0.04968**
79	**1.00000**	0.00256	0.98821	0.05260
63	**1.00000**	0.00467	0.98821	0.05974
73	**1.00000**	0.00342	0.98691	0.06101

5.2.2 Deep Learning

Binary classification task - Component relevance

For the PyTorch Tabular models of the binary classification task, Table 20 shows the results of the four models with the highest F_2-scores on the validation set. As more models share the same best F_2-scores, those four models are selected, which have additional the lowest four losses on the validation set. The corresponding hyperparameters and the training complexities for these models are listed in Table A8.

Table 20: Results of the top four PyTorch Tabular models for the binary classification task after grid-search hyperparamter optimization.

	Train		**Validation**	
Model	F_2-score	Loss	F_2-score	Loss
13	0.99932	**0.00027**	**0.99855**	**0.00077**
26	**0.99966**	0.00516	**0.99855**	0.00103
7	0.99932	0.00042	**0.99855**	0.00111
8	0.99897	0.00068	**0.99855**	0.00119

These four models are further evaluated using 4-fold cross-validation, which leads to the results in Table 21. Also here, the ranking of the models based on their F_2-score has changed. The highest average validation F_2-scores of 89.933% is achieved by Model 26, which is therefore selected as the best PyTorch Tabular model after cross-validation. For the PyTorch Tabular Model 26, the average validation F_2-score from k-fold cross-validation of 89.933% is considerably lower than the validation F_2-score of 99.855% using one fold.

Table 21: Results of the top four PyTorch Tabular models for the binary classification task after 4-fold cross-validation.

	Train		**Validation**	
Model	Average F_2-score	Average loss	Average F_2-score	Average loss
26	**0.99829**	0.00623	**0.89933**	**0.03145**
8	0.99840	0.00226	0.88154	0.03198
7	0.99898	**0.00160**	0.87669	0.03396
13	0.98489	0.00251	0.83800	0.06220

On the test set, the PyTorch Tabular Model 26 achieves an F_2-score of 99.855%, which is comparable to the validation F_2-score and considerably higher than the average F_2-score from 4-fold cross-validation. Detailing the metric further, the model reaches on the test set a sensitivity of 100% and a precision of 99.281%. The resulting confusion matrix in Figure A10 shows that zero out of 138 relevant components are incorrectly classified as not relevant, while one out of 2784 not relevant components are incorrectly classified as relevant.

6 Discussion

While Chapter 5 presented the evaluation and selection of the best model for each method and task individually, this chapter compares the best models for a task across the different methods. It discusses the individual advantages and determines a final model in the context of this thesis for each task, which is then further analyzed and re-trained on the combination of the original training and test set for the inference model to use all available data. The chapter starts with the models for the binary classification tasks, which is followed by the models for the multi-class classification task.

6.1 Binary Classification Task

In this chapter, the best models for the binary classification task identified in Chapter 5.2 are compared with regard to the performance and training complexity. Subsequently, the model, which is determined as the final model for the binary classification task, is further discussed in terms of its explainability and performance.

6.1.1 Comparison of the Models

Table 22 shows the validation results and the training complexity of the three best models of the gradient boosting methods XGBoost, LightGBM, and CatBoost. Across the different models, the LightGBM Model 13 achieves the highest average F_2-score of 95.465% with k-fold cross-validation, slightly outperforming both the XGBoost Model 11 by 0.141% and the CatBoost Model 24 by 0.465%. Considering the average validation loss from k-fold cross-validation, the XGBoost Model 11 achieves the lowest loss of 0.01526, indicating a better performance in minimizing the error.

In terms of training complexity, the LightGBM Model 13 is characterized by requiring the fewest number of estimators (55) and the shortest training time (4 seconds). This shows that LightGBM is highly efficient, making it the preferable choice when training time and computational resources are limited.

In conclusion, the choice of the best gradient boosting model among XGBoost, LightGBM, and CatBoost depends on the specific requirements of the task. As the primary goal in this thesis is to maximize the F_2-score, the LightGBM Model 13 is selected as the best gradient boosting model for the binary classification task. The findings are in line with the benchmarking for gradient boosting frameworks performed by Florek and Zagdański (2023), which demonstrate across various data sets that all three frameworks achieve promising results but also give a slight tendency towards LightGBM due to its high performance and low runtime. In the following, the best gradient boosting model (LightGBM Model 13) is compared to the best deep learning model for the binary classification task.

Table 22: Best gradient boosting models of each method for the binary classification task.

	Validation		Training Complexity	
Model	Average F_2-score	Average loss	Number of estimators	Training time in seconds
XGBoost Model 11	0.95324	**0.01526**	232	292
LightGBM Model 13	**0.95465**	0.02003	**55**	**4**
CatBoost Model 24	0.95000	0.08678	111	106

Table 23: Best deep learning model for the binary classification task.

	Validation		Training Complexity	
Model	Average F_2-score	Average loss	Number of epochs	Training time in seconds
PyTorch Tabular Model 26	**0.89933**	**0.03145**	**199**	**895**

The results of the best deep learning model are given in Table 23, where the PyTorch Tabular Model 26 achieves the highest average F_2-score of 89.933% with k-fold cross-validation.

Comparing the PyTorch Tabular Model 26 with the LightGBM Model 13, the deep learning model achieves a 5.532% lower performance on the averaged F_2-score with cross-validation than the LightGBM model. In addition, the training complexity is substantially higher. Therefore, the **LightGBM Model 13** is chosen as the final model for the binary classification task and is further analyzed in terms of performance and explainability in the following chapter.

6.1.2 Discussion of the Best Model

To provide insights into how the selected LightGBM Model 13 makes its predictions, Figure A6 shows a beeswarm plot of the 30 most important features for the predictions on the validation set identified using the SHAP framework. The color coding shows the magnitude of the feature values, where a dark color represents a high feature value and a light color a low feature value. The x-axis shows the effect on the output, where features with positive values have an effect on the positive class (relevant) and negative values on the negative class (not relevant). The five most influential features are “center_y”, “center_z”, “volume”, “length”, and “value”. When considering “center_z”, a direct correlation becomes evident that

high feature values have a substantial influence on the positive class, which indicates that vehicle components positioned higher in the vehicle are more likely to be relevant. For "volume", high feature values also correlate with a substantial impact on the positive class, indicating that vehicle components with a higher volume are more likely to be relevant. In addition, an examination of certain features representing the n-grams of the vehicle component designation shows that a feature value of one for the feature "KANT" is a strong indicator for a relevant vehicle component, probably due to the "KANTENSCHUTZ" components.

The training and validation losses of the LightGBM Model 13 shown in Figure A4 have a consistent decreasing trend, and no overfitting or fluctuation is observable. However, there is a small gap between the training loss and validation loss observable, which could indicate a slight imbalance in splitting the data sets.

The LightGBM Model 13 identified as the best model for the binary classification task achieves a F_2-score of 95.442% on the validation set. On the test set, it achieves a higher F_2-score of 97.142% and, in more detail, a sensitivity of 98.551% and a precision of 91.892%. The resulting confusion matrix in Figure A5 shows that two out of 138 relevant components are incorrectly classified as not relevant, while 12 out of 2784 not relevant components are incorrectly classified as relevant. By classifying not relevant components as relevant, the runtime for loading the considered relevant components in CATIA is increased. However, this problem is minor if the number is small. Classifying relevant components as not relevant requires manual adaptions in the aftermath, which was the motivation for choosing the F_2-score as the important metric. As one vehicle contains on average 28 relevant components and the split is stratified distributed, the test set with 138 relevant components and 2784 not relevant components can be considered as a data set from five vehicle models. Then, this result can, for example, be interpreted that for three out of five vehicles, no manual adaption is required as all relevant components have been correctly classified, and for the other two vehicles, one additional component must be added manually that is not identified correctly as a relevant component.

The differences between the F_2-scores on one validation set and the average F_2-scores from 4-fold cross-validation, the deviations between the results on the validation and test sets, and the gap in the loss plot between the training and validation set indicate that the results are dependent on the performed split and a larger data set could further reduce the variances and improve the performance of the models.

Hence, after identifying and analyzing the best model for the binary classification task, this model is re-trained to use the entire available data for the inference model. The original validation set is kept, but a new train set is generated that combines the original train and test set. Here, the model achieves a validation F_2-score of 95.514%, a sensitivity of 95.652%, and a precision of 94.624%.

6.2 Multi-class Classification Task

In this chapter, the best models for the multi-class classification task identified in Chapter 5.2 are compared with regard to the performance and training complexity. Subsequently, the model, which is determined as the final model for the multi-class classification task, is further discussed in terms of its explainability and performance.

6.2.1 Comparison of the Models

As mentioned in Chapter 4.2.3, PyTorch Tabular currently has no support for multi-class classification tasks. Therefore, solely the gradient boosting methods are compared for this task to find the best model of the three methods XGBoost, LightGBM, and CatBoost. Table 24 shows the results of the best models of each gradient boosting method. The highest average validation F_2-score of 98.951% with 4-fold cross-validation is achieved by CatBoost Model 76, outperforming the XGBoost Model 46 by 0.782% and the LightGBM Model 17 by 0.796%. Considering the average loss from k-fold cross-validation, the CatBoost Model 76 achieves the lowest loss of 0.04968, indicating the best performance in minimizing the error.

In terms of training complexity, the XGBoost Model 46 is characterized by utilizing the fewest number of estimators (84), and the LightGBM Model 17 by requiring the shortest training time (5 seconds). This shows again that LightGBM is highly efficient, making it the preferable choice when training time is essential.

In conclusion, the choice of the best gradient boosting model among XGBoost, LightGBM, and CatBoost depends on the specific requirements of the task. As the primary goal in this thesis is to maximize the F_2-score, the **CatBoost Model 76** is selected as the best model for the multi-class classification task.

Table 24: Best gradient boosting models of each method for the multi-class classification task.

	Validation		**Training Complexity**	
Model	Average F_2-score	Average loss	Number of estimators	Training time in seconds
XGBoost Model 46	0.98169	0.11452	**84**	46
LightGBM Model 17	0.98155	0.07849	308	**5**
CatBoost Model 76	**0.98951**	**0.04968**	389	1024

6.2.2 Discussion of the Best Model

Also, for the multi-class classification task, the SHAP beeswarm plot shown in Figure A9 provides insights into how the model makes its predictions on the validation set.

Here, the five most important features are “center_z”, “center_y”, “length”, “width”, and “center_x”. Even though the order of the most important features is different than for the binary classification task, again, the features describing the bounding box and the weight of the component are ranked as the most important features. This ranking is firstly surprising for the generation of uniformly coded designations, where one could assume that the n-grams of the “Benennung (dt)” are particularly important. However, the reason for the overall higher relevance of the bounding box features is probably that they are available for all components, while most n-grams are only tailored to specific components. The n-grams among the 30 most important features for the multi-class classification task differ largely from the n-grams that are among the 30 most important features for the binary classification task. This shows that other char combinations are meaningful for deciding whether a component is relevant or not and for generating uniform designations.

The training and validation loss of the CatBoost Model 76 shown in Figure A8 does not indicate general overfitting but also has a gap between the training and validation loss. Especially as the data set consists only of the relevant components for the multi-class classification task, the data set is small, and the gap in the plot could indicate that the performed split has an imbalance.

The CatBoost model 76 is identified as the best model for the multi-class classification task with an F_2-score of 99.254% on the validation set. On the test set, it achieves an even higher F_2-score of 100.00%. Hence, the confusion matrix shows no incorrectly classified component. As explained before, the test set can be considered as a data set from five vehicle models. Then, the result can be interpreted such that for all five out of five vehicles, no classification errors are present. Therefore, no manual adjustments in the uniformly coded designations are required.

Like in the binary classification task, the deviations between the results on the different sets and the gap in the loss plot between the training and validation set indicate that the results are dependent on the performed split, and a larger data set could further reduce the variances and improve the performance of the models.

Hence, after identifying and analyzing the best model for the multi-class classification task, this model is re-trained to use the entire available data for the inference model. The original validation set is kept, but a new train set is generated that combines the original train and test set.

7 Deployment

This chapter focuses on the deployment process of the AI system. The architecture shown in Figure 23 illustrates the modules involved in the deployment and includes an optional way (dotted lines) for the engineer to access the model via a website.

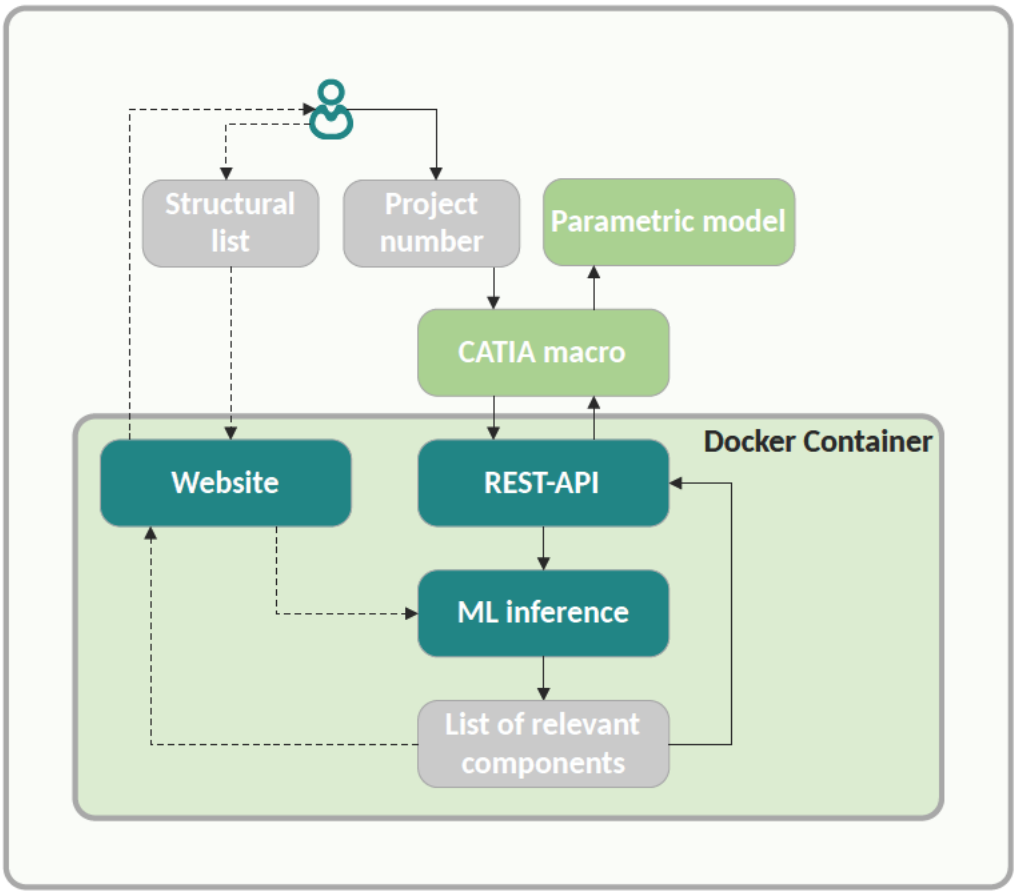


Figure 23: Deployment architecture.

The module for the interface to the engineer in the measurement tool is the CATIA macro. Here, the engineer can enter the project number of the vehicle model for which the measurements should be performed. The CATIA macro then loads the structural list from the database and sends it to the REST-API with an HTTP request. Afterward, the REST-API uses the first machine learning model to identify the relevant car components and then the second machine learning model to get a uniformly coded designation for each relevant component. Afterward, the list of relevant components is sent back as a JSON response of the REST-API to the CATIA macro, which loads the components into CATIA, creates cuts and planes, and loads them into the CATIA parametric model.

An additional option for the engineer to utilize the machine learning models is by uploading the structural list on a website. After uploading, the engineer gets the list of relevant components directly displayed. This option is especially implemented for

the development phase, as here the engineer can review the results to give feedback on the models' performance.

To get a more detailed understanding of the deployment architecture, the main technologies and frameworks are explained in the following. The CATIA macro and the parametric model are not developed in this thesis and, therefore, are not explained in detail.

The **REST-API** handles HTTP requests and responses. It is developed using FastAPI, a Python web framework for building APIs. It combines important features of other popular frameworks, such as Flask and Django, while leveraging Python features (Ramírez, 2018). One of the fundamental principles of FastAPI is its support for asynchronous programming, which allows multiple requests to be handled parallelly, making it well-suited for high-performance applications (Ramírez, 2018).

The **website** is developed using streamlit, which is an open-source Python library. It simplifies and accelerates the building of interactive web applications for machine learning and data science. The API includes functions for creating widgets, displaying data, and generating visualizations.

Docker is an open-source software that is commonly used to deploy and run applications. It provides a platform for separating applications from the underlying infrastructure, which allows developers to quickly build, test, and deploy code (Rad et al., 2017). The key objects of docker are images and containers, which are explained in the following:

At the core of docker is the concept of images, which is a self-contained and executable package that includes all the necessary dependencies, libraries, and configurations required to run a particular application. An image is built using a docker file, which contains a set of instructions for creating the image. These instructions specify the base image, the dependencies to install, the files to copy, and the commands to execute during image creation.

A docker container is an isolated and executable instance of an image. It decouples the application and all of its dependencies from the underlying infrastructure. Hence, they are designed to be portable and platform-independent, which means that containers can run on any system where docker is installed.

In the AI system, the docker container virtualizes the REST-API, the website, and the ML models.

AWS is a large cloud computing platform that offers many services to build, deploy, and manage applications and infrastructure in a flexible and scalable way. In the AI system, AWS hosts the docker container, including the website and the API.

8 Limitations and Future Work

This chapter focuses on the limitations of this thesis and proposes potential future work.

Limitations

One limitation of the models developed in this thesis is their proneness to **data shift**. Data shift refers to the phenomenon where the distributions or characteristics of the data change over time. In the context of the tasks in this thesis, data shift occurs when the relevant car parts in the automotive industry change or evolve. This can happen due to various factors such as technological advancements, design improvements, or new regulations. Ignoring data shifts can lead to outdated or ineffective models for making accurate predictions.

Another limitation of the developed models is the **amount of data** used to train and evaluate the machine learning models. The data set for this thesis is generated from a limited number of 33 vehicles, and the reported results indicate a sampling bias.

Finally, the techniques used in this thesis are limited by the **capabilities of the frameworks** employed. For instance, PyTorch Tabular is solely available for binary classification, and XGBoost does not allow using a combination of two metrics if one is a custom metric like the F_β-Score.

Future Work

Besides the improvements to the frameworks, one direction of future work is how the models are deployed. The current deployment of the machine learning models (see Chapter 7) is solely temporary. In the future, the models will be hosted on an **AI platform** where the models and the code for data preparation will be integrated into an existing infrastructure, including model evaluation, data labeling, data analysis, and model deployment. In the AI platform, data shift detection techniques should be implemented to ensure that changes in the probability distribution of input variables and targets can be detected.

Future work should also prepare the **data from more vehicles** to increase the available data set, which could further improve the performance and the generalization of the models.

Another potential future work direction is the adaptation of the developed models for **other use cases**. For example, the AI model could be used by another department to assign specific components to a generic component. This use case is similar to the tasks solved in this thesis, as the same data can be used, but only the class labels need to be changed.

9 Conclusion

Geometric measurements are frequently performed along a virtual vehicle development chain to monitor and confirm the fulfillment of dimensional requirements for purposes like safety and comfort. The current manual measuring process lacks in comparability and quality aspects and involves high time and cost expenditure due to the repetition across different departments, engineers, and vehicle projects.

Thereby motivated, this thesis develops and implements automated solutions for the component identification within the geometric measurement process. The first goal is to classify the components of a vehicle as relevant and not relevant for the geometric measurements (binary classification task). The second goal is to generate uniformly coded designations for the relevant car components (multi-class classification task).

In addition to the data labeling for both tasks, extensive data preparation is required to effectively utilize the data for the machine learning models. This included feature engineering, data preprocessing, and data augmentation steps. Due to the high imbalance between relevant and not relevant components, the components are pre-filtered in a rule-based approach. LightGBM, XGBoost, and CatBoost as gradient boosting methods and FNNs as deep learning methods are investigated for the binary classification task, which are compared regarding performance and training complexity. Due to the restriction of Pytorch Tabular, for the multi-class classification task, solely LightGBM, XGBoost, and CatBoost are compared.

To find the best method and model for both tasks, grid-search hyperparameter tuning, and an iterative model selection process is implemented, which incorporates different validation methods. For the binary classification task, the highest average F_2-score of 95.465% using k-fold cross-validation is achieved by the LightGBM Model 13. On the test set, this model reaches an F_2-score of 97.142%, a sensitivity of 98.551%, and a precision of 91.892%. It classifies two out of 138 relevant components incorrectly as not relevant and 12 out of 2784 not relevant components incorrectly as relevant. For the multi-class classification task, the highest average F_2-score of 98.951% using k-fold cross-validation is achieved by the CatBoost Model 76. On the test set, this model reaches an F_2-score of 100% and, hence, assigns to all components the target uniformly coded designation. The explainability analysis revealed that for both tasks especially the features corresponding to the bounding box of the component and its weight are important criteria.

This thesis shows that machine learning methods can effectively be integrated into the geometric measurement process to increase comparability and reduce time and cost expenditure by identifying relevant components and assigning uniformly coded designations. For both tasks, gradient boosting models have been proven to be performant choices and require solely limited manual corrections. To make the results of the thesis directly and easily usable, the software solution is integrated into a CATIA pipeline via an API. Future work should focus on integrating the models in the AI platform and on training and evaluating the models on more data to reduce the sampling bias.

A Appendix

			Bounding Box (Bauteil)						Konst. Gewicht		
Ebene	Sachnummer	Benennung (dt)	X-Min	X-Max	Y-Min	Y-Max	Z-Min	Z-Max	Wert	Einheit	Gewichtsart
1	1234567	VIRTUELLES GESAMTFAHRZEUG									
2	P0M1AA1	VIRTUELLE PRODUKTIONSBAUGRUPPE									
3	P1MAAA5	VPBG ECE									
4	P1MAAA6	EP									
5	P1MBBB0	MODUL BODENGRUPPE									
6	P1MBBB1	VORDERBAU, STIRNWAND									
7	P1MBBB2	KOGR XX VORDERBAU ROHKAROSSERI									
8	7321234	BUCHSE MOTORTRAEGER	340.0	302.5	-512.2	-380.4	-123.5	-40.0	1000	g	R
8	7321234	BUCHSE MOTORTRAEGER	340.0	302.5	-512.2	-380.4	-123.5	-40.0	1000	g	R
8	7325678	CRASHVERSTAERKUNG MOTORTRAEGER LINKS	340.0	302.5	-512.2	-380.4	-123.5	-40.0	1000	g	R
8	7325679	CRASHVERSTAERKUNG MOTORTRAEGER RECHTS	340.0	302.5	-512.2	-380.4	-123.5	-40.0	1000	g	R

Table A1: Modified snapshot of a data set (structural list).

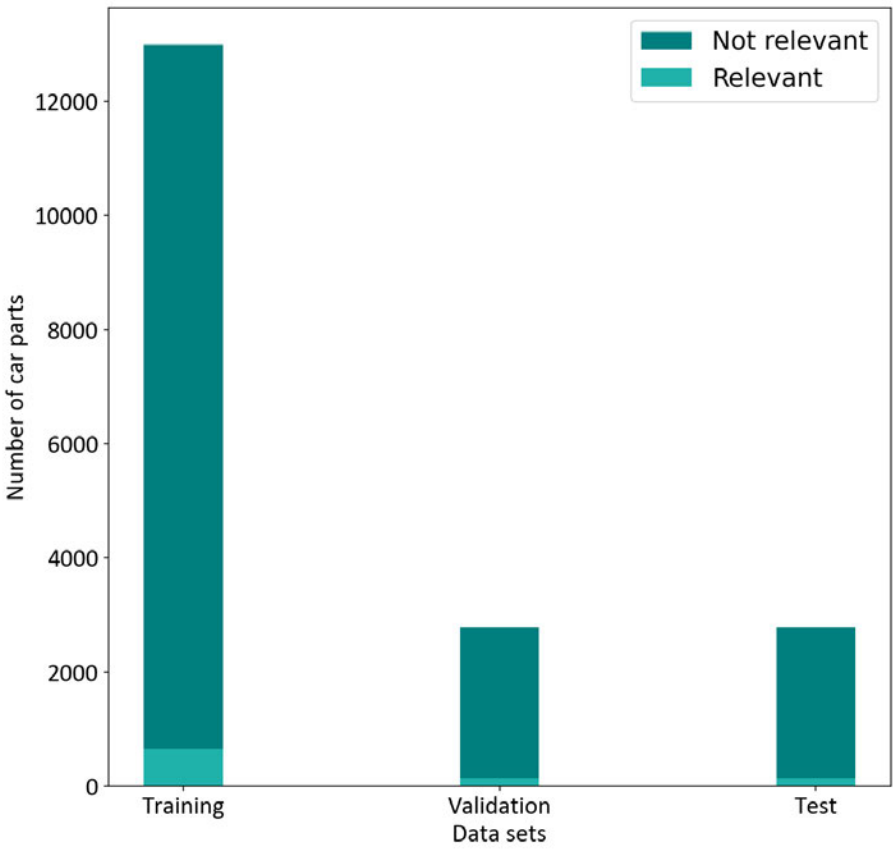


Figure A1: Distribution of the training, validation, and test sets of the binary classification task.

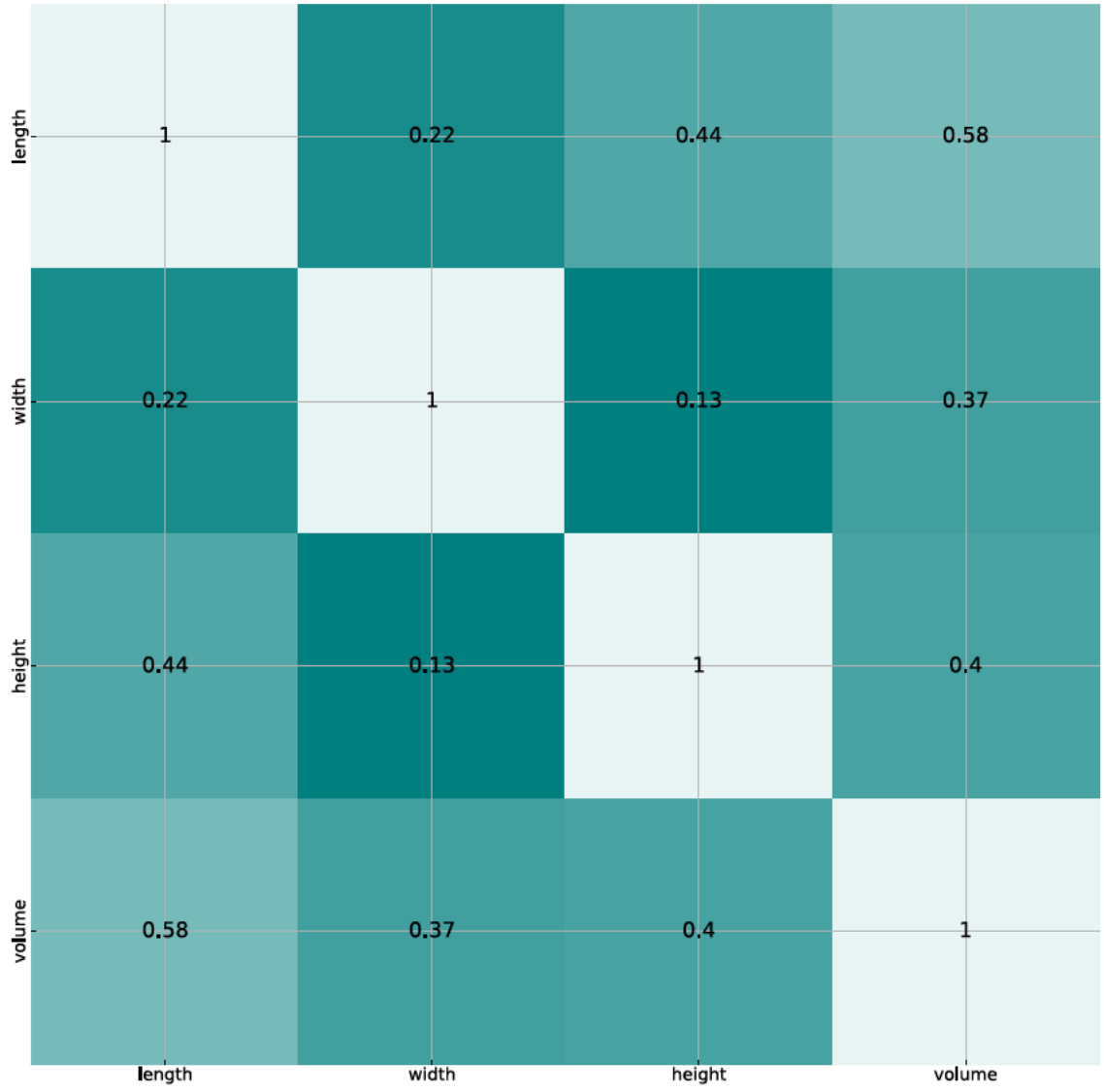


Figure A2: Correlation matrix of selected bounding box features.

Table A2: Hyperparameters of the top four XGBoost models for the binary classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				Training complexity	
Model	Tree depth	Learning rate	L2 regularization	Colsample by tree	Number of estimators	Training time in seconds
23	9	0.3	0.7	0.05	130	299
11	6	0.3	0.5	0.05	232	292
26	9	0.3	1.0	0.05	120	386
51	9	0.2	1.0	0.2	91	315

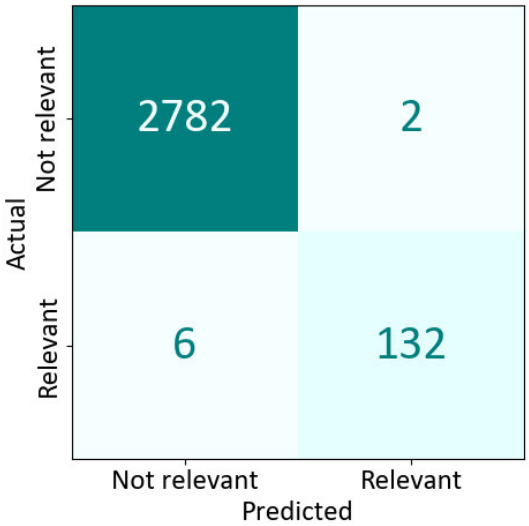


Figure A3: Confusion matrix for the test set with the XGBoost Model 11 for the binary classification task.

Table A3: Hyperparameters of the top four XGBoost models for the multi-class classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				**Training complexity**	
Model	Tree depth	Learning rate	L2 regularization	Colsample by tree	Number of estimators	Training time in seconds
40	6	0.2	0.1	0.7	31	36
43	6	0.2	0.1	1.0	32	51
44	6	0.2	0.05	1.0	29	45
46	9	0.2	0.1	0.5	84	46

Table A4: Hyperparameters of the top four LightGBM models for the binary classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				Training complexity	
Model	Tree depth	Learning rate	L2 regularization	Colsample by tree	Number of estimators	Training time in seconds
0	12	0.2	0.2	1.0	67	5
13	9	0.2	0.1	0.7	55	4
21	6	0.2	0.2	0.7	153	7
9	9	0.2	0.2	1.0	106	6

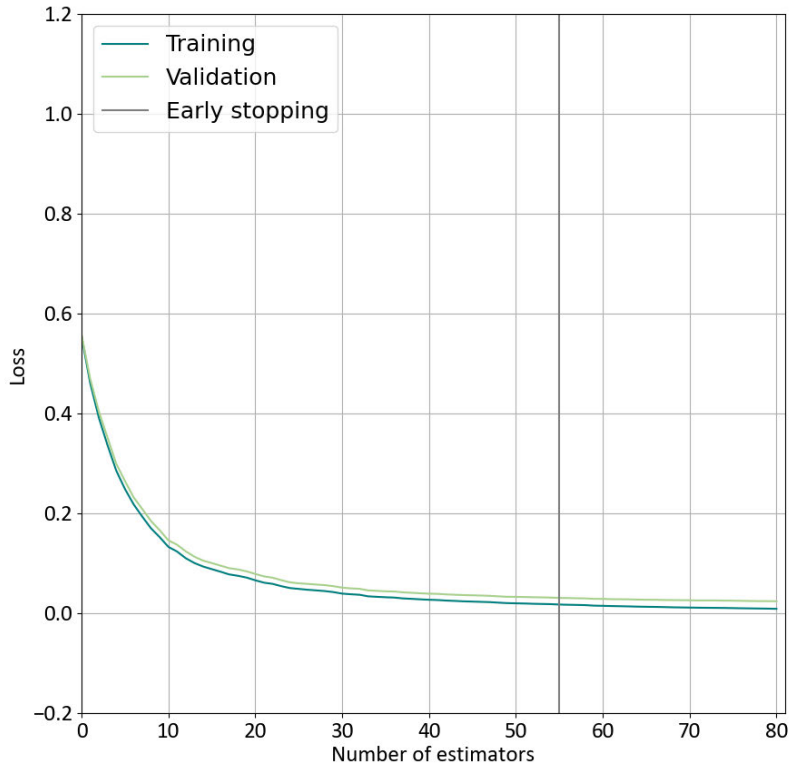


Figure A4: Training and validation loss of the LightGBM Model 13 for the binary classification task.

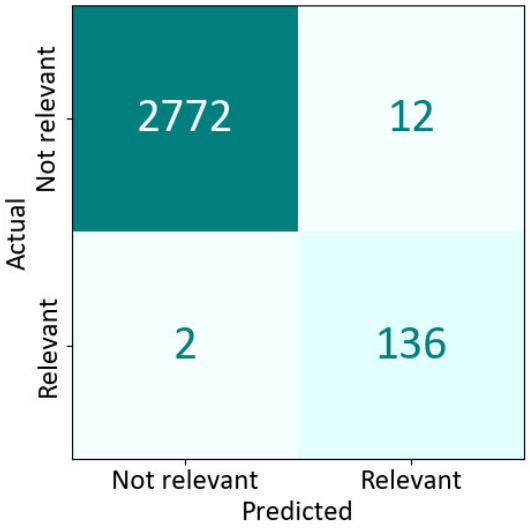


Figure A5: Confusion matrix for the test set with the LightGBM Model 13 for the binary classification task.

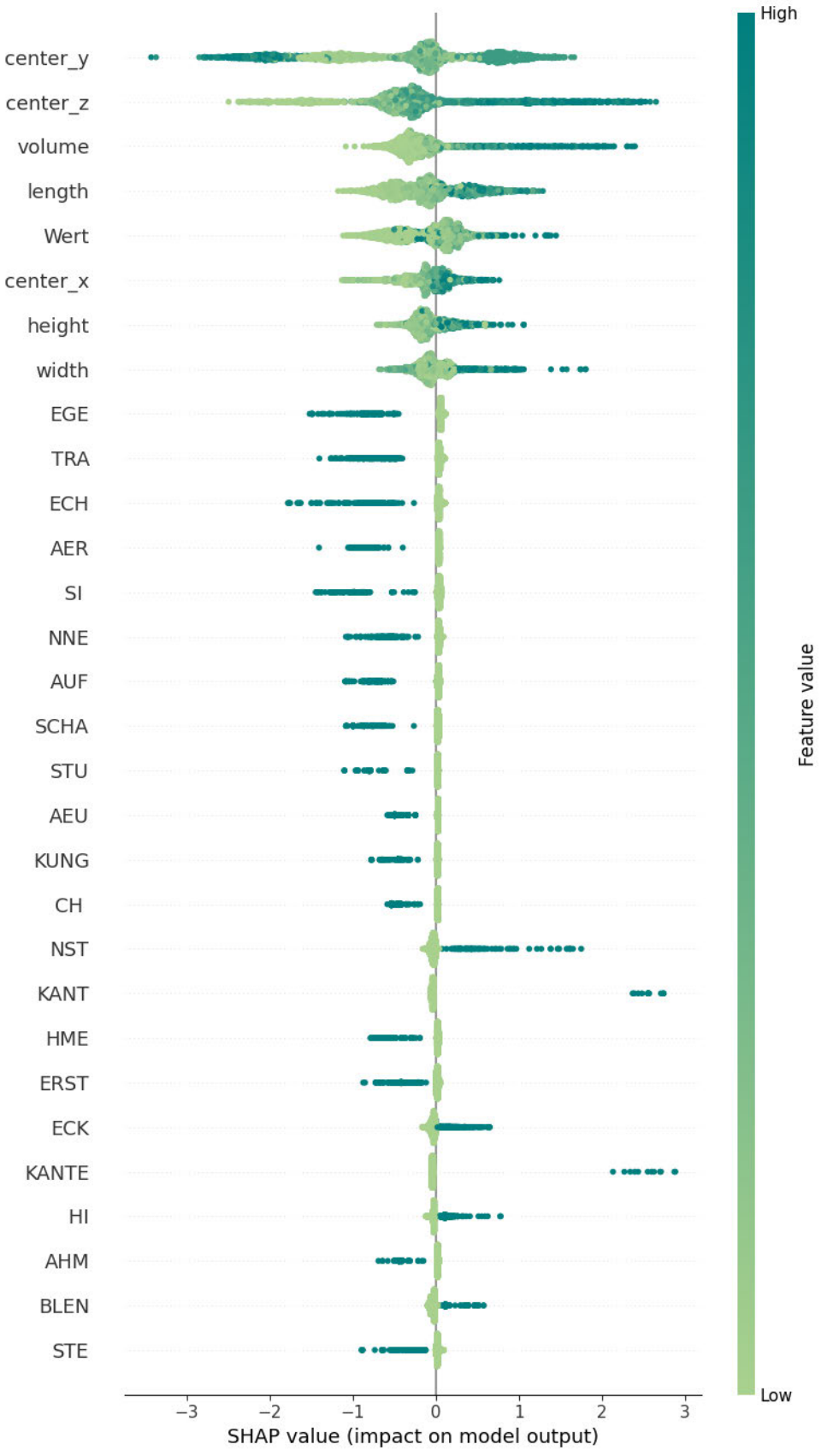


Figure A6: Feature importance for the LightGBM Model 13 for the binary classification task on the validation set.

Table A5: Hyperparameters of the top four LightGBM models for the multi-class classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				**Training complexity**	
Model	Tree depth	Learning rate	L2 regularization	Colsample by tree	Number of estimators	Training time in seconds
16	9	0.2	0.1	0.5	178	2
7	12	0.2	0.1	0.5	48	1
17	9	0.2	0.05	0.5	308	5
26	6	0.2	0.05	0.5	254	2

Table A6: Hyperparameters of the top four CatBoost models for the binary classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				Training complexity	
Model	Tree depth	Learning rate	L2 regularization	Bagging temperature	Number of estimators	Training time in seconds
78	9	0.3	0.2	0.5	46	117
79	9	0.3	0.2	1.0	46	118
80	9	0.3	0.2	1.5	46	116
24	9	0.1	0.2	0.5	111	106

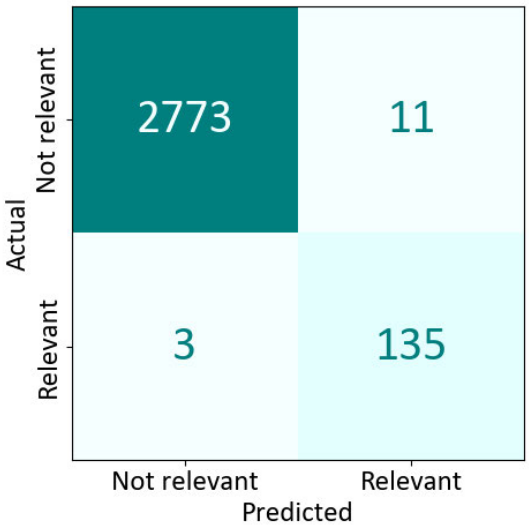


Figure A7: Confusion matrix for the test set with the CatBoost Model 24 for the binary classification task.

Table A7: Hyperparameters of the top four CatBoost models for the multi-class classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				Training complexity	
Model	Tree depth	Learning rate	L2 regularization	Bagging temperature	Number of estimators	Training time in seconds
73	9	0.1	0.2	1.0	543	1332
79	9	0.1	0.1	1.0	330	920
76	9	0.1	0.05	1.0	389	1024
63	6	0.1	0.2	0.5	453	164

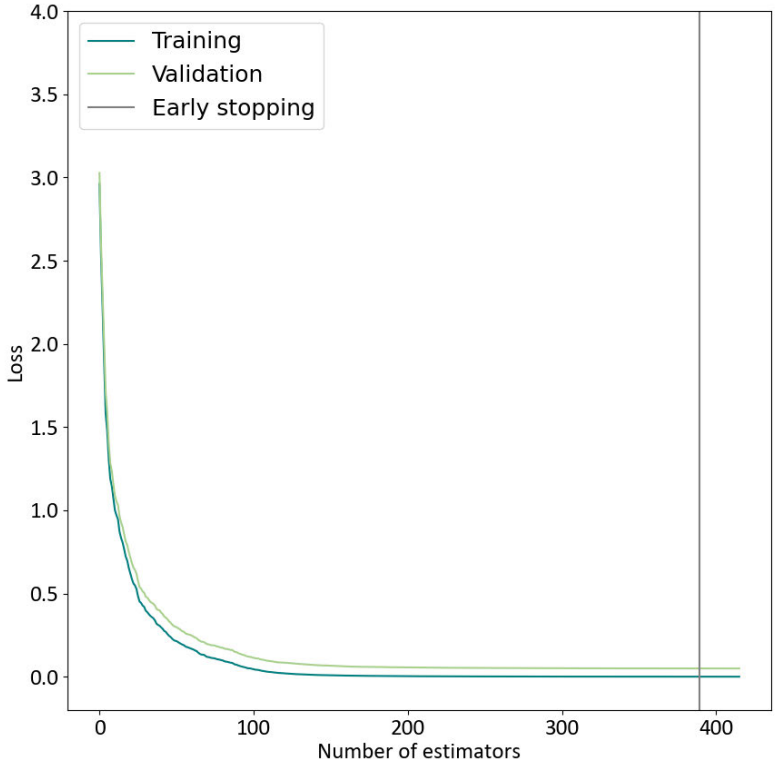


Figure A8: Training and validation loss of the CatBoost Model 76 for the multi-class classification task.

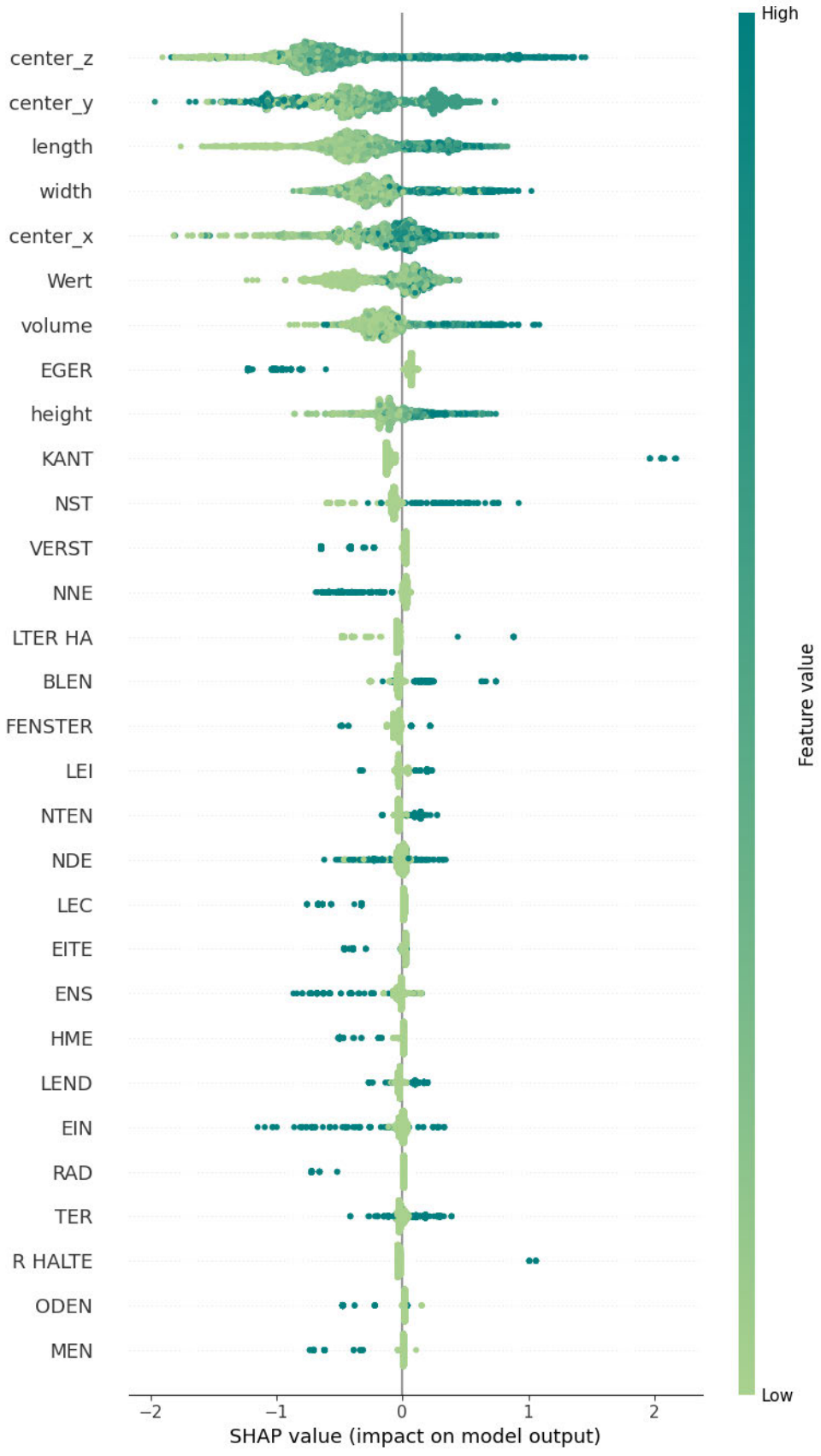


Figure A9: Feature importance for the CatBoost Model 76 for the multi-class classification task on the validation set.

Table A8: Hyperparameters of the top four PyTorch Tabular models for the binary classification task after grid-search hyperparameter optimization, including the resulting model training complexity.

	Hyperparameter				Training complexity	
Model	Layers	Activation Function	Batch size	Dropout	Number of epochs	Training time in seconds
13	1024-512-256	ReLU	128	0	41	292
26	1024-512-256	Sigmoid	512	0.1	199	895
7	1024-512-256	LeakyReLU	512	0	39	169
8	1024-512-256	LeakyReLU	512	0.1	40	175

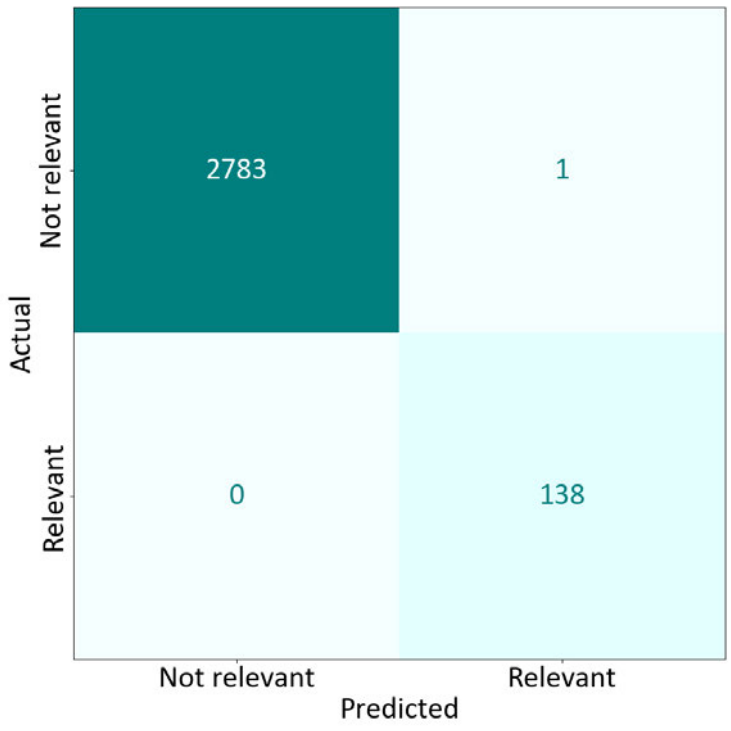


Figure A10: Confusion matrix for the test set with the PyTorch Tabular Model 26 for the binary classification task.

Bibliography

Haithm Alshari, Abdulrazak Yahya Saleh, and Alper Odabaş. Comparison of Gradient Boosting Decision Tree Algorithms for CPU Performance. 2021.

Roland Baddeley, Peter Hancock, and Peter Földiák. *Information Theory and the Brain*. 2000.

Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A Comparative Analysis of XGBoost. *Artificial Intelligence Review*, pages 1937–1967, 2021.

Leo Breiman. Bagging predictors. *Machine Learning*, pages 123–140, 1996.

Stephen Cass. The top programming languages. https://spectrum.ieee.org/the-top-programming-languages-2023, 2023.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. pages 785–794, 2016.

Tianqi Chen and Carlos Guestrin. XGBoost Documentation — xgboost 2.0.1 documentation. https://xgboost.readthedocs.io/en/stable/index.html, 2022.

Tianqi Chen and Tong He. Xgboost: eXtreme Gradient Boosting. 2023.

Filip de Roos, Carl Jidling, Adrian Wills, Thomas Schön, and Philipp Hennig. A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization. 2021.

Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support, 2018.

Piotr Florek and Adam Zagdański. Benchmarking state-of-the-art gradient boosting algorithms for classification. 2023.

Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, pages 119–139, 1997.

Yoav Freund and Robert E Schapire. A Short Introduction to Boosting. 1999.

Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, pages 1189–1232, 2001.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Andrey Gulin. CatBoost Documentation. https://catboost.ai/en/docs/, 2023.

Shereen Ismail, Zakaria El Mrabet, and Hassan Reza. An Ensemble-Based Machine Learning Approach for Cyber-Attacks Detection in Wireless Sensor Networks. *Applied Sciences*, page 30, 2023.

Manu Joseph. PyTorch tabular: A framework for deep learning with tabular data. *CoRR*, 2021.

Spyridon Kardakis, Isidoros Perikos, Foteini Grivokostopoulou, and Ioannis Hatzilygeroudis. Examining Attention Mechanisms in Deep Learning Models for Sentiment Analysis. *Applied Sciences*, page 3883, 2021.

Guolin Ke. LightGBM 4.0.0 documentation. https://lightgbm.readthedocs.io/en/stable/index.html, 2023.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Curran Associates, Inc., 2017.

Ivana Marin, Ana Kuzmanic Skelin, and Tamara Grujic. Empirical evaluation of the effect of optimization and regularization techniques on the generalization performance of deep convolutional neural network. *Applied Sciences*, 2020.

Hildegard Müller. VDA Jahresbericht. https://www.vda.de/de/aktuelles/publikationen/publication, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, pages 2825–2830, 2011.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: Unbiased boosting with categorical features. 2017.

Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An Introduction to Docker and Analysis of its Performance. 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. 2018.

B. Ramsundar and R.B. Zadeh. *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly Media, 2018.

Sebastián Ramírez. FastAPI. https://github.com/tiangolo/fastapi, 2018.

Sebastian Ruder. An overview of gradient descent optimization algorithms. 2016.

C E Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, pages 379–423, 1948.

Gregory G Slabaugh. Computing Euler angles from a rotation matrix. 2020.

Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. 2017.

Mohammad Mustafa Taye. Understanding of machine learning with deep learning: Architectures, workflow, applications and future directions. 12(5), 2023.

David Tolin. Doing CBT: A Comprehensive Guide to Working with Behaviors, Thoughts, and Emotions. 2016.

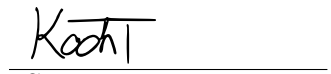
Gokul Yenduri, Ramalingam M, Chemmalar Selvi G, Supriya Y, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj G, Rutvij H. Jhaveri, Prabadevi B, Weizheng Wang, Athanasios V. Vasilakos, and Thippa Reddy Gadekallu. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. 2023.

Dongyang Zhang and Yicheng Gong. The comparison of lightgbm and xgboost coupling factor analysis and prediagnosis of acute liver failure. *IEEE Access*, pages 220990–221003, 2020.

Declaration of Authorship

Ich erkläre hiermit gemäß §9 Abs. 12 APO, dass ich die vorstehende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Abschlussarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Place, Date



Signature