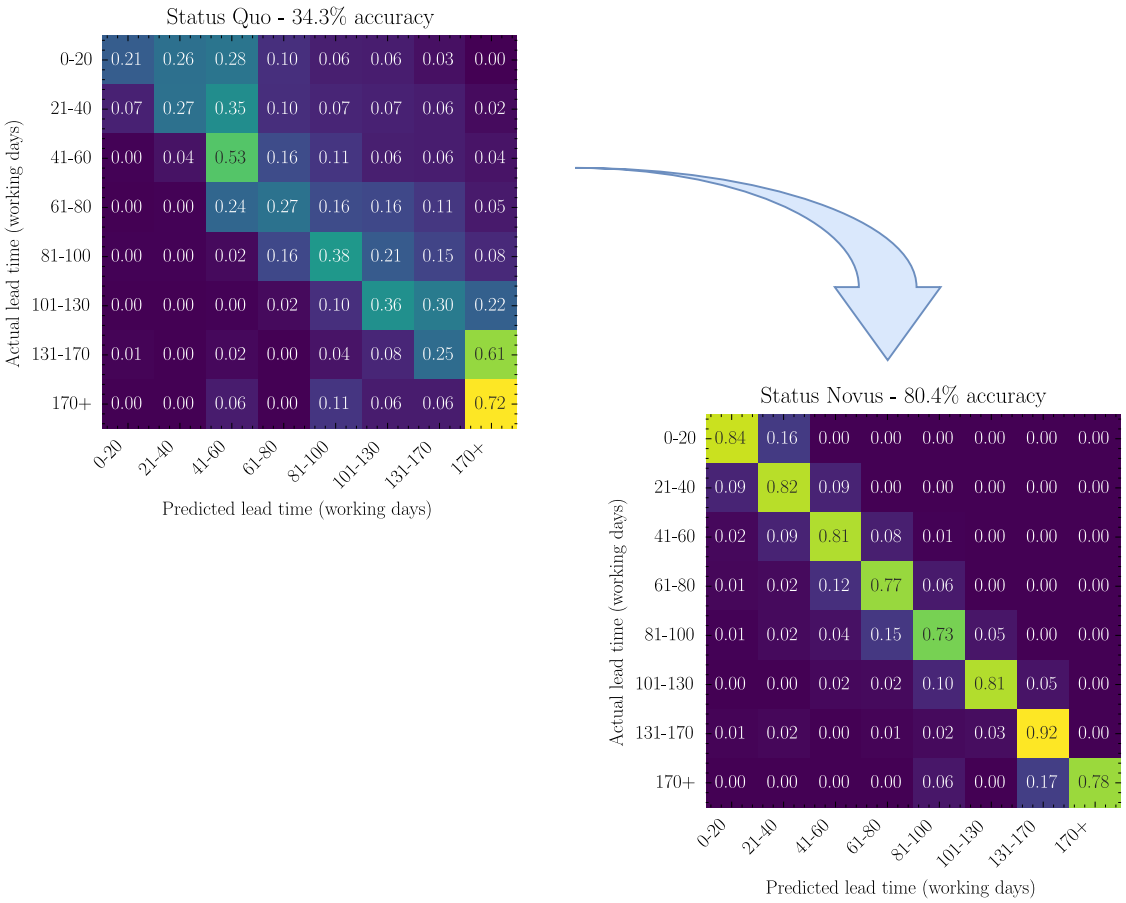
CHALMERS
UNIVERSITY OF TECHNOLOGY



Lead time forecasting for supplier management using machine learning

Complex Adaptive Systems

Fredrik Sitje
Felix Waldschock

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis in Complex Adaptive Systems

Lead time forecasting for supplier management using machine learning

Fredrik Sitje
Felix Waldschock

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Lead time forecasting for supplier management using machine learning
Fredrik Sitje
Felix Waldschock

Supervisor: Mattias Wahde, Department of Mechanics and Maritime Sciences
Examiner: Mattias Wahde, Department of Mechanics and Maritime Sciences

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

Cover: The cover shows two confusion matrices of prediction of lead times. On the left there is status quo, and on the right the confusion matrix of the best performing model of this thesis. A more dominant diagonal indicates better prediction performance.

Typeset in LaTeX
Gothenburg, Sweden 2025

Lead time forecasting for supplier management using machine learning
A machine learning approach to improve forecasting of lead times in the supply chain, evaluated at Maxon Motor AG
Master's Thesis in Complex Adaptive Systems
Fredrik Sitje
Felix Waldschock
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Forecasting lead times in the supply chain of manufacturing companies is a business-critical, yet time-consuming task. Typically, suppliers are contacted regularly to provide lead time estimations for parts, a process often prone to bias, especially if suppliers have an incentive to sell express deliveries. To address this issue, a machine learning pipeline was developed and tested using a case-study involving Maxon, a high-precision electric drive manufacturer. The pipeline considers the steps of data collection, filtering, feature engineering, selection, and model training. Multiple machine learning models were evaluated, and the best-performing model achieved a prediction accuracy of 0.804 on a test data set, where the supplier's estimation reached an accuracy of 0.343. This is a promising result. Furthermore, a time series analysis indicates that the model's performance improves with larger data sets, suggesting the pipeline's potential for real-world manufacturing environments.

Keywords: Machine learning, Supply chain, Forecasting, Feature selection, Feature engineering

Acknowledgements

We would like to express our sincere gratitude to all those who supported and contributed to the successful completion of this master thesis.

First and foremost, we would like to thank Maxon International AG for providing us with the opportunity to work on this project. Special thanks go to Thomas Gittler, Chief Digital Transformation Officer (CDTO) at Maxon International AG, for his continuous guidance, valuable insights, and unwavering support throughout the course of our work.

We would also like to extend our heartfelt appreciation to Chalmers University of Technology for their academic support and resources. We are particularly grateful to our supervisor, Professor Mattias Wahde, for his expert guidance, constructive feedback, and constant encouragement, which were instrumental in shaping the direction and quality of this thesis.

Fredrik Sitje, Felix Waldschock, Gothenburg, June 2025

Contents

1
Introduction

Global supply chains are complex systems consisting of networks of suppliers, manufacturers and distributors operating under local conditions. For a manufacturing company like Maxon International AG, a Swiss manufacturer of high-precision electric drives, the accurate forecast of delivery times is crucial to efficiently plan production capacities and ensure competitive delivery times. Currently, the planning of delivery times for procurement parts is done through manual intermediate steps (i.e., on the phone or via mail with the supplier) via the enterprise resource planning (ERP) system, a system for managing business processes such as orders and supply chains. This approach is labor-intensive, not scalable and often based on unreliable, conservative lead time estimates from the suppliers themselves. To address these challenges, this work pursues a data-driven approach that uses historical order data to predict delivery times with machine learning models. In Switzerland, such data-driven approaches are becoming increasingly important, as a recent market study shows.

A current market study form 2024 [1] shows that 35% of Swiss small and medium-sized enterprises (SMEs) use data analytics for process automation and 38% are exploring generative AI. However, many SMEs lack the understanding of machine learning as well as the specialists to implement it. This thesis addresses this gap by developing a data-driven model for Maxon International AG, a typical Swiss SME with complex supply chains due to small batch sizes and high product diversity. This demonstrates the relevance of machine learning for supply chain optimization in Switzerland.

1.1 Aims and limitations

The aim of this thesis is to develop a data-driven machine learning model for predicting delivery times for components such as printed circuit boards or semi-finished products from international suppliers at Maxon International AG. Another goal is to compare different machine learning algorithms, such as perceptron, decision trees and ensemble learners, as well as multiclass classification and regression models, to determine the best prediction accuracy. The methodological approach comprises three steps:

First, the aggregation and cleaning of historical purchase order data, and the identification of incorrect entries. *Secondly*, the analysis of feature redundancies and importance using data-driven methods. *Third*, the training and comparison of machine learning models.

Maxon International AG serves as a case study to evaluate the results and compare the models. The work focuses on Maxon's Swiss production site and excludes other sites in order to limit the analysis to a representative scenario. No operational prediction system is implemented, as the focus is on model development. Only supervised learning models are considered. Simple statistical methods (median-based) are used as a benchmark for the trained models. In addition, the models are not designed for extreme events such as the COVID-19 supply chain crisis, as such "black swans" have unpredictable patterns.

The goal of the model training is to find patterns based on order-specific information, such as part or supplier characteristics, and derive a purchase lead time prediction from them. For the lead time prediction, two approaches are chosen: firstly, training multiclass classifiers where the lead times are previously discretized and binned (e.g., 0–10 days → class 0; 11–20 days → class 1); secondly, regression models are trained, whose output is a floating-point number that directly represents the predicted lead time. The following chapters describe the theoretical foundations, the methodology, and the results of the case study at Maxon.

2
Theory

Accurate prediction of lead times of Maxon suppliers will rely on machine learning methods applied to order-related datasets. Challenges like statistical noise and the *curse of dimensionality* - where high-dimensional data lead to sparse distributions in data and reduced model accuracy - require a structured approach to mitigate their effects. This enables improved training times of the machine learning algorithms used [2]. In this chapter, we outline a theoretical pipeline to address these challenges, which includes feature engineering to transform existing raw data, feature selection to identify relevant and non-redundant features, and machine learning architectures for multiclass classification and regression. These steps should make the development of robust models for predicting lead times at Maxon's production site in Switzerland possible. The pipeline begins with feature engineering, which enables the creation of information-rich features, as explained in the next section.

2.1 Feature engineering

Feature engineering is a process of applying predefined rules to extract additional information from data points, leading to information-rich features that enhance the predictive power of machine learning models. Given that the data are either numerical, categorical, or temporal, feature engineering methods range from simple transformations, like logarithmic scaling of numerical values [3] to more advanced methods like cyclic encoding, for temporal features [4]. This section explores these techniques, illustrating their application to create predictive features.

2.1.1 Numerical feature transformations

Numerical features like quantities or price often require transformations to improve their suitability for machine learning models, because the raw data may have skewed distributions or have varying scales that some models do not respond well to. Simple methods include logarithmic scaling to reduce skewness in data distributions, squaring to capture non-linear relationships, or normalization to ensure consistent scales for all features. Other transformations such as differences between features can capture interactions relevant to supply chain dynamics which in turn is closely tied to domain knowledge. These transformations on numerical features can enhance the predictive ability of models.

2.1.2 Categorical feature encoding

Categorical features, such as supplier names or part types, require, depending on the machine learning model, an encoding to convert non-numerical data into numerical features. Common methods are target encoding, which replaces categories with a statistical value of the target (e.g., mean or median), label encoding which assigns a number as an unique ID, or one-hot encoding which creates a new dimension based on the feature values. Creating new features by string concatenation (e.g., supplier and part type) to form new features are also used to create more information-rich features. Target encoding can be especially effective for features with high cardinality since it does not introduce additional dimensions like one-hot encoding while preserving predictive information [5].

2.1.3 Temporal feature engineering

Temporal features, such as dates require specialized techniques to capture time-based patterns. Extracting categorical features such as month or weekday from a date (e.g., "YYYY-MM-DD"), can provide insights into seasonal or weekly trends affecting lead times. However, simple label encoding (e.g., January = 1, December = 12) introduces discontinuities, as January and December are actually close together but numerically far apart. This is addressed by cyclic encoding to better capture the continuous, cyclical behavior of features, e.g., month. This creates two numerical features using the sine and cosine functions, as shown below:

$$\begin{aligned} \text{month-sin} &= \sin\left(2\pi \times \frac{m-1}{12}\right), \\ \text{month-cos} &= \cos\left(2\pi \times \frac{m-1}{12}\right) \end{aligned} \tag{2.1}$$

where m is the month number. For example, applying cyclic encoding to order dates ensures that the model captures the cyclical nature of months, which in turn enhances predictions of lead times if there are seasonal variations [4]. Such techniques make temporal features more robust for machine learning models.

2.1.4 Role of domain knowledge

Domain knowledge plays a critical role in feature engineering by guiding the creation of features that reflect real-world supply chain dynamics or internal processes. As an example, understanding Maxon's manual handling of ordering processes may suggest combining features to capture their coupled impact on lead times. Manual feature engineering informed by experience and expertise, often produces highly relevant features but requires iterative testing to validate their predictive value [6]. In Maxon's context, domain knowledge ensures that engineered features align with operational realities, enhancing the robustness of lead time predictions.

2.2 Feature selection

Feature selection is used to identify the most information-rich features out of a set of available features, for a given problem. This not only benefits the prediction accuracy and reduces computational cost, but may make the prediction results more interpretable by focusing on only a very limited number of features. This is the second step in the pipeline which focuses on finding redundant features, removing them, and ranking their relevance to predict the lead time. To find the optimal feature set, that balances redundancy and relevance, iterative improvements are necessary until a satisfactory result is found [2]. These methods for assessing feature relevance and redundancy are explored by illustrating their application or theoretical basics.

2.2.1 Relevance and redundancy

Relevance measures are used to determine a feature's predictive power for lead times, while redundancy calculates the information overlap between features that may reduce model performance. By assessing both, a systematic consideration of features can be implemented and thereby create an information-rich set of features. Methods for evaluating relevance or redundancy are correlation-based for numerical features, mutual information-based for non-linear dependencies [7], and similarity based for feature names (e.g., PartInformation-SupplierId ≈ SupplierInformation-SupplierId). This subsection explores these methods and aims to give a basic understanding of the methods used.

2.2.1.1 Pearson correlation

The Pearson correlation $\rho_{X,Y}$ is well suited to determine a linear dependency for numerical features, so it is a measure of redundancy. The strength of the dependency is specified between $[-1, 1]$.

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{2.2}$$

The calculation uses the covariance $\text{cov}(X, Y)$ and the standard deviations, σ, of the features to be compared. No linear correlation applies for the case $\rho_{X,Y} = 0$, whereas a perfect positive or negative linear correlation applies for $\rho_{X,Y} = \pm 1$. This method has already been successfully used in feature selection to calculate the relevance of features [8] and might also be useful to determine redundancy between numerical features such as e.g., part-price and order-quantity. For relevance the same equation applies, but every numerical feature is compared to the target i.e., the lead time. Since this approach only recognizes linear correlations a more robust information theory based method can complement the Pearson correlation.

2.2.1.2 Mutual information

Mutual information is a proven method of calculating non-linear correlation between discrete and continuous features. It shows how much information two features share and is based on Shannon entropy, which takes a probability density function $p(x)$ to

calculate the information content of a random feature X, where $x \in \mathcal{X}$ represents the possible values of X, and $\mathcal{X}$ is the set of all possible outcomes for X.

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{2.3}$$

The conditional entropy can be used to quantify how much information is required to determine the result of a random feature Y in the presence of a random feature X that is known, where $y \in \mathcal{Y}$ are the possible values of Y, and $\mathcal{Y}$ is the set of all possible outcomes for Y.

$$\mathrm{H}(X|Y) := -\sum_{y \in \mathcal{Y}, x \in \mathcal{X}} p(y, x) \log \frac{p(y, x)}{p(y)} \tag{2.4}$$

By subtracting the conditional entropy from the Shannon entropy, the mutual information can be calculated.

$$\mathrm{I}(X; Y) = \mathrm{H}(X) - \mathrm{H}(X|Y) \tag{2.5}$$

This equation describes how much information the knowledge of the random feature Y provides about the random feature X. A high level of mutual information indicates a high level of redundancy [8]. However, it is difficult to compare multiple features, since the value depends on the entropy of the features, which grows with the number of possible outcomes. Therefore a normalization yielding a value between $[0, 1]$ can be used to determine redundant features.

One approach that is suitable for the normalization of the mutual information is to calculate the normalization as average normalized mutual information, which has the advantage of outputting a value between $[0, 1]$. To do this, use the definition

$$\mathrm{NI}(X, Y) = \frac{\mathrm{I}(X; Y)}{\min\{\mathrm{H}(X), \mathrm{H}(Y)\}} \tag{2.6}$$

to obtain the normalized mutual information. Further, $\mathcal{Y}_s = \{Y\}$ is a set of selected random features and $|\mathcal{Y}_s|$,$(s = 1, \ldots, |\mathcal{Y}_S|)$ is their cardinality.

$$\frac{1}{|\mathcal{Y}_s|} \sum_{Y \in \mathcal{Y}_s} \mathrm{NI}(X; Y) \tag{2.7}$$

This provides a measure for the average normalized mutual information, which serves as a correlation measure between random features. For a value of 0 it is assumed that the features are independent, for 1 it is assumed that one feature is a deterministic function of the other [9].

This method yields reliable and useful results but is costly compared to the next method that uses a shortcut to find redundancies between variables by calculating the similarity between feature names for clustering.

2.2.1.3 Jaccard and Jaro-Winkler analysis

The Jaccard index and Jaro-Winkler distance measure similarities in feature names and can be used to cluster them based on similarity. This method benefits feature

selection by domain knowledge, as the clustering can be automated. The Jaccard index describes a similarity of sets A and B with the ratio of intersection divided by union [10]. The sets can contain letters or words of feature names, for example.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.8}$$

The Jaro-Winkler distance measures the similarity between two strings based on their edit distance. The formula is given by:

$$s_w = s_j + \ell p(1 - s_j) \tag{2.9}$$

Here, s_w represents the Jaro-Winkler similarity, and s_j denotes the Jaro similarity. The parameter ℓ indicates the length of the common prefix (up to a maximum of 4 characters), and p is a weighting factor, typically set to 0.1, which scales the contribution of the prefix similarity. A limitation of this method is the need to determine an appropriate value for the parameter p, as it impacts the overall performance.

2.2.2 Balancing relevance and redundancy

To achieve an optimal selection of features for predicting lead times, it is essential to balance feature relevance with minimal redundancy. While methods discussed in Section 2.2.1 can be used to find redundant and relevant features, it is necessary to apply a systematic optimization technique in order to find an optimal subset of features. The following subsections introduce two approaches. Both iteratively refine the feature set to maximize predictive accuracy while minimizing redundancy, thereby creating compact and interpretable feature subsets for machine learning models.

2.2.2.1 Normalized mutual information feature selection

This is an approach proposed in [9] and is based on the following greedy pseudo-algorithm.

Algorithm 1 Normalized mutual information feature selection algorithm

1: **Initialization**: Create a set $F = \{f_i \mid i = 1, \ldots, N\}$, where N is the number of features, an empty set $S = \{\emptyset\}$, and set a threshold k defining the number of features that will be selected.

2: **Calculation of the mutual information**: For each feature $f_i \in F$, calculate the mutual information $I(f_i; C)$ to the target C according to equation (2.5).

3: **Select the first feature**: Select the feature $\hat{f}_i = \arg\max_{f_i \in F} I(f_i; C)$. Update:

$$F \leftarrow F \setminus \{\hat{f}_i\},$$
$$S \leftarrow \{\hat{f}_i\}.$$

4: **Greedy selection**: Repeat until $|S| = k$:

1. Calculate the mutual information $I(f_i; f_s)$ for all pairs (f_i, f_s), where $f_i \in F$ and $f_s \in S$.
2. Choose the feature $\hat{f}_i \in F$ that maximizes the following expression:

$$G = I(C; f_i) - \frac{1}{|S|} \sum_{f_s \in S} \mathrm{NI}(f_i; f_s), \tag{2.10}$$

where $\mathrm{NI}(f_i; f_s)$ is the normalized mutual information calculated with equations (2.6) & (2.7).

3. Update:

$$F \leftarrow F \setminus \{\hat{f}_i\},$$
$$S \leftarrow \{\hat{f}_i\}.$$

5: **Output**: Return the set S with the k selected features.

The function described in Equation (2.10) has the penalty term that causes the algorithm to create a feature set with features that have low mutual information among themselves but the highest possible mutual information for the target. This creates a compact, informative feature set with k features that is easy to understand. The disadvantage of this algorithm is that it creates a set of a local optimum that does not necessarily have to be the global one. This is due to the fact that higher-order interactions between three or four features, for example, are not taken into account. A method that can take this into consideration is considered in the next section.

2.2.2.2 Feature selection using genetic algorithms

The previously introduced methods for the feature selection focus on the direct connection between a feature f_A and a target feature f_T. It is not yet possible to say whether the combination of two features $f_A + f_B$ correlates better with the target f_T. The following two approaches introduce a method to search through the feature-combination-space semi-systematically and efficiently. This space describes all possible combinations of features, and is exponentially growing, 2^N with the

number of possible features (N). Both approaches are based on genetic algorithms (GAs), once employing *normalized mutual information* (Equation 2.6) and once a *decision tree classifier* (Section 2.3.1.2), and aim to find the optimal set of features $\mathcal{S}$. The following section gives a brief introduction to the field of GAs.

Introduction to genetic algorithms (GAs)

Genetic Algorithms (GAs) were introduced by Holland [11] and belong to the family of evolutionary algorithms. GAs are inspired by the principles of natural selection and survival of the fittest and are used to solve optimization problems. The following subsection covers the basic principles of GAs. Wahde [12] provides a clear introduction to genetic algorithms.

GAs can be used in high-dimensional, non-linear optimization, and are often able to escape local optima. The structure of a basic GA is described as pseudo-code in Algorithm 2. The individuals are represented using binary encoding, proposed by Holland [11], where each individual is converted to a string of 0s and 1s describing if a gene g_i is active or not, see Figure 2.1. Properly designing the fitness function is essential when applying GAs. An incorrectly specified evaluation method may embed misleading incentives in the system, which the GA can exploit and thus produce unintended results.

Algorithm 2 Structure of a basic genetic algorithm

1: **Initialization (P_0)**, create the first population by randomly creating n single individuals (I_i), where $i = 1, ..., n$ and n = population size
2: **Evaluation**, each individual is evaluated (assigned a fitness score) by a predefined measure and is ranked inside the population
3: **Create new generation (P_{t+1})**, based on the evaluation in step 2 the new population is formed by using the principles of:
   1. *Elitism*, the n best individuals survive (copy to next generation)
   2. *Mating*, select 2 individuals (parents) based on fitness score, to generate 2 offspring individuals by crossover with probability p_c (crossover probability)
   3. *Mutation*, loop over all new individuals, with probability p_m (mutation rate) mutate single genes
4: Go to step 2 and repeat until stopping criterion is reached (e.g., fitness threshold, number of generations, convergence threshold)

1. Genetic algorithm using mutual information

As introduced above, GAs strive to maximize the fitness value F of individuals and build on *good* solutions. This section describes a methodology that is used to find an optimal feature set $\mathcal{S}$ using the normalized mutual information. The binary string of each individual is used as a mask M to create a subset T from the original dataset $\mathcal{D}$.

$$T = \{s_i \in S | m_i = 1, i = 1, 2, ..., j\} \tag{2.11}$$

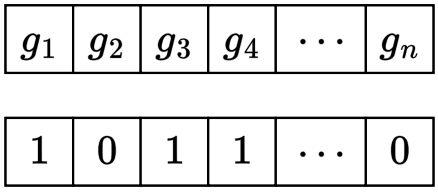


Figure 2.1: *Binary encoded individuals with j genes, where n describes the number of features that are represented in the dataset. Genes with the value* 1*, describe that a feature is used when the individual is evaluated.*

the features $\tilde{f}$ in this subset are string concatenated to form the new feature $\hat{f}$. For the fitness function the normalized mutual information (NI), introduced in Section 2.2.1.2, between $\hat{f}$ and the target feature f_t is computed, which is bounded between $[0, 1]$.

$$F = \frac{NI(\hat{f}, f_t)}{\rho} \tag{2.12}$$

The factor ρ in the denominator acts as a penalty score and is chosen arbitrarily. It is introduced to give the GA an incentive to find solutions with as few activated features as possible and is defined as:

$$\rho = \begin{cases} \frac{g}{d} & \text{if } g \geq d \\ 1 & \text{otherwise} \end{cases} \tag{2.13}$$

The variable g (gene count) describes how many features are chosen to compute the solution (sum of the binary string) and d (desired gene count) is an arbitrarily chosen integer that guides the GA to find solutions that have about this number of active genes. The case distinction helps to ensure that very small solutions ($N \approx 1$) do not obtain exploding fitness values.

A similar approach using a combination of GAs and mutual information is proposed by Emmert-Streib and Dehmer [13]. Their approach also aims to find and eliminate feature redundancy using the GA. As the combinatorial possibilities explode exponentially with base 2, this intermediate step is fulfilled using the methods described in section 2.2.1 at significantly lower computation costs.

2. Genetic algorithm using decision tree

Analogous to the previous approach, a decision tree and its accuracy on a test set are used as the base for the fitness score F instead of normalized mutual information. This method was previously proposed by Suciu and Lang [14]. To implement this, the dataset is initially partitioned into a separate training set $\mathcal{D}_{TR}$ and test set $\mathcal{D}_{TE}$. For each individual the set $\hat{\mathcal{D}}_{TR}$ and $\hat{\mathcal{D}}_{TE}$ is created based on $\mathcal{D}_{TR}$ and $\mathcal{D}_{TE}$, where the binary string of the individual is used as a mask to select the used features. Using $\hat{\mathcal{D}}_{TR}$ a decision tree, as introduced in Section 2.3.1.2, is built and the accuracy a is computed inferring $\hat{\mathcal{D}}_{TE}$ it. This accuracy is used for the fitness score:

$$F = \frac{a}{p} \tag{2.14}$$

the penalty score p is computed as described in Equation 2.13.

The result of both GA approaches is a list, I_1 and I_2, of individuals and their fitness. Each list I_i is filtered to only keep the individuals where the fitness is at $\geq$ 95% of the maximum fitness found, resulting in $\hat{I}_i$. Following, the frequency count for each feature in the filtered list is computed, to get an understanding of how often a feature is present in *good* solutions. *Good* is specified in this case by the threshold 95% of the maximum accuracy. According to this information the optimal set $\mathcal{S}$, which the GA aims to find, can be constructed. At this stage it is not certain that $\mathcal{S}$ represents the global optimum of selected features, but assumed that it is a reasonably good local optimum.

2.3 Model architectures

After feature engineering and feature selection the next step in the process pipeline is to select a suitable machine learning architecture to make robust predictions. Two different approaches exist that are suitable but differ fundamentally. Multiclass classification assigns data points to predefined categories, suitable for binned predictions (e.g., weeks, months) while regression provides precise numerical outputs. This section explores the different approaches, which helps to identify the most effective model for accurate and reliable lead time predictions.

2.3.1 Multiclass classifier

2.3.1.1 Single-Layer Perceptron

The perceptron is a machine learning algorithm, proposed by Rosenblatt in 1958 [15]. It is a linear model that connects McCulloch-Pitts neurons in a feed-forward network, and belongs to the group of supervised learning algorithms. The single-layer perceptron (SLP) comprises a two-layer network, consisting of an input and output layer. The interconnection between these two layers is described by the weight matrix W. Inputs X are fed into the network, multiplied by the individual weights $w_{i,k}$, passed through the activation function g, producing the output O.

$$f(X) = g(WX + \Theta) \tag{2.15}$$

Θ describes the bias terms, g is typically a sigmoid or ReLU. For each training iteration, the weights are updated according to the rule:

$$w_{i,j} = w_{i,j} + \eta(d_i - y_i)x_{i,j} \tag{2.16}$$

where d_i is the desired output, y_i is the predicted output and η is the learning rate. To obtain a probability distribution as the output, a softmax layer is applied, scaling the output values so that they sum to 1.

$$\hat{O}(X) = \text{softmax}(f(X)) \tag{2.17}$$

2.3.1.2 Decision tree (CART)

The CART algorithm was introduced in the 1980s by Breiman et al. [16]. The algorithm builds, data-driven, binary decision trees. This means that each internal node has exactly two branches, one that fulfills the condition described by the node and the other that does not. The algorithm determines these conditions - also known as splits - independently, using a metric such as the Gini index or entropy (Eq. 2.3), described in Section 2.2.1.2. At each level of the tree hierarchy, it evaluates which split can be used to divide the dataset most homogeneously, i.e., which split divides the classes best. Learning decision trees optimally is a NP-complete problem, discussed by Laurent et al. [17]. Therefore, a greedy algorithm is typically used to identify the best local split (i.e., for each node). However, this local best split does not guarantee that the global optimum, i.e., the perfect decision tree, is finally found.

2.3.1.3 Random forest

Random forest (RF) is fundamentally based on the idea that, instead of using only one decision tree for decision-making, multiple individual trees are used in an ensemble, see in Figure 2.2, and was proposed by Breiman et al. [18]. The term random forest refers to the use of random samples of the data to construct a large number (typically a few hundred) of individual decision trees. This ensemble approach significantly reduces the risk of overfitting. The three main concepts of random forest, *bagging*, *attribute sampling*, and *decision strategy* are discussed below:

Bagging: When the individual decision trees in a random forest are constructed, the bagging (also called bootstrap aggregation) method is typically used. This method proposes that each decision tree may not receive the complete dataset, but rather a random sample of about $\frac{2}{3}$ of the original dataset are drawn with replacement. Meaning a data sample can be drawn more than once. Hence, no single decision tree has access to the entire dataset, which contributes to the robustness of the model and helps mitigate overfitting.

Attribute sampling: In addition to sampling data points via bagging, there is also the method of attribute sampling (also called feature sampling). In this approach, the features used to train each individual decision tree are randomly selected. This method ensures that different split criteria are used, leading to improved diversity among the decision trees, reducing the likelihood of twin trees (exact copies).

Decision strategy: The output of the random forest is ultimately a combination of the outputs of each individual tree. The output of a single tree is the probability distribution describing the likelihood of the input being assigned to a class k, as the vector c with shape $(k \times 1)$. Below two strategies are introduced how the individual outputs are processed to compute the final output.

1. **Majority vote of the combined distributions**, here the outputs of the individual n trees are globally accumulated, leading to the output vector $\tilde{c}$ of

size $(k \times 1)$:

$$\tilde{c} = \sum_{i=1}^{n} c_i \tag{2.18}$$

from the vector $\tilde{c}$ the prediction class $\hat{k}$ is chosen with:

$$\hat{k} = \text{argmax}(\tilde{\text{c}})$$

2. **The Winner Takes It All**, here the argmax function is applied for each tree individually, and the local winning class is determined. Only the identity of the local winner is forwarded to the random forest. The class that wins the majority of votes across all trees is selected as the final prediction of the RF.

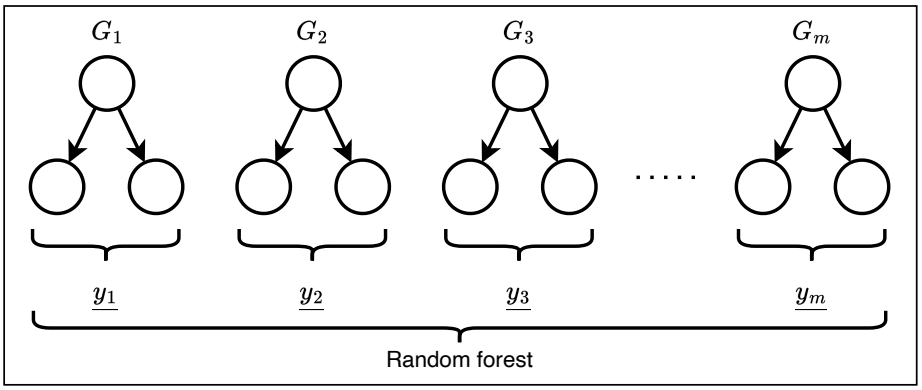


Figure 2.2: *Layout of a Random Forest consisting of m trees. Trees are visualized with single split for the sake of clarity, but typically contain more layers. Each tree G_i leads to an output $\underline{y_i}$. Output $\hat{y}$ of the random forest, is computed with one of the above introduced strategies.*

2.3.1.4 Gradient boosted trees

Gradient boosting trees (GBTs) were introduced by Friedman [19]. In this approach, multiple decision trees are trained sequentially, with each tree G_m aiming to minimize the error of its predecessor G_{m-1}. The error metric is chosen based on the problem; for multiclass classification, cross-entropy loss is commonly used [20]. The training procedure involves iteratively updating the model by adding new decision trees that minimize the loss function. Therefore, so-called pseudo-residuals (R_i) are introduced, which describe the error of the previous model relative to the target. The update rule is given by:

$$G_m(x) = G_{m-1}(x) + \eta \times h_m(x) \tag{2.19}$$

where:

- $G_m(x)$: Model prediction at iteration step m
- G_{m-1}: Model prediction at previous iteration step
- η: Learning rate (defines the impact of a single training step)
- $h_m(x)$: The decision tree fitted on the pseudo-residuals at iteration m

- x: Input data

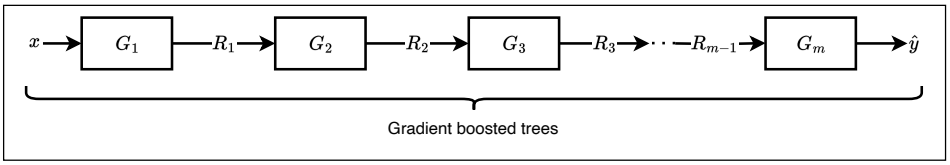


Figure 2.3: *Layout of a gradient boosted trees. Trees are trained consecutively, where the subsequent tree tries to minimize the error of its predecessor.*

The GBT is trained until a certain termination criterion is reached (i.e., maximum number of trees, or an accuracy threshold on the validation set). For a multiclass classification task with k classes, the algorithm trains k tree sequences individually, where each GBT predicts a binary output, whether an input belongs to class k or not.

2.3.1.5 CatBoost classification

CatBoost is a gradient-boosted-trees technique, similar to what is described in subsection 2.3.1.4. The *Cat* in CatBoost stands for *categorical* and stems from the fact that the algorithm handles categorical features. In [21], it is suggested to preprocess the categorical features using the ordered target encoding method before they are fed into the GBT. Regular target encoding was proposed by Micci-Barreca [5], and offers an alternative to commonly used one-hot or label encoding schemes. It is particularly useful and successful with high cardinality features. For a binary classification task the computation of these proxy values V is defined as follows:

$$V_i = \lambda(n_i)\frac{n_{iY}}{n_i} + (1 - \lambda(n_i))\frac{n_Y}{n_{TR}} \tag{2.20}$$

Where λ is used to balance between the prior and the posterior, and is computed based on the frequency of the category in the feature. Micci-Barreca proposes the use of the sigmoid function to compute λ. The prior describes the likelihood of the target being positive for the given category, the posterior describes the likelihood of the target being positive in the entire training dataset, while neglecting the category.

Prokhorenkova et al. [21] argue, that this method leads to data leakage, and therefore propose the method of ordered target encoding, where the target encoding is performed sequentially for each data point, and only previously processed data points are taken into consideration for the target encoding.

2.3.2 Regression models

Regression models are used to predict continuous target features, e.g., supplier lead times. The basis is linear regression, which enables a prediction for linear relationships but has weaknesses that can be counteracted with regularization [22]. There are various methods for non-linear regression, one of which is CatBoost regression, which is based on gradient boosted decision trees which is especially suitable for high cardinality categorical features.

2.3.2.1 Regulated linear regression

Linear regression can be used to make predictions in cases of linear relations.

$$y = \mathbf{x}^T \boldsymbol{\beta} \tag{2.21}$$

where $\mathbf{x} = [1, x_1, \ldots, x_n]^T$ are features and $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_n]^T$ are the coefficients of the features.

$$\mathrm{O_l} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.22}$$

The closed objective function for linear regression $\mathrm{O_l}$ in Eq.(2.22) is optimized by an analytical solution where the objective function minimizes the Ordinary Least Squares (OLS) and thus determines the coefficients $\boldsymbol{\beta}$.

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{2.23}$$

However, the method has the weakness that it tends to overfit many features and is therefore poorly suited as a reliable predictor. It suffers particularly when the data are multicollinear or there are too many features. These disadvantages can be mitigated with a regularization term.

$$\mathrm{O}_r = \sum_{i=1}^{n} (y_i - \hat{y}i)^2 + \lambda \sum_{j=1}^{n} \beta_j^2 \tag{2.24}$$

$$\boldsymbol{\beta}_r = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \tag{2.25}$$

One method is to complement the objective function O_r with a penalty term as described in Equation 2.24, which is still closed. Thus an analytical solution exists for this function. The penalty term squares the coefficients and multiplies them by λ, which defines the level of the penalty. Previous research has shown that this type of regularization improves the stability and interpretability of the coefficients Eq. (2.25) in models with collinearity by reducing the variance of the estimators, leading to more robust conclusions, but the choice of the optimal parameter λ is crucial for model performance [23].

$$\mathrm{O}_{\hat{l}} = \sum_{i=1}^{n} (y_i - \hat{y}i)^2 + \lambda \sum_{j=1}^{n} |\beta_j| \tag{2.26}$$

A similar method uses L1 regularization (Eq. 2.26) instead of L2 regularization (as used in Equation 2.24), which allows the coefficients to be set to zero. Thus, the insignificant features are omitted, which is effectively a feature selection [22]. Since the objective function $\mathrm{O}_{\hat{l}}$ in Equation (2.26) is not closed, there is no analytical solution, thus an iterative numerical method must be applied to find the optimal coefficients $\boldsymbol{\beta}_{\hat{l}}$.

2.3.2.2 CatBoost regression

CatBoost regression builds on the same algorithm as explained for CatBoost classification (Section 2.3.1.5). The fundamental difference lies in the target variable and

in the objective function that is minimized. CatBoost classification is created for discrete target values and CatBoost regression is designed for a continuous target variable. Therefore the loss function is changed from logloss to a mean squared error (MSE) that penalizes larger deviations more heavily.

Despite this change the algorithm retains its strength in handling categorical data through ordered target encoding.

3
Methods

This chapter describes the methods used to filter the dataset, and to split it into subsets for training, validation, and testing. Furthermore, the setup of the feature selection methods is detailed, the different algorithms and their parameters for the classification and regression tasks are introduced. Finally, the evaluation methods of these models are discussed.

3.1 Data introduction

Purchase Order Lines (POLs), as referred to in Maxon's internal ERP system, serve as the foundation for the data analysis. These entries represent all purchases made by Maxon. In addition to goods that Maxon integrates into its products, the data also includes items such as software licenses, vehicles of the company fleet, and office supplies. Consequently, it is necessary to filter this data as an initial step.

Developing domain knowledge about the dataset is essential in order to apply appropriate filters and ensure a clean and relevant data foundation. For this purpose, subject-matter experts from within the company were consulted. Two of the applied filters are, firstly, the exclusion of internal suppliers - only purchases from external suppliers are considered - and secondly, the inclusion of only those parts that are used in market-ready products (not prototyping phase).

The data are from the time range 1.1.2023 to 1.4.2025 and structured in tabular format consisting of rows (referred to as data points) and columns (referred to as features). The following naming conventions are used, $\mathcal{F}$ set of features, $\mathcal{F}_C$ set of categorical features, $\mathcal{F}_N$ set of numerical features, and $C(f_c)$ cardinality of the categorical feature f_c.

The original dataset $\mathcal{D}$ consists of 46,815 purchase orders, in which Maxon ordered components - 18,786 unique items-from 855 distinct suppliers. Each order in the dataset includes several features, such as `SupplierName`, `SupplierCountry`, `PartName`, `PartGroup`. After filtering, the number of data points is reduced to $11,565$. This dataset is subsequently divided into three subsets: the training set ($\mathcal{D}_{TR}$), the validation set ($\mathcal{D}_V$), and the test set ($\mathcal{D}_{TE}$), with a ratio $[0.6, 0.1, 0.3]$. The splitting is performed chronologically, ensuring that $\mathcal{D}_{TR}$ contains the earliest 60% of data points.

For each order, the expected lead time (L_E) - the lead time recorded in the ERP system - and the actual lead time (L_A) - the actual time it took for the item to be delivered to Maxon - are known. The value L_A is used as the ground truth for training the classifiers / predictor. It is a positive real number representing

the number of working days (Mondays - Fridays) between the order date and the delivery date. This value ranges between 1 and 455 days. The distributions of L_A and L_E are shown in Figure 3.1. L_E are typically higher than L_A:

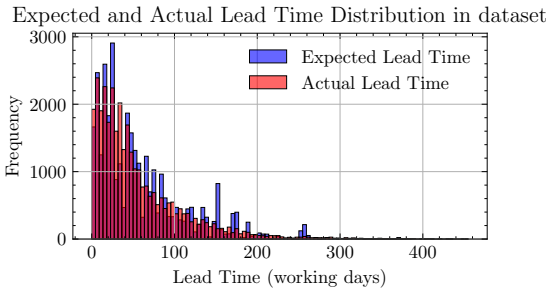


Figure 3.1: *Distribution of L_E and L_A. L_E is on average 4 working days longer (60 days) than L_A. $median(L_A) = 38$, while $median(L_E) = 45$ business days.*

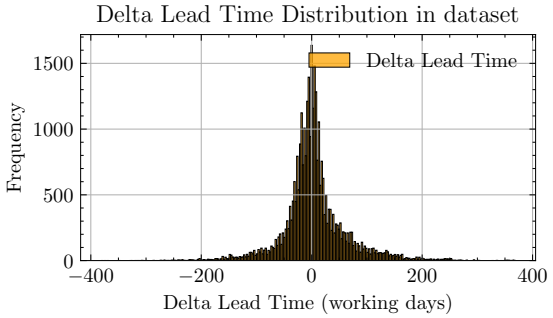


Figure 3.2: *Distribution of lead time delta between expected and actual lead times. Median 0 days, mean 4.6 days and $\sigma = 57.5$.*

	Median LT	Mean LT
L_E	45	59.9
L_A	38	55.2

Table 3.1: *Basic statistics of (L_E) and (L_A).*

3.2 Feature engineering

All the feature engineering methods already described in Section 2.1 were applied. We paid particular attention and spent a lot of time on feature engineering where we already suspected that process-specific knowledge could generate relevant features. The knowledge was based on many meetings and interviews with process owners and developed through hours of process analysis.

Two of the features that were created: `FE_StressFactor_NUMERICAL`, and `FE_DELTA_YEARS_PurchPart_DateCreated_minus_POL_DateEntered_NUMERICAL`. We noticed that we can measure how delayed an order is and created `FE_StressFactor_NUMERICAL` from this. We also suspected that it plays a role how long we have been purchasing a part and this could be measured and is reflected by `FE_DELTA_YEARS_PurchPart_DateCreated_minus_POL_DateEntered_NUMERICAL`.

3.3 Feature selection

With feature engineering, the number of features increased, but not all of them have the same significance compared to the target. In addition, the redundancy between the features had to be taken into account so that the model performance would not be negatively influenced by noise. Selecting the right features is a crucial

step in order to reduce the complexity of the models and avoid overfitting. In this chapter, we will discuss the methods we used to identify the most important features.

The methods have in common that all features with no values, 95% missing values or constant values were deleted at the beginning, as the relevance of these features was considered to be insignificant. The remaining features were divided into categorical or numerical. In addition, supposedly numerical features such as `PartNo` were manually changed to categorical variables. Subsequently, missing categorical values were replaced with `missing`. For the numerical features, a negative marker value based on three times the minimum value of the column was set to indicate missing values. This sentinel value was chosen so that it always consists of one or more nines (e.g., -9, -99) in order to clearly identify it as an imputation value.

3.3.1 Redundant feature detection using string similarity

This method was used to find ten numerical and ten categorical variables that had the lowest possible redundancy to each other and yet were highly relevant in relation to the target. The redundancy measure was applied inside of feature clusters based on feature names. The selection is based on a greedy selection paired with deep domain knowledge. For the calculations of mutual information the *Scikit* package was used, and the function `mutual_information_regression` applied. To make this function applicable for categorical features, a label encoding was used before calculating the mutual inforamtion.
First, the features were divided into numerical and categorical features. For both feature types, a descending sorted list was created with the mutual information score between features and target required for greedy selection. The feature names were then tested for similarity and clustered using Jaccard index (threshold 0.8) and Jaro-Winkler similarity (threshold 0.75). Within the cluster, features with a mutual information score of less than 4.0 were then removed and the feature with the highest score from the mutual information lists was selected. This was done separately for the numerical and categorical features. Then, the top 10 of both feature types were selected and sorted based on the mutual information score on the target. Lastly, the top 10 features were selected, taking domain knowledge into account.
Since we needed a lot of domain knowledge for this approach, we aimed for a more automated approach to feature selection, which is discussed in the next section.

3.3.2 Normalized mutual information feature selection

This algorithm was implemented exactly as described in Section 2.2.2.1 and with $k = 20$ to accommodate features that had to be dropped after review based on domain knowledge. Label encoding was again used for the categorical variables to make them compatible with the function `mutual_information_regression` from the *Scikit* library. The entropy was calculated with the function `entropy` from the *SciPy* library.

3.3.3 Guided wrapper with genetic algorithm

For both GAs, the one using mutual information and the one using decision tree, the same parameters are chosen:

Parameter	Symbol	Value
Population Size	N	100
Crossover Probability	p_c	0.9
Number of Generations	G	60
Minimal Mutation Rate	μ_0	$\|\mathcal{F}\|^{-1}$

Table 3.2: *Genetic Algorithm Parameters*

The $|\mathcal{F}|$ describes the number of features used in a solution. The mutation rate changes over the generations with the following equation:

$$\mu_m(g) = \max(\mu_0, 5 \times \mu_0(\frac{1}{1 + \exp(-g + \delta)})) \tag{3.1}$$

where the parameter δ is used to determine when the decay starts, and g is the generation count. Bit strings of individuals are initialized randomly with 97% chance of the bit being 0 and 3% of being 1, this ratio is chosen to let the GAs start with small sets of selected features (230 features $\times$ 3% = 6.9 features are used on average in the initial round), this due to the fact that solutions with small sets shall be found and to reduce computation time for the decision tree approach.

3.3.4 Validating feature selection with domain knowledge

Besides the systematic approaches to detect the necessary features for the problem, a domain-knowledge-based approach is also used. This as a final step of the entire feature selection process, enabling a direct validation step of the previously found features. Therefore the specific Maxon knowledge from the authors as well as from internal experts is used. This step interrogates every feature, which was outputted in the selection methods before.

Features that were not included are for example `SupplierForPurchPart_InternalControlTime_NUMERICAL` as there is no connection between internal control time and the supplier or, e.g., `POL_Cf_Standard_Loc_CATEGORICAL` as the location of the delivery is irrelevant for the delivery.

3.4 Model training

As previously outlined, two approaches are explored for the prediction of the purchase part lead times. The first approach formulates the problem as a multiclass classification task, where the continuous lead time values, from the historical dataset, are discretized into eight predefined bins. For this task, suitable classification algorithms (perceptron, decision trees, random forest, gradient boosted trees and CatBoost) are used.

The second approach formulates the problem as a regression task, aiming to predict the lead times as a continuous numerical value. This approach allows for higher resolution of the prediction. This method is more sensitive to outliers and noise. The used algorithms are LASSO, ridge, CatBoost, and gradient boosted trees.

3.4.1 Multiclass classifiers

For the multiclass classification, the historical lead times in the dataset are replaced by their corresponding class k, which is defined in Table 3.3. For the first five classes, an equidistant spacing of four working weeks, is chosen, after reaching 100 working days the spacing is expanded to 30 working days and the last class is a semi-open interval with only a lower limit. These definitions were modeled in close consultation with procurement experts at Maxon.

For all discussed models the same split of training ($\mathcal{D}_{TR}$), validation ($\mathcal{D}_V$) and test data ($\mathcal{D}_{TE}$) is used, primarily to ensure a fair comparison of architectures. The following section describes the selection of the training parameters for the different model architectures, the model performance is discussed in the Results chapter 4. To find an optimal set of parameters, grid search is used, and aims to maximize the accuracy on the validation set $\mathcal{D}_V$.

Class	**Lead time range [working days]**
1	0-20
2	21-40
3	41-60
4	61-80
5	81-100
6	101-130
7	131-170
8	170+

Table 3.3: *Binning of lead times into* 8 *classes. Ranges are determined with experts at Maxon*

3.4.1.1 Simple estimator (Benchmark)

As a initial model, we use basic statistics on the historical data to predict future lead times. Therefore building a model, that takes the historical orders of a specific part into consideration to predict the future lead time of it. More precisely, we propose to use the median of the last γ orders of a specific part, where γ is an integer value described the maximum number of orders that are taken into consideration.

Therefore, a lookup table $\mathcal{L}$ is created, based on the L_A, actual lead time, (the information before discretization into the eight bins), using the data points of $\mathcal{D}_{TR}$, with the following procedure:

Algorithm 3 Calculate lead time for parts

1: Sort $\mathcal{D}_{TR}$ for *DateEntered* descending ▷ latest order on top
2: list-of-unique-partNo $\leftarrow \mathcal{D}_{TR}$[“PartNo”].unique() ▷ extract unique part numbers
3: $\mathcal{L} \leftarrow \{\}$ ▷ initialize dictionary for lead times
4: **for** partNo in list-of-unique-partNo **do**
5: $\hat{\mathcal{D}}_{TR} \leftarrow \mathcal{D}_{TR}[\mathcal{D}_{TR}[\text{“PartNo”}] == \text{partNo}]$ ▷ filter by partNo
6: $\hat{\mathcal{D}}_{TR} \leftarrow \hat{\mathcal{D}}_{TR}.\text{head}(\gamma)$ ▷ take top γ entries
7: $\mathcal{L}[\text{partNo}] \leftarrow \text{median}(\hat{\mathcal{D}}_{TR}[\text{“ActualLeadTime”}])$ ▷ compute median lead time
8: **end for**

Using the computed $\mathcal{L}$ the lead times for $\mathcal{D}_{TE}$ can be predicted. The parameter γ is optimized with a grid search, where $\mathcal{D}_V$ is used as the validation set, and defined as $\gamma = 7$. This approach only holds for predictions, where the part is present in $\mathcal{D}_{TR}$.

3.4.1.2 Perceptron

The perceptron is implemented using the Python library *PyTorch* and employs `CrossEntropyLoss` as the loss function to be optimized. The chosen optimizer is Adam [24], a momentum-based algorithm. The parameter to be tuned, is the learning rate η introduced in Equation 2.16. The shape of the input vector is (127×1), and represents the result of a rule set which defines the encoding scheme, described in Algorithm 4.

Algorithm 4 Feature encoding scheme for perceptron and decision tree

for $f \in \mathcal{F}$ **do**
  if $f \in \mathcal{F}_N$ **then**
    Encode f using Min-Max-Scaler
  else if $f \in \mathcal{F}_C$ **then**
    if $C(f_c) < 50$ **then**
      Apply One-Hot Encoding to f
    else
      Apply Label Encoding to f
      Apply Min-Max-Scaler to f
    end if
  end if
end for

3.4.1.3 Decision tree (CART)

The Python library *Scikit* is used to implement the decision tree, specifically the `DecisionTreeClassifier` class, which is based on the CART algorithm [16]. The features are preprocessed according to the scheme described in Algorithm 4.

Table 3.4: *Hyperparameter ranges and best settings for the grid search of the decision tree model.*

Hyperparameter	Values Tested
`max_depth`	{7, 10, 15, 20}
`min_samples_split`	{2, 3, 5, 8}
`min_samples_leaf`	{3, 5, 8, 10}
`criterion`	{gini, entropy}

3.4.1.4 Random forest

For the implementation of the random forest model the Python library *YDF* [25] is employed. To find an optimal set of parameters a grid search is conducted, described in Table 3.5.

Table 3.5: *Hyperparameter ranges and best settings for the grid search of the Random Forest model.*

Hyperparameter	Values Tested
`n_estimators`	{100, 200, 300}
`max_depth`	{10, 30, 50, 70}
`min_samples_split`	{1, 3, 5}
`max_samples`	{0.8, 0.9}

3.4.1.5 Gradient boosted trees

The GBT model employs, like the RF-model, the *YDF* library [25]. The library is capable of handling categorical features natively, using a similar approach to target encoding [5]. Using grid search, an optimal set of hyperparameters is found, see Table 3.6.

Table 3.6: *Hyperparameter ranges and best settings for the grid search of the gradient boosted trees model.*

Hyperparameter	Values Tested
`n_estimators`	{100, 200, 300}
`max_depth`	{3, 5, 7, None}
`learning_rate`	{0.01, 0.1, 0.3}
`subsample`	{0.7, 0.8, 0.9}

3.4.1.6 CatBoost classifier

The realization of the *CatBoost classifier* model utilizes the homonymous library. For the hyperparameter tuning, a grid search is carried out, described in Table 3.7.

Table 3.7: *Hyperparameter ranges and best settings for the grid search of the CatBoost model.*

Hyperparameter	Values Tested
`depth`	{6, 8, 10}
`learning_rate`	{0.01, 0.03, 0.08}
`l2_leaf_reg`	{1, 3, 5}
`min_data_in_leaf`	{1, 3, 5}

3.4.2 Regression

The regression models used the exact same data split setup like the classifier models, training ($\mathcal{D}_{TR}$), validation ($\mathcal{D}_V$) and test data ($\mathcal{D}_{TE}$). To make them comparable to the classification models the binning presented in Table 3.3 was applied to the predictions of the regressor.

For all discussed models the same split of training ($\mathcal{D}_{TR}$), validation ($\mathcal{D}_V$) and test data ($\mathcal{D}_{TE}$) is used, primarily to be able to fairly compare architectures.

3.4.2.1 CatBoost regression

The Table 3.8 represents the parameters that were used to conduct a grid search in order to find the lowest validation E_{RMS} (Equation 3.9) and the highest binned accuracy among the tested combinations.

Table 3.8: *Hyperparameter ranges tested in the grid search for the CatBoost regression model.*

Hyperparameter	Values Tested
`iterations`	{3500}
`depth`	{6, 8, 10}
`learning_rate`	{0.01, 0.03, 0.08}
`l2_leaf_reg`	{1, 3, 5}
`min_data_in_leaf`	{None, 3, 5}

3.4.2.2 Gradient boosted regression

Table 3.9 summarizes the hyperparameter ranges that were used in the grid search to find the optimal model based on E_{RMS} and accuracy.

Table 3.9: *Hyperparameter ranges tested in the grid search for the gradient boosted regression model.*

Hyperparameter	Values Tested
`n_estimators`	{500, 1000, 1500, 6000}
`max_depth`	{6, 8, 10}
`learning_rate`	{0.001, 0.01, 0.03, 0.08}
`min_samples_split`	{2, 5}

3.4.2.3 Lasso regression

The grid search results, shown in Table 3.10, indicate the optimal λ value that minimized validation E_{RMS}.

Table 3.10: *Hyperparameter ranges tested in the grid search for the lasso regression model.*

Hyperparameter	Values Tested
λ	{0.0001 to 10, logspace, 20000 steps}
`max_iter`	{100000}

3.4.2.4 Ridge Regression

Table 3.11 represents the hyperparameter range for the grid search of ridge regression to find the optimal penalty factor λ.

Table 3.11: *Hyperparameter ranges tested in the grid search for the ridge regression model.*

Hyperparameter	Values Tested
λ	{0.0001 to 1000, logspace, 20000 steps}
`max_iter`	{100000}

3.5 Model evaluation

Defining a clear evaluation of models is necessary to compare them. Therefore different metrics for the multiclass classification and the regression models are introduced below. This evaluation is conducted on the test set $\mathcal{D}_{TE}$.

3.5.1 Evaluation metrics for multiclass classification

To compare the different classification models the confusion matrix C, which provides a detailed breakdown of correct and incorrect predictions by comparing the

true labels with the predicted ones, is used. This squared matrix typically represents, with each row the instances of the true labels and each column the predicted classes. C is of dimensions $(t \times p)$, representing true class and predicted class, where $t = p = 1, \ldots, 8$.

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,p} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{t,1} & c_{t,2} & \cdots & c_{t,p} \end{pmatrix} \tag{3.2}$$

In order to make a quantitative assessment, the three metrics *Recall* R, *Precision* P, and *F1-score* F1 are calculated for each class k, as well as *global accuracy* A and *weighted F1-score* W for each model.

$$R_k = \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k} \tag{3.3}$$

$$P_k = \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k} \tag{3.4}$$

$$\mathrm{F1}_k = \frac{2 \times P_k \times R_k}{P_k + R_k} \tag{3.5}$$

$$A = \frac{\sum_k C_{k,k}}{\sum_{t=1}^{k} \sum_{p=1}^{k} C_{t,p}} \tag{3.6}$$

$$W = \sum_k \mathrm{F1}_k \times \alpha_k \tag{3.7}$$

where α_k represents the proportion of class k among all true labels. This score helps to mitigate effects of imbalanced datasets.

The *true positives* (TP), *false positives* (FP), and *false negatives* (FN) for a class k are computed with:

$$\begin{aligned} \mathrm{TP}_k &= C_{k,k} \\ \mathrm{FP}_k &= \sum_{p \neq k} C_{k,p} \\ \mathrm{FN}_k &= \sum_{t \neq k} C_{t,k} \end{aligned} \tag{3.8}$$

3.5.2 Evaluation metrics for regression

To evaluate regression models, the metrics introduced in Section 3.5.1 are unsuitable, as they require discretized outputs. Therefore, the root mean square error (E_{RMS}) is employed instead.

$$E_{\mathrm{RMS}} = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{n}} \tag{3.9}$$

$\hat{y}_i$ describes the predicted value and y_i the ground truth, and n expresses the observations. Due to the square, the sign of the error is neglected, and therefore errors cannot cancel out one another.

3.6 Estimation of future model performance

To evaluate the performance of our best predictive model for future predictions, we implemented a methodology that simulates how the predictive ability evolves as the amount of training data increases. To do so, we used a time series cross-validation strategy to simulate the performance of the model over growing parts of the data. Specifically, we used a `TimeSeriesSplit` function from the *Scikit* library in which the dataset was divided into 100 splits, with each split incrementally increasing the size of the training set by approximately 1% of the total data. This allowed us to evaluate the performance of the model as it was exposed to increasingly larger training sets, mimicking real-world scenarios where data becomes available over time. Performance was evaluated using E_{RMS} to measure prediction accuracy for the continuous target, and classification accuracy based on binned predictions to evaluate the model's ability to correctly categorize the target into predefined intervals. Confusion matrices were calculated for both the training and validation sets to derive the accuracy metrics.

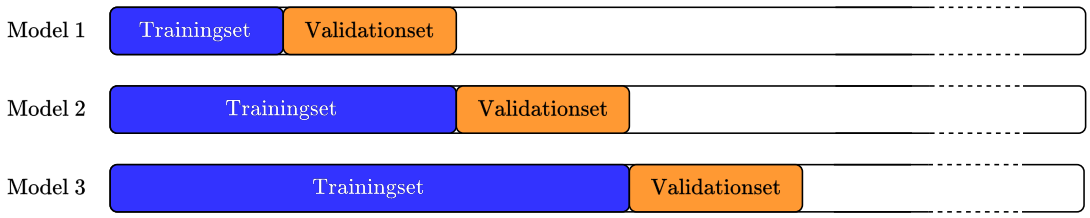


Figure 3.3: *Evolution of model performance across incrementally increasing training sets, with each segment representing a new model trained on the growing dataset. The sequence extends beyond the three depicted models to reflect 100 splits and the set sizes are not to scale only for illustrative purposes.*

4
Results

4.1 Feature selection

The NMIFS algorithm, as described in Section 2.2.2.1 found these $k = 20$ most relevant features:

InvPart_Cf_C_Cum_Leadtime_NUMERICAL, FE_DELTA_YEARS_PurchPart_DateCreated_minus_POL_DateEntered_NUMERICAL, FE_StressFactor_NUMERICAL, InvPart_Cf_Location_No_CATEGORICAL, SupplierForPurchPart_VendorManufLeadtime_NUMERICAL, SupplierForPurchPart_InternalControlTime_NUMERICAL, InvPart_AbcClass_CATEGORICAL, FE_POL_PlannedReceiptDate_Quarter_CATEGORICAL, InvPart_CountryOfOrigin_CATEGORICAL, InvPart_ExpectedLeadtime_NUMERICAL_binned, MasterPart_UomForWeightNet_CATEGORICAL, FE_POL_PlannedReceiptDate_Weekday_CATEGORICAL, SupplierForPurchPart_Cf_Purchase_Group_Desc_CATEGORICAL, FE_POL_LatestOrderDate_Quarter_CATEGORICAL, POL_EngChgLevel_NUMERICAL, FE_POL_DateEntered_Year_NUMERICAL, InvPart_FrequencyClass_CATEGORICAL, POL_StatGrpDescription_CATEGORICAL, FE_POL_LatestOrderDate_Weekday_CATEGORICAL, POL_SupplierAssortmentDescription_CATEGORICAL.

The GA algorithm using mutual information and decision trees, as described in Section 2.2.2.2, was applied to select the $k = 10$ most relevant features. The detailed results are found in Table 4.1. The evolution of the *Fitness*, *Accuracy*, and *Number of selected features* is shown in Figure 4.1.

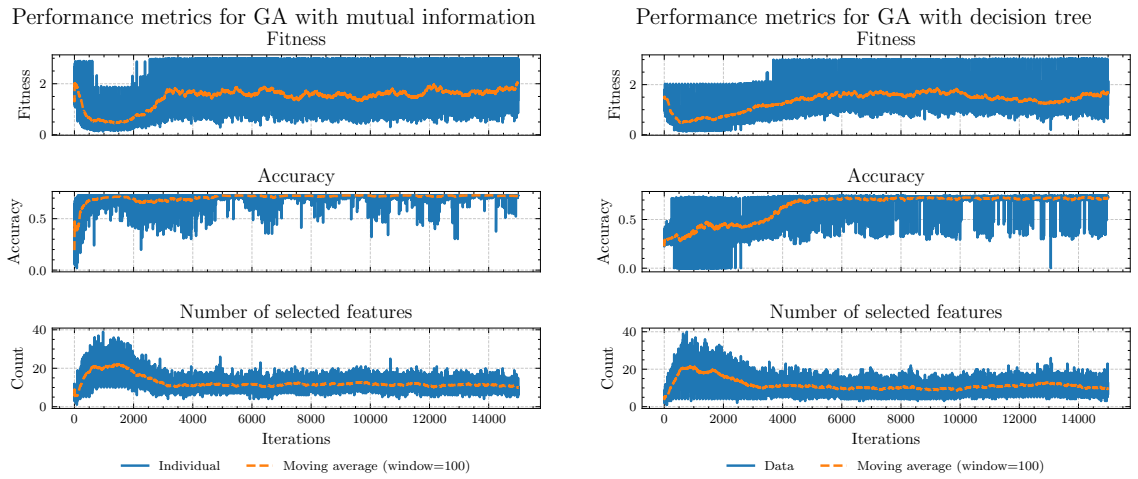


Figure 4.1: *Development of fitness, accuracy, and number of selected features during training of the GA with mutual information (left) and decision tree (right), using a fixed set of parameters (mutation rate, population size) for* 150 *generations. Every individual of every generation is shown on the x-axis (*150 *generations of each* 100 *individuals).*

Table 4.1: *Results of feature selection process using GA with mutual information and with decision tree. Features are ordered by frequency score descending.*

	GA MI			**GA DT**		
#	Feature Name	Frequency	Average Accuracy	Feature Name	Frequency	Average Accuracy
1	FEStressFactorNUM	10508	0.7255	FEStressFactorNUM	8492	0.7269
2	MasterPart-LotTrackingCodeCAT	9161	0.7241	InvPartCfCCum-LeadtimeNUM	8492	0.7269
3	MasterPart-MxnLegacyClassCAT	8913	0.7240	MasterPart-MxnSubmReqMatCompl-CAT	5986	0.7264
4	SupplierInformation-PurchaseGroupCAT	8546	0.7250	InvPartCfSmartBin-LocationsCAT	3918	0.7286
5	POLNetAmtBaseNUM	6167	0.7248	SupplierInformationCf-ContractsCAT	2557	0.7257
6	FEPOLDateEntered-MonthCAT	4765	0.7268	SupplierInformationCf-SupplierCatDescCAT	1646	0.7295
7	InvPartDopConnection-CAT	3042	0.7206	POLEngChgLevelNUM	1497	0.7249
8	POLBuyerNameCAT	2257	0.7246	POL-ExtTranspCalDescrCAT	1362	0.7311
9	PurchPart-OverDeliveryTolerance-NUM	2137	0.7203	SupplierInformation-PurchaseGroupCAT	1067	0.7276
10	PurchPart-DocumentTextCAT	1930	0.7254	POLEngRevisionDesc-CAT	893	0.7221

After performing feature selection using both `GA` and `NMIFS`, the resulting feature lists were compared. By validating these features with domain knowledge, the following final set of features was compiled:

`InvPart_PurchLeadtime_NUMERICAL`, `FE_DELTA_YEARS_PurchPart_DateCreated_minus_POL_DateEntered_NUMERICAL`, `FE_StressFactor_NUMERICAL`, `SupplierForPurchPart_VendorManufLeadtime_NUMERICAL`, `InvPart_AbcClass_CATEGORICAL`, `InvPart_CountryOfOrigin_CATEGORICAL`, `FE_POL_LatestOrderDate_Quarter_CATEGORICAL`, `InvPart_FrequencyClass_CATEGORICAL`, `SupplierForPurchPart_Cf_Purchase_Group_Desc_CATEGORICAL`, `FE_POL_DateEntered_YYYYQuarter_CATEGORICAL`, `PurchPart_StatGrp_CATEGORICAL`

4.2 Grid search for classifiers and regressions

Following, the results of the grid searches for the classification models are described in Table 4.2, the grid search results for the regression models are found in Table 4.3.

Table 4.2: *Comparison of classification model performance metrics on the validation set. An accuracy score for each model, along with the respective hyperparameters and training times. Accuracy is calculated based on a confusion matrix with classes representing the values as defined in Table 3.3.*

Model	Hyperparameters	Accuracy	Time
Decision tree	Max. depth 15 Min. samples split 3 Min. samples leaf 8 Criterion entropy	0.7550	3 s
Random forest	300 trees Max. depth 30 Min. samples split 3 Max. samples 0.8	0.7768	1 min 44 s
Gradient boosted trees	300 trees Max. depth 7 Learning rate 0.01 Subsample 0.9	0.7765	10 min 24 s
CatBoost classifier	Depth 8 Learning rate 0.03 L2 leaf reg. 5 Min. data in leaf 1	0.7880	9 h 44 min

Table 4.3: *Comparison of regression model performance metrics on the validation set. The table presents the E_{RMS} and accuracy for each model, along with their respective hyperparameters and training times. Accuracy is calculated based on the binning of regression predictions as defined in Table 3.3.*

Model	**Hyperparameters**	E_{RMS}	**Accuracy**	**Time**
CatBoost regression	3500 iterations Depth of 6 Learning rate 0.03 L2 leaf reg. 5 Min. data in leaf 5	10.9221	0.8062	7 min 51 s
Gradient boosted	6000 trees Max. depth 6 Learning rate 0.03 Min. samples split 5	13.2123	0.7811	14 min 39 s
Lasso regression	Alpha 0.4345 Max. iterations 100,000	11.8339	0.6955	2 min 15 s
Ridge regression	Alpha 117.4771 Max. iterations 100,000	12.2334	0.7052	1 min 11 s

4.3 Model evaluation

Table 4.4 and 4.5 describe the performance results of the trained classification models. Table 4.6 contains the performance results for the discretized regression models.

Table 4.4: *Performance metrics (Precision, recall, F1-score) across different bins for simple estimator, decision tree, and random forest classifiers, with global accuracy (Gl.-Acc) and weighted F1-Score (W.-F1) summarized at the bottom.*

	Simple estimator			Decision tree			Random forest		
Bin	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
0-20	0.491	0.407	0.445	0.770	0.838	0.803	0.796	0.841	0.818
21-40	0.384	0.335	0.358	0.752	0.735	0.743	0.764	0.783	0.773
41-60	0.396	0.383	0.390	0.756	0.752	0.754	0.794	0.805	0.799
61-80	0.283	0.301	0.292	0.742	0.763	0.753	0.755	0.720	0.737
81-100	0.270	0.246	0.258	0.777	0.607	0.682	0.752	0.681	0.715
101-130	0.197	0.269	0.227	0.759	0.770	0.765	0.818	0.826	0.822
131-170	0.194	0.254	0.220	0.741	0.809	0.774	0.900	0.848	0.873
170+	0.091	0.278	0.137	0.395	0.789	0.526	0.810	0.850	0.829
Gl.-Acc		0.332			0.752			0.783	
W.-F1		0.337			0.752			0.783	

Table 4.5: *Precision, recall, and F1-score for gradient boosting, CatBoost, and perceptron classifiers across bins, with global accuracy and weighted F1-score.*

	Gradient boosting			CatBoost			Perceptron		
Bin	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
0-20	0.815	0.865	0.839	0.807	0.795	0.801	0.466	0.258	0.332
21-40	0.796	0.735	0.764	0.747	0.777	0.762	0.400	0.116	0.180
41-60	0.735	0.849	0.788	0.761	0.832	0.795	0.303	0.873	0.450
61-80	0.768	0.567	0.652	0.784	0.696	0.737	0.162	0.094	0.119
81-100	0.588	0.709	0.643	0.745	0.674	0.708	0.050	0.003	0.005
101-130	0.650	0.726	0.686	0.798	0.779	0.789	0.324	0.054	0.092
131-170	0.909	0.642	0.753	0.864	0.864	0.864	0.328	0.142	0.198
170+	0.762	0.889	0.821	0.867	0.684	0.765	0.000	0.000	0.000
Gl.-Acc		0.745			0.773			0.310	
W.-F1		0.743			0.773			0.238	

Table 4.6: *Performance metrics of regression models with precision, recall, F1-score, global accuracy, weighted F1-score, and E_{RMS} comparison for CatBoost, gradient boost, lasso, and ridge models with binned predictions proposed in Table 3.3.*

	CatBoost			Gradient boosting			Lasso			Ridge		
Bin	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
0-20	0.830	0.814	0.822	0.817	0.739	0.776	0.832	0.466	0.597	0.842	0.550	0.665
21-40	0.782	0.833	0.806	0.721	0.794	0.755	0.582	0.663	0.620	0.625	0.678	0.650
41-60	0.830	0.808	0.819	0.788	0.816	0.802	0.686	0.763	0.722	0.692	0.761	0.724
61-80	0.767	0.772	0.769	0.771	0.779	0.775	0.670	0.728	0.698	0.681	0.748	0.713
81-100	0.751	0.711	0.731	0.782	0.715	0.747	0.688	0.710	0.699	0.731	0.729	0.730
101-130	0.837	0.842	0.840	0.817	0.769	0.792	0.804	0.830	0.816	0.798	0.832	0.815
131-170	0.906	0.868	0.887	0.879	0.773	0.823	0.888	0.826	0.856	0.875	0.804	0.838
170+	0.667	0.769	0.714	1.000	0.650	0.788	0.643	0.529	0.581	0.714	0.526	0.606
Gl.-Acc			0.804			0.777			0.686			0.708
W.-F1			0.803			0.777			0.683			0.707
E_{RMS}			10.427			11.178			10.958			11.171

4.4 Time series test

In the following section, the result of the time series test, introduced in Section 3.6 are displayed. Figure 4.2 describes the development of the accuracy over increasing training set size, while Figure 4.3 shows the E_{RMS} over training data size.

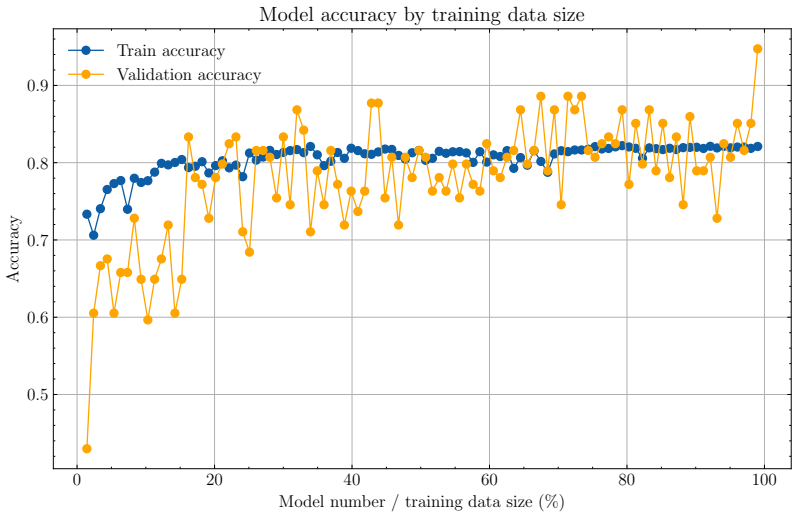


Figure 4.2: *This figure shows the accuracy of the model as the training data size increases from ≈1% to ≈99%. The method explained in Section 3.6 was applied and the time range of the data is [1.1.2023, 1.4.2025]. The train accuracy (blue) and validation accuracy (orange) both increase with rising data size, accompanied by some fluctuations. The slope of the curves indicates a rise in accuracy as more data is used, with validation accuracy showing a steeper incline at higher training percentages (e.g., beyond 80%).*

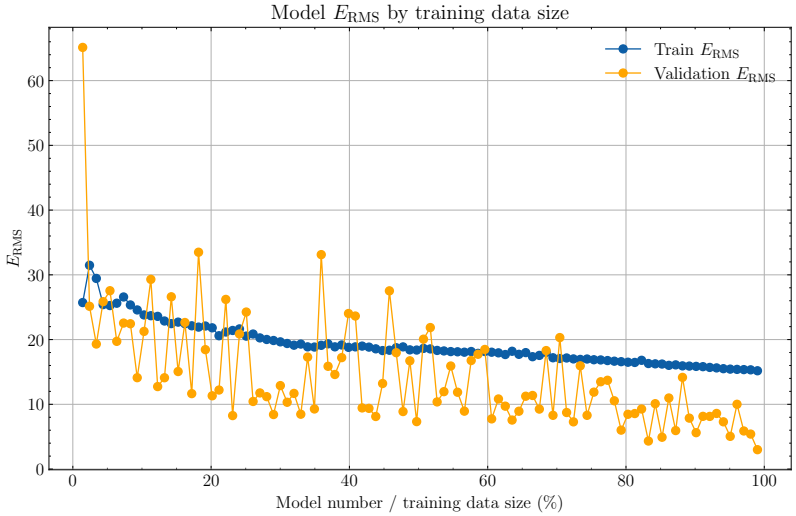


Figure 4.3: *This figure presents the E_{RMS} of the model as the training data size ranges from ≈1% to ≈99%. The method explained in Section 3.6 was applied and the time range of the data is [1.1.2023, 1.4.2025]. Both train E_{RMS} (blue) and validation E_{RMS} (orange) decrease with increasing data size, with the slope becoming more gradual as the data size approaches 100%. The curves reflect a downward trend in E_{RMS} as more data is incorporated.*

5
Discussion

The aim of this work was to use machine learning algorithms to create a lead time prediction that can improve the status quo. For this purpose, we followed a self-constructed machine learning pipeline that used different classification and regression algorithms and summarized their prediction capability in the Tables 4.4, 4.5 and 4.6.

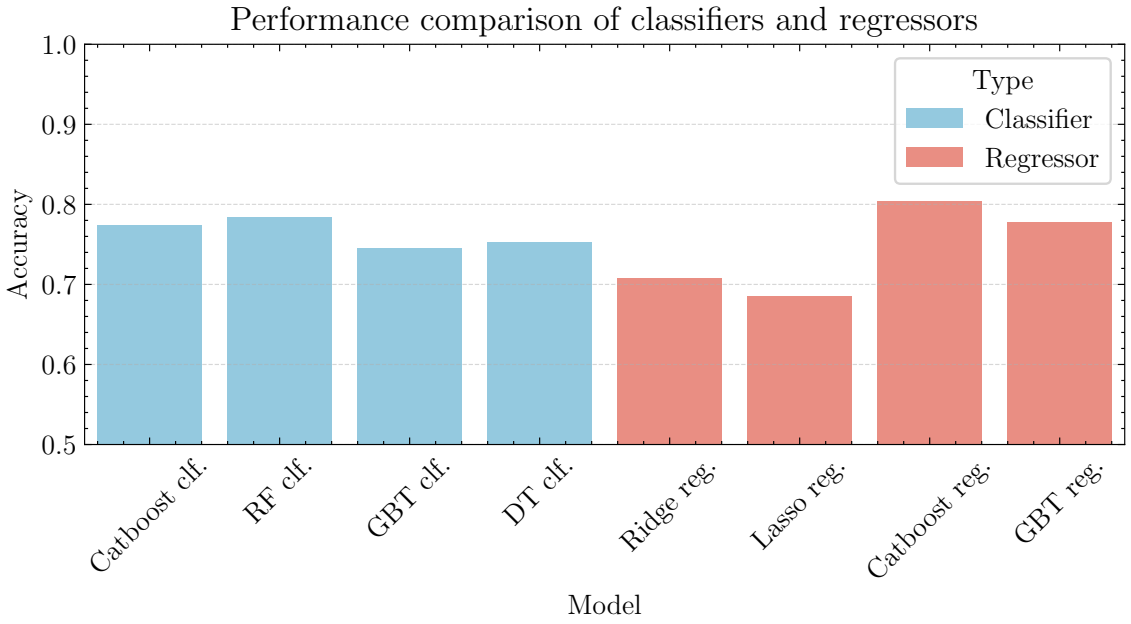


Figure 5.1: *Summary of model performance metrics across different machine learning models, highlighting the superior performance of the CatBoost regression model with a global accuracy of 0.804.*

As can be seen in Figure 5.1, CatBoost regression is the model with the highest accuracy of all models. The model achieves a global accuracy of 0.804 as shown in the Tables 4.6,4.4 and 4.5. The F1-scores per bin show a consistent performance across different delivery time ranges and the E_{RMS} is 10.427 indicating an accurate prediction of delivery times. Compared to classification models such as CatBoost classification (global accuracy 0.773) or random forest (global accuracy 0.783), the CatBoost regression outperforms all models and performs especially better in the lower bins (0-20, 21-40 days). The perceptron classifier collapses and predicts dominantly class 3; this may be due to the encoding of the input. That the regression model performs better than a classification model comes as a surprise, as it was

assumed that the discretizations of the regression predictions at the bin boundaries would lead to problems that would not occur with the classification. However, the increased resolution of the CatBoost regression seems to allow a finer discrimination that compensates for the problems of discretizations.

The results are satisfactory and correspond to what was expected. Nevertheless, there is room for improvement in some areas of the pipeline, starting with data filtering. In this work it was recognized that only with domain knowledge data can be filtered effectively and that this step is crucial for the performance of prediction models. This is because understanding how a large amount of data is linked together is impossible without knowledge of exact processes. The same is true for feature engineering where there is believed to be a high potential for improvement. In our case, it turned out that features that are already collected are not enough to make a good prediction. Only thanks to a deeper understanding of how processes run and how this can be shown with some features was it possible to significantly improve the forecasting performance of all models. For example, the feature `FE_StressFactor_NUMERICAL` increased the accuracy in each model by about 20%. Since the focus was strongly on feature engineering, there are possible improvements by testing different encodings of categorical variables, scaling skewed data points and testing different imputation strategies for missing values of orders. Improvements are also possible through a more comprehensive grid search. The results in Figure 5.1 are all based on label-encoding for categorical variables (except CatBoost models, which natively apply ordered target encoding), a scaling of the numerical values in the interval $[0, 1]$ and imputation of missing values by placeholders to indicate missing values. The feature selection methods used proved to be extremely effective. Although the final tuning had to be carried out manually, the appropriate features could be found quickly. Furthermore, fine-tuning the binning of the delivery times can greatly increase the quality of the predictions, although this can affect the accuracy of the models. For regressions, it would also make sense to use a variable metric for the performance of the prediction, e.g., *In what percentage of cases are the predictions within an interval of* $[0.9, 1.05]$ *of the actual value?*.

From the Figures 4.2 and 4.3 an apparent trend can be seen – with more data, the accuracy increases and the E_{RMS} of the CatBoost regressor decreases. Although the fluctuation of the validation accuracy remains high and fluctuations differ between $5 - 10\%$, it is possible that this is due to binning. The validation E_{RMS}, on the other hand, seems to have a decreasing fluctuation with increasing training data. Since the data is for a period of just over two years, we suspect that the model recognizes cyclical market dynamics, but the time period is the recovery phase of the COVID-19 pandemic, which introduces some uncertainty. Therefore, it is important to pay attention to this and, in the event of any changes to the forecasting capability, either the model or, more likely, the filtering and feature engineering must be adjusted.

6
Conclusion

This thesis shows that data-driven approaches using machine learning can enable significant improvements in the prediction of delivery times for complex supply chains such as those at Maxon International AG. By developing and applying a machine learning pipeline that includes both classification and regression models, a more accurate prediction of lead times was achieved. The supplier's estimation only reaches an accuracy of 0.343, while the CatBoost regression model – the best-performing model – achieved an accuracy of 0.804.

The results underline the importance of careful data management, especially data filtering and feature engineering, for model performance. In addition to potential improvements in model-specific matters, it is advisable to standardize data creation and preparation, making future forecasts easier and more accurate.

In summary, this work offers a valuable contribution to the optimization of supply chain planning at Swiss SMEs such as Maxon International AG. It shows that data-driven models can support manual, labor-intensive planning and thus enable more precise delivery time forecasts.

Bibliography

[1] "Einblick in die Nutzung von Künstlicher Intelligenz in Schweizer Unternehmen," https://fh-hwz.ch/news/einblick-in-die-nutzung-von-kuenstlicher-intelligenz-in-schweizer-unternehmen.

[2] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[3] J. Heaton, "An empirical analysis of feature engineering for predictive modeling," in *SoutheastCon 2016*. IEEE, 2016, pp. 1–6.

[4] A. Adams and P. Vamplew, "Encoding and decoding cyclic data," *The South Pacific Journal of Natural Science*, vol. 16, 01 1998.

[5] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," vol. 3, no. 1, pp. 27–32. [Online]. Available: https://dl.acm.org/doi/10.1145/507533.507538

[6] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. S. Turaga, "Learning feature engineering for classification." in *IJCAI*, vol. 17, 2017, pp. 2529–2535.

[7] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[8] H. Gong, Y. Li, J. Zhang, B. Zhang, and X. Wang, "A new filter feature selection algorithm for classification task by ensembling Pearson correlation coefficient and mutual information," *Engineering Applications of Artificial Intelligence*, vol. 131, p. 107865, 2024.

[9] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on neural networks*, vol. 20, no. 2, pp. 189–201, 2009.

[10] B. Liu *et al.*, *Web data mining: exploring hyperlinks, contents, and usage data*. Springer, 2011, vol. 1.

[11] J. H. Holand, "Adaptation in natural and artificial systems," *Ann Arbor: The University of Michigan Press*, vol. 32, 1975.

[12] M. Wahde, *Biologically inspired optimization methods: an introduction*. WIT press, 2008.

[13] F. Emmert-Streib and M. Dehmer, Eds., *Information Theory and Statistical Learning*. Boston, MA: Springer US, 2009.

[14] M.-A. Suciu and R. I. Lung, "Feature selection based on a decision tree genetic algorithm," in *Hybrid Artificial Intelligent Systems*, P. García Bringas, H. Pérez García, F. J. Martínez de Pisón, F. Martínez Álvarez, A. Tron-

coso Lora, Á. Herrero, J. L. Calvo Rolle, H. Quintián, and E. Corchado, Eds. Cham: Springer Nature Switzerland, 2023, pp. 433–444.

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[16] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[17] H. Laurent and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information processing letters*, vol. 5, no. 1, pp. 15–17, 1976.

[18] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[19] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[20] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[21] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," Jan. 2019.

[22] R. Jonas and J. Cook, "Lasso regression," *British Journal of Surgery*, vol. 105, no. 10, 2018.

[23] D. N. Schreiber-Gregory, "Ridge regression and multicollinearity: An in-depth review," *Model Assisted Statistics and Applications*, vol. 13, no. 4, pp. 359–365, 2018.

[24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017.

[25] M. Guillame-Bert, S. Bruch, R. Stotz, and J. Pfeifer, "Yggdrasil Decision Forests: A Fast and Extensible Decision Forests Library," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Aug. 2023, pp. 4068–4077.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY