



Vaasan yliopisto  
UNIVERSITY OF VAASA

Tomi Kallio

# **Usage of a multilayer perceptron Neural Network in the prediction task**

Predicting rare occurring components in product material lists

School of Technology and Innovations  
Master's Thesis in Automation and  
Information Technology

Vaasa 2025

---

**UNIVERSITY OF VAASA****School of Technology and Innovation**

**Author:** Tomi Kallio  
**Title of the Thesis:** Usage of machine learning in the prediction task: Predicting rare occurring materials in product material lists  
**Degree:** Master of Science  
**Programme:** Automation and Information Technology  
**Supervisor:** Jouni Lampinen  
**Year:** 2025      **Pages:** 83

---

**ABSTRACT:**

In modern production, the growing complexity of products, processes, and customization has led to the generation of large amounts of data. However, many companies struggle to make effective use of this data in their daily operations. This challenge is also present in the case company of this thesis, where delays in purchasing certain components have created difficulties in production scheduling. To address this, the thesis applies machine learning and predictive analysis to forecast the occurrence of components in the bill of materials (BOM). To achieve this, the study builds a prediction model on the company's historical data using a multilayer perceptron (MLP) neural network designed to identify rare and procurement-critical components early in the process.

The work began by defining the scope of the problem and building data sets from five years of historical order data and their BOM lists. The data was prepared by cleaning, normalizing, one-hot encoding, and embedding so that it can be used for training and testing the neural network. An MLP neural network was developed and tested in Python, combining structured product attributes and variant codes as input data for the model. The model was optimized by adjusting hyperparameters, including embedding size, hidden layer configuration, dropout, and learning rate. The model's performance was evaluated using training and test data, and performance was measured using precision, recall, F1 score, and binary cross-entropy loss.

The study demonstrated the effectiveness of the MLP neural network in predicting BOM lists by implementing a neural network. The effectiveness of the MLP neural network was demonstrated using test cases that evaluated the model's performance in the prediction task. The results of the study show that the MLP model is suitable for the task of classifying multi-label BOM predictions. It achieves stable and reliable performance when components appear at least 40–50 times in the model training data. The model's predictions are inconsistent for very rare components that appear less than ten times in the training data. However, the model's performance improves as the training frequency increases. The conclusion of the study is that applying neural networks to BOM forecasts can improve procurement success by identifying components that are difficult to procure in advance. In the future, the model's performance could be improved by using oversampling methods and historical data from more than five years ago.

---

**KEYWORDS:** Machine Learning, Artificial Neural Network, Data preparation, Multilayer Perceptron network

---

**VAASAN YLIOPISTO****School of Technology and Innovation**

<b>Tekijä:</b>	Tomi Kallio		
<b>Tutkielman nimi:</b>	Usage of machine learning in the prediction task: Predicting rare occurring materials in product material lists		
<b>Tutkinto:</b>	Master of Science		
<b>Oppiaine:</b>	Automation and Information Technology		
<b>Työn ohjaaja:</b>	Jouni Lampinen		
<b>Vuosi:</b>	2025	<b>Sivumäärä:</b>	<b>83</b>

---

**TIIVISTELMÄ:**

Nykyaikaisessa tuotannossa tuotteiden, prosessien ja räätälöinnin kasvava monimutkaisuus on johtanut suurten tietomäärien syntymiseen. Monet yritykset kamppailevat tämän tiedon tehokkaassa hyödyntämisessä päivittäisessä toiminnassaan. Tämä haaste on läsnä myös tämän opinnäytetyön tapausyrityksessä, jossa tiettyjen komponenttien hankinnan viivästykset ovat aiheuttaneet vaikeuksia tuotannon aikataulutuksessa. Tämän ratkaisemiseksi opinnäytetyössä sovelletaan koneoppimista ja ennustavaa analyysia komponenttien esiintymisen ennustamiseen materiaaliluettelossa (BOM). Tätä varten tutkimuksessa rakennetaan ennustemalli yrityksen historiallisista tiedoista käyttämällä monikerroksista perceptron-neuroverkkoa (MLP), joka on suunniteltu tunnistamaan harvinaiset ja hankinnan kannalta kriittiset komponentit prosessin varhaisessa vaiheessa.

Työ aloitettiin määrittelemällä ongelman laajuus ja rakentamalla tietokokonaisuudet viiden vuoden historiallisista tilaustiedoista ja niiden BOM-luetteloista. Tiedot valmisteltiin puhdistamalla, normalisoimalla, one-hot-koodaamalla ja upottamalla, jotta niitä voitiin käyttää neuroverkon kouluttamiseen ja testaamiseen. MLP neuroverkko kehitettiin ja testattiin käyttäen Python ohjelmointikieltä, jossa yhdistettiin strukturoidut tuoteominaisuudet ja varianttikoodit mallin syöttötiedoiksi. Mallia optimoitiin säätämällä hyperparametreja, kuten upotuksen kokoa, piilokerroksen konfiguraatiota, pudotusta ja oppimismisnopeutta. Mallin suorituskykyä arvioitiin koulutus- ja testitietojen avulla, ja suorituskykyä mitattiin tarkkuudella, palautuksella, F1-pisteillä ja binäärisellä ristientropiahäviöllä.

Tutkimuksessa osoitettiin MLP-neuroverkon toimintakelpoisuus BOM-luetteloiden ennustamisessa toteuttamalla neuroverkko. MLP-neuroverkon tehokkuus osoitettiin testitapauksilla, joissa arvioitiin mallin suorituskykyä ennustustehtävässä. Tutkimuksen tulokset osoittavat, että MLP-malli soveltuu monimerkittyjen BOM-ennusteiden luokittelutehtävään. Se saavuttaa vakaan ja luotettavan suorituskyvyn, kun komponentit esiintyvät vähintään 40–50 kertaa mallin koulutusdatassa. Mallin ennusteet ovat epäjohdonmukaisia hyvin harvinaisille komponenteille, jotka esiintyvät alle kymmenen kertaa koulutusdatassa. Mallin suorituskyky paranee kuitenkin koulutustiheyden kasvaessa. Tutkimuksen johtopäätös on, että neuroverkkojen soveltaminen BOM-ennusteisiin voi parantaa hankintojen onnistumista tunnistamalla etukäteen hankkimisvaikeat komponentit. Tulevaisuudessa mallin suorituskykyä voitaisiin parantaa ylinäytteenottomenetelmillä ja käyttämällä historiadataa yli viiden vuoden takaa.

---

**AVAINSANAT:** Machine Learning, Artificial Neural Network, Data preparation, Multilayer Perceptron network

## Contents

1	Introduction	8
2	Practical background to the thesis	12
3	Theoretical framework	14
3.1	Data preparation for the machine learning	14
3.1.1	Data Collection	16
3.1.2	Data cleaning	17
3.1.3	Data transformation	20
3.1.4	Feature engineering	24
3.1.5	Data Reduction	25
3.2	MLP Neural Network	26
3.2.1	MLP Architecture	27
3.2.2	Activation functions	29
3.2.3	Loss function	30
3.2.4	Model optimization	32
3.2.5	Model Evaluation and Hyperparameter Optimization	34
4	Research Plan	38
4.1	Dataset creation and preparation	38
4.2	Model building	39
4.3	Model testing and hyperparameter optimization	40
5	Dataset creation for the machine learning model	43
5.1	Building the dataset	43
5.2	Historical data collection	43
5.3	Data preparation	46
6	MLP Neural Network implementation	53
6.1	Dataset Final Preparation for the ANN	54
6.2	MLP Model Hyperparameter Optimization, Training and Evaluation	58
6.2.1	MLP model architecture implementation	58
6.2.2	Hyperparameter optimization	60

6.2.3 Model training and evaluation	62
7 Conclusions	68
References	71
Appendix	78
Appendix 1. Python code for data preparation and model implementation	78

## Algorithms

Algorithm 1. SQL query for product BOMs.	45
Algorithm 2. SQL query for all variant codes of sales order positions.	46
Algorithm 3. Deletion of sales order positions where variant code 999 appears.	47
Algorithm 4. Deletion of variant codes that don't affect the structure.	48
Algorithm 5. Merging variant codes to the BoM dataset and handling empty values.	48
Algorithm 6. Basic code extraction to meaningful features.	50
Algorithm 7. A structured matrix creation of scaled numbers and one-hot encoded features.	56
Algorithm 8. Preparation of variant codes using embeddings.	57
Algorithm 9. Material codes transformation into the target matrix	57
Algorithm 10. Model structure definition.	59
Algorithm 11. Training parameters definition.	60

## Figures

Figure 1. Data cleaning activities. (Adapted from Côté et al., 2024)	19
Figure 2. Difference between semantic and syntactic transformations (Ilyas & Chu, 2019).	21
Figure 3. One-Hot encoding and Label encoding. (Adapted from Al-shehari & Alsowail, 2021)	22
Figure 4. MLP neural network architecture (Adapted from Comesaña et al., 2020).	28
Figure 5. 6-Fold Cross-validation (Jiang et al., 2020)	35
Figure 6. Top five rows from Snowflake BoM query.	46

Figure 7. The top five rows from the Snowflake variant code query.	47
Figure 8. The top five rows of the printed table after inner merging.	49
Figure 9. Material code class imbalance analysis.	51
Figure 10. Top 20 most frequent classes in material code feature.	51
Figure 11. Single-row representation of an order item with all features.	55
Figure 12. Median F1-score and IQR error bars of materials with training frequency under 100.	66

## Tables

Table 1. Confusion Matrix (Johnston & Mathur, 2019)	37
Table 2. Test plan for hyperparameter optimization.	41
Table 3. Test cases for hyperparameter optimization.	60
Table 4. Hyperparameter optimization test case results.	61
Table 5. Test results for predicting components occurring less than 100 times with different thresholds for training data.	63
Table 6. Test results with different model training parameters.	64
Table 7. Topology of final MLP model.	65
Table 8. Final training parameters of the MLP model.	65

## Abbreviations

<b>AI</b>	<b>Artificial Intelligence</b>
<b>ANN</b>	<b>Artificial Neural Network</b>
<b>AUC</b>	<b>Area under the curve</b>
<b>BCE</b>	<b>Binary cross-entropy</b>
<b>BOM</b>	<b>Bill of Material</b>
<b>CSV</b>	<b>Comma-Separated Values</b>

<b>FFANN</b>	<b>Feedforward Artificial Neural Network</b>
<b>FPR</b>	<b>False Positive Rate</b>
<b>IQR</b>	<b>Interquartile Range</b>
<b>MAE</b>	<b>Mean Absolute Error</b>
<b>ML</b>	<b>Machine Learning</b>
<b>MLB</b>	<b>Multi Label Binarizer</b>
<b>MLP</b>	<b>Multi-Layer perceptron</b>
<b>MSE</b>	<b>Mean Square Error</b>
<b>NN</b>	<b>Neural Network</b>
<b>OMS</b>	<b>Order management system</b>
<b>OTD</b>	<b>On-time Delivery</b>
<b>PCA</b>	<b>Principal component analysis</b>
<b>ReLU</b>	<b>Rectified Linear Unit</b>
<b>ROC</b>	<b>Receiver operating characteristic</b>
<b>RMSE</b>	<b>Root Mean Square Error</b>
<b>SQL</b>	<b>Structured Query Language</b>
<b>TPR</b>	<b>True Positive Rate</b>

## 1 Introduction

In modern production environments, the complexity of products and processes and the high level of customization have led to the production of huge amounts of data. Yet many companies have difficulty effectively using this data (Trattner et al., 2019; Saad et al., 2023). While automation and digital technologies contribute to data collection, companies often lack the analytical capabilities to extract working insights (Wuest et al., 2016; Wang et al., 2016). Despite the potential of machine learning and predictive analytics to optimize processes, many organizations face challenges in deploying these tools due to insufficient infrastructure and expertise (Paleyes et al., 2023).

Machine learning provides an effective approach to complex prediction tasks, as prediction is a central theme in most AI applications (Aljohani, 2023). Predictive analysis is increasingly used across industries to enhance decision-making and optimize processes (Bokonda et al., 2020). By using large amounts of historical data, machine learning models can identify potential risks more accurately and at an earlier stage compared to traditional methods (Aljohani, 2023). Learning refers to the ability of a system to recognise and understand input data and make decisions and predictions based on it (Bokonda et al., 2020).

The target company has encountered problems in the procurement process and delays in the procurement of components needed for product manufacturing. These delays often cause production delays. To avoid this, the target company must be able to predict the bill of materials (BOM) for the product to be manufactured as quickly as possible, immediately after the customer has confirmed the order, so that production can start on time. The BOM simply lists all the components needed to manufacture the final product (Cinelli et al., 2020). Difficulties in procuring components rarely arise with components that appear in product configurations, as they are very rarely procured.

BOM forecasting is particularly important for materials that rarely appear in a product's BOM list, as these rare components are often not kept in stock, and their delivery times can be long. If such materials are not identified early in the process, procurement delays can significantly disrupt the entire production schedule. In contrast, commonly used materials are usually readily available or easier to procure, which is why accurate forecasting of rare components is critical to ensuring smooth and timely manufacturing. The target company's product is highly customizable to customer needs, which means that some components may occur very rarely, only in specific product configurations.

BOM purchases often follow the company's time model, which means that purchasing activities are planned according to a predefined production or delivery schedule. This time model is typically based on standard delivery times, production cycles, and internal planning rules that help align purchases with manufacturing schedules. Time-based component procurement enables efficient production planning, optimized inventory management, and better cost control, all of which directly impact a company's ability to meet customer demands and maintain profitability (Aljohani, 2023; Cinelli et al., 2020).

This thesis was commissioned by an industrial technology company. The aim of the study is to implement a multilayer perceptron neural network that would enable the target company to predict customer order BOM lists. The prediction of BOM lists focuses on components that rarely occur in rare product configurations. The model should focus on components that have been purchased very rarely or for which there have been delays in procurement in the past. The BOM list would be predicted immediately after the orders are confirmed, based on the basic and variant codes of the ordered product. This would help the company to implement timely corrective measures in component procurement to minimize production disruptions. Supply chain efficiency is essential not only for operational reliability but also for customer satisfaction, competitive advantage, and maintaining long-term business results.

To achieve this goal, the study implements a multi-layer perceptron (MLP) neural network. Integrating machine learning models into a company's purchasing process improves its ability to predict potential future material shortages and reduces procurement delays (Aljohani, 2023). By predicting potential delays and taking corrective action early in the production process, a company is better able to meet customer expectations and maintain its competitive advantage in the market (Steinberg et al., 2023).

The MLP neural network will be trained and tested using the company's historical data to maximize its ability to predict the components needed for new orders. A key part of implementing this model is thorough data preparation and feature design. Generating BOM predictions is complex because the bill of materials includes many components with different relationships, quantities, and dependencies that vary between products and variants.

The multilayer perceptron model (MLP) was chosen for the research problem because it is efficient in processing structured data and is suitable for the multi-label classification task (Raschka & Mirjalili, 2019). An MLP neural network can learn complex and non-linear relationships between input features (Shomope et al., 2025). This is essential in this work because the input values are categorical and numerical, and they affect the outcome in complex and nonlinear ways. Using a neural network, embedded layers can be used to efficiently represent high-cardinality categorical variables, reducing dimensionality while preserving useful relationships (Hrinchuk et al., 2019).

In addition, the data often contains inconsistencies, missing values, and categorical variables that require careful cleaning and transformation. Preprocessing steps such as data cleaning, normalization, and encoding categorical variables are essential. These steps help the model understand the connections between the product base code, variant codes, and the components listed in the BOM (Abdelouahed et al., 2024). This understanding is important because the combination of base and variant codes determines which components are needed for each product configuration, directly

impacting the accuracy of the BOM prediction. Well-prepared data ensures the model can recognize patterns accurately and make reliable predictions, which is critical for effective procurement planning and reducing production delays.

The study aims to maximize the predictive power of the model and minimize the risks associated with inaccurate predictions by carefully preparing the data and selecting the most relevant features (Frye et al., 2021). The objective of the thesis is to predict the occurrence of difficult materials in the time model of the BOM of the products ordered. This objective will be achieved by answering two research questions.

RQ1: Is the performance of the MLP neural network sufficient for predicting rare components?

RQ2: How quickly does the predictive power of the model decline when the number of occurrences of components in the training data decreases?

ChatGPT, an AI-based chatbot, was used in the writing of this thesis. It supported the work by helping develop the final artifact, improving the text's structure, and understanding the overall topic. Therefore, AI was used to smooth the language and help match an academic style. The author has carefully checked that the AI did not add anything that was not originally written by the author or change the meaning of the text.

The author remains fully responsible for the content and final version of the thesis.

## 2 Practical background to the thesis

The need for the research emerged during discussions with different teams and stakeholders within the case company. The first meeting was held on 2 March 2024, where the problem was generally highlighted as the delay in orders. It was pointed out that the reason for the delay in orders is often the late start of production compared to the time schedule. The company uses a tool called Celonis to analyze data, and it was discovered that orders have been delayed and are expected to continue to be delayed due to the late start of production. Attendees included the Processes and Tools Manager, Operations Manager, and the Business Development Manager of the case company.

After the first meeting, it was clear that in most cases, the reason for the delay in starting production is the incomplete components. The components needed to manufacture the product are not available in the right places at the right time, so the production order for the ordered product does not reach VM1 in the system. The VM1 status means that the components needed to manufacture the product are available, moved to the right places, so that production can start. If the order does not reach VM1 status, production of the order cannot be started. Other crucial statuses are VM2 and VM3. The VM2 state means that the components needed to manufacture the product may have arrived but have not been moved to the correct locations, so production cannot start. If a production order has a VM3 status, then the parts to be procured have not yet arrived, so the procurement of parts has been delayed, and production cannot start. In many cases, a production order has status VM3, meaning that the materials to be procured have not arrived on time, delaying the start of production.

Discussions with different teams revealed that the time needed to respond to changes and challenges in the procurement process is too short for rarely procured parts and long-distance deliveries, making it difficult or nearly impossible for procurement teams to anticipate or respond. The products sold by the target company often have a high degree of customisation, which means that rare product configurations occur

occasionally. Rare product configurations pose problems for procurement because the components needed to produce them are ordered very rarely and in very small quantities.

The purchase request for parts is created in the background of the company's enterprise resource planning system. This happens after the design department has confirmed the BOM for the product to be ordered, i.e., the list of components needed to manufacture the product. There can often be a long time between order validation and designing, which could be used to purchase rare components. In such cases, it should be possible to predict the possible BOM of the product immediately after the order is confirmed. In this way, it would be possible to identify in the product design those parts that have traditionally been difficult to source. The BOM forecast would not necessarily lead to direct procurement actions, but order planning could be prioritised based on the forecast to identify the actual product structure at an earlier stage.

The work is limited to materials covered by operational procurement. The dataset used to train and test the model will be used to remove materials that should always be available. Also, orders that are fan motors or are not motors (e.g. certificates) are removed from the dataset. In addition, all order items that contain the variant code 999 will be removed, as they may contain any free text field information. The company's database only stores data that is five years old, so the data used in the work is limited to a five-year period.

### **3 Theoretical framework**

In this chapter, the theoretical framework for the study will be formed. It lays the foundation for predictive machine learning with historical data by introducing key concepts and methods. First, the data preparation steps are explained in detail, highlighting pre-processing techniques that are essential to ensure data quality and model performance (Badal & Sungkur, 2023). This is followed by an analysis of the different machine learning models suitable for prediction tasks and a discussion of their strengths, limitations, and the contexts in which they are most effective.

#### **3.1 Data preparation for the machine learning**

Machine learning (ML) and artificial intelligence (AI) are used for data analysis and prediction because of their ability to handle large amounts of data and complex interactions (Bokonda et al., 2020). However, many projects fail due to poor data quality, leading to poor results (Frye et al., 2021). Data is an essential part of the process of AI, especially machine learning, because training and testing a machine learning model cannot be performed without data (Abidin et al., 2020; Frye et al., 2021).

Effective data management and preparation are crucial for ensuring that the machine learning model learns patterns and relationships accurately, as poor data handling can lead to biased and flawed models (Ilyas & Rekatsinas, 2022). It is well known that the predictive power of ML algorithms can be improved by careful and high-quality implementation of the data preparation phase (Masmoudi et al., 2021). To improve the performance of machine learning models, it is important to focus more on the data used to train and test the model rather than just constantly developing new algorithms (Verdonck et al., 2024).

There is no universally accepted set of data preparation steps, but it is widely understood that data preparation involves several critical stages aimed at refining and optimizing the data. Fernandes et al, 2023 note that data preparation involves a series of processing steps, as Masmoudi et al. (2021) note that the three most used steps in data preparation are cleaning, normalization, and reduction. Schock et al, (2021) adds that in addition to the usual activities of data collection, cleaning, formatting, integration, and data reduction, data preparation also includes feature engineering.

In general, data preparation involves transforming raw and often unstructured data into a cleaned, structured format that must meet the requirements of the chosen analytical or machine-learning models (Ilyas & Rekatsinas, 2022). The process normally starts with data collection, where relevant information is collected from various sources. The objective of this step is to collect all the necessary data while ensuring that it contains the scope and objectives of the information required by the use case. However, the raw data collected is rarely complete. For example, missing values, inaccuracies, and overlaps are common and require data cleaning (Côté et al., 2024).

Once the data has been collected and cleaned, it needs to be converted into a format that can be analyzed. Data conversion helps to transform data into consistent formats and structures according to the requirements of machine learning models (Frye et al., 2021). This step often involves scaling, coding, and organizing the data to improve interpretability (Côté et al., 2024). Data reduction techniques are also often applied, especially when dealing with large data sets (Bharadiya, 2023). Data reduction aims to reduce the complexity of a dataset by focusing only on the most important features or dimensions that improve the performance of the model (Bokonda et al., 2020). Together, these data preparation steps provide a good basis for machine learning, ensuring that the data is accurate, relevant and optimised.

Feature engineering is a part of a data preprocessing process that prepares a dataset for machine learning. It aims to transform raw data into meaningful features, significantly

improving the performance of machine learning (Abdelouahed et al., 2024; Campesato, 2023; Verdonck et al., 2024). The most common steps and techniques in this process are feature transformation, creation, selection, and extraction (Verdonck et al., 2024).

### **3.1.1 Data Collection**

The primary goal of data collection is to acquire datasets suitable for training machine learning models (Roh et al., 2021). In the era of Big Data, scalable and accurate data collection techniques are critical, necessitating a comprehensive approach to data management (Roh et al., 2021) as well as for developing accurate and reliable machine-learning models (Goyal & Mahmoud, 2024; Huang & Zhao, 2024; Roh et al., 2021).

In general, there are three main approaches to data collection: the first approach is to find, extend or create new data sets. The second approach is to add individual samples to existing data sets. The third approach is to refine existing data sets or use pre-trained models for further training. (Huang & Zhao, 2024). Roughly, data acquisition includes data discovery, data augmentation, and data generation (Roh et al., 2021). Its aim is to obtain detailed information, often using historical data, in order to meet the demands of development (Huang & Zhao, 2024). Roh et al. (2021) highlight that while the process is time-consuming, it is indispensable because the performance of machine learning models heavily depends on the availability of large, high-quality datasets. Efficient data collection processes contribute to assembling comprehensive and high-quality datasets, enabling models to produce accurate predictions (Savadatti et al., 2022).

One of the steps in data acquisition is data discovery. It is a systematic process of identifying, locating, and using data sets relevant to a machine learning task (Roh et al., 2021). The information collected serves as the basis for the learning process of the machine learning algorithm and has a significant impact on its effectiveness in achieving the desired outcome (Frye et al., 2021). The challenge can be, for example, that the data comes from many different sources, such as structured or unstructured databases

(Huang & Zhao, 2024). In general, data retrieval is a complex and time-consuming process, involving manual searching of data repositories and finding relevant datasets (Roh et al., 2021). Data augmentation techniques support data discovery by extending the datasets found where necessary. They improve the datasets used by machine learning by refining existing data or integrating additional data from different sources. (Huang & Zhao, 2024; Roh et al., 2021).

Data generation is used when existing datasets do not meet the requirements of the study due to specificity, diversity, or size constraints (Huang & Zhao, 2024; Roh et al., 2021). New datasets can be created using alternative methods such as crowdsourcing or synthetic data generation. Crowdsourcing is based on manually collecting data to build datasets, while synthetic data generation is a process of creating data using algorithms that mimic real data (Roh et al., 2021). These techniques can be used to fill gaps in existing datasets to make the dataset more comprehensive (Goyal & Mahmoud, 2024).

### **3.1.2 Data cleaning**

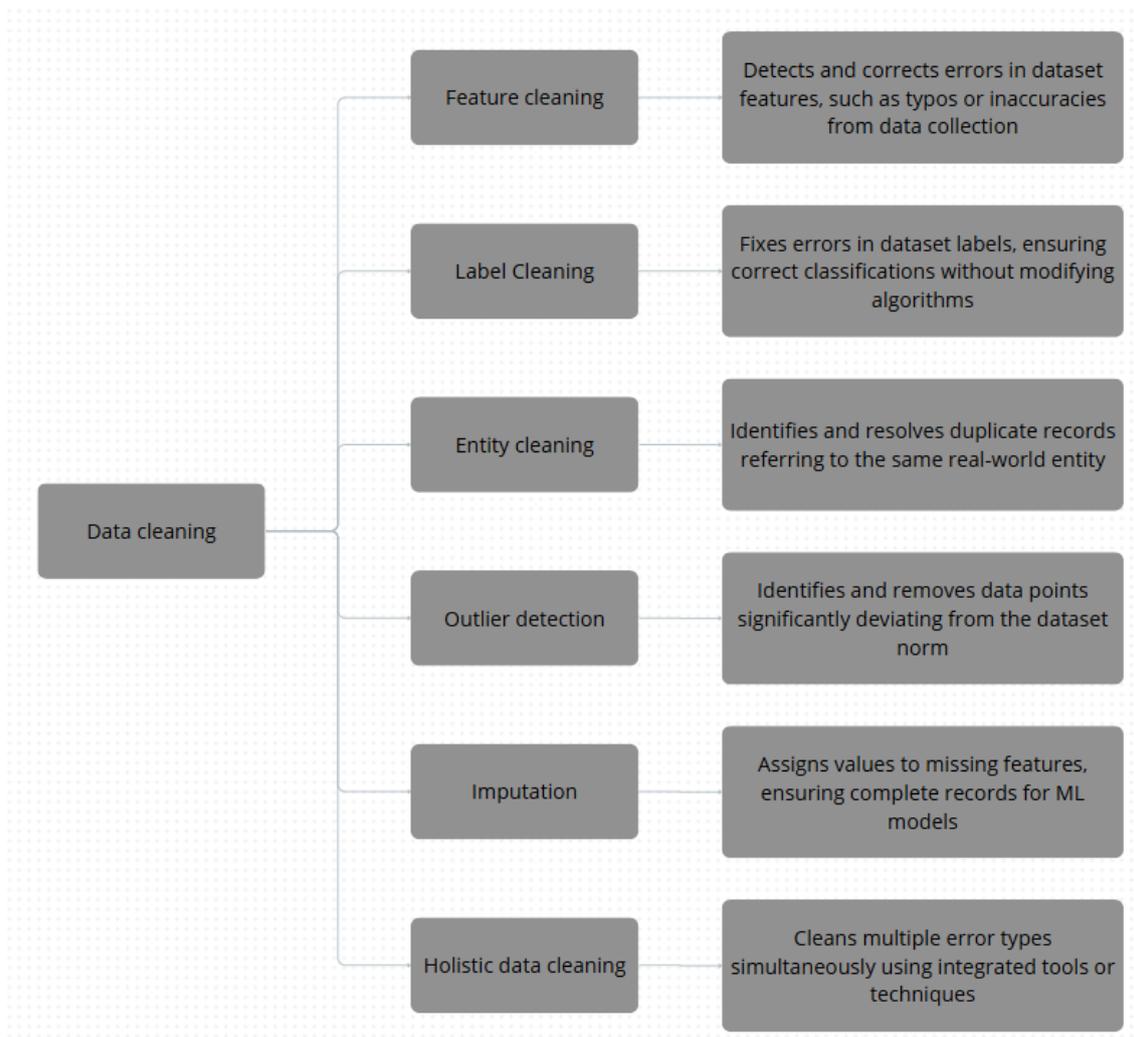
As stated earlier, data cleaning is a vital and critical step in ML projects to ensure high-quality datasets that are complete, accurate, unique, and valid (Ilyas & Chu, 2019; Ilyas & Rekatsinas, 2022; Masmoudi et al., 2021). The quality of ML applications is inherently based on the quality of the data used to train the model (Côté et al., 2024; Savadatti et al., 2022; Verdonck et al., 2024). According to a survey of IT professionals, the most common barrier to machine learning remains data quality issues. Researchers and developers still spend more than 60% of their time cleaning and organizing data (Ilyas & Chu, 2019). Data cleaning has thus been identified as a cornerstone of successful ML projects (Li et al., 2021).

Côté et al. (2024) states that data cleaning is a process of identifying and removing redundant, erroneous, noisy, or corrupt data from a dataset to enhance its quality. Zhao et al. (2024) highlight that data quality challenges have slowed the progress of Industry

4.0. Addressing data quality challenges is relevant to enabling the development of ML models and reliable data-driven analytics.

Data cleaning is generally divided into error detection and correction phases (Côté et al., 2024; Ilyas & Chu, 2019). Error detection aims to identify errors and inconsistencies in a dataset with the help of domain experts (Ilyas & Rekatsinas, 2022). Errors in data sets during data collection can include missing values, typos, inconsistent formats, or duplicate entries (Ilyas & Chu, 2019). These errors can lead to significant inaccuracies that complicate further data-based analysis and decision-making processes (Ilyas & Chu, 2019).

The correction of identified errors aims to either correct the errors or remove the incorrect records to make the dataset valid for machine learning applications (Ilyas & Chu, 2019). Missing values should always be handled, most commonly handled by imputation or deleting rows with missing values. In imputation, missing values can be replaced in two different ways. Missing values can be predicted using other features or by mean, median, or mode (Abdelouahed et al., 2024; Njeri, 2022). Data cleaning can be divided into the activities shown in Figure 1.



**Figure 1.** Data cleaning activities. (Adapted from Côté et al., 2024)

In Figure one, data cleaning has been divided into feature cleaning, label cleaning, entity cleaning, outlier detection, imputation, and holistic data cleaning (Côté et al., 2024)

Noise refers to irrelevant, incorrect, or inconsistent information in the dataset and has been identified as a critical challenge in machine learning (Côté et al., 2024). To mitigate noise, data cleaning techniques are often necessary to improve data quality before modelling. Noise is caused by qualitative problems in the dataset used for machine learning, such as mislabelled data, out-of-order values, or errors in feature values. To reduce noise, efforts have been made to develop algorithms that can handle noise without further processing of the data (Li et al., 2021). It is therefore possible to improve

ML model performance by correcting errors in the data or by using approaches such as label-noise robust learning, which allow models to maintain their performance under the influence of noise (Côté et al., 2024; Li et al., 2021). However, they do not replace the importance of data cleaning steps illustrated in Figure 1, especially if data errors significantly degrade the quality of the dataset (Côté et al., 2024). Machine learning has started to be used for data cleaning, transforming data cleaning from rule-based systems to scalable automated solutions (Ilyas & Chu, 2019; Ilyas & Rekatsinas, 2022). They can handle complex data errors such as deduplication, anomaly detection, and error correction.

### **3.1.3 Data transformation**

Data transformation is a data preparation process that transforms already cleaned data using specific protocols, programs, or scripts (Frye et al., 2021; Ilyas & Chu, 2019; Njeri, 2022). The goal is to transform data from an existing format or structure into a consistent format or structure that meets the needs of the application (Ilyas & Chu, 2019; Njeri, 2022). In addition, data transformation is sometimes seen as a data correction tool, allowing the values contained in the data to be corrected or modified according to their intended use (Ilyas & Chu, 2019).

Ilyas & Chu. (2019) and Frye et al. (2021) approach data transformation in two ways: semantic/syntactic transformations and normalization, encoding, and discretization. In the first approach, information is transformed at the meaning level (semantics) and at the structural level (syntax) (Ilyas & Chu, 2019). Figure two shows that semantic transformations seek to change information from one format to another without adding information from outside (Ilyas & Chu, 2019). Syntactic transformation, on the other hand, focuses on changing the data's structure or format without changing its meaning.

Syntactic transformation			Semantic transformation
<b>Customer</b>			<b>Country code</b>
X	Customer number: 423		FI
	Post code: 86574		DE
Y	Customer number: 422		DK
	Post code: 86543		IT
Z	Customer number: 425		
	Post code: 86234		
<hr/>			<hr/>
<b>Customer</b>	<b>Customer number</b>	<b>Post code</b>	<b>Country</b>
X	423	86574	Finland
Y	422	86543	Germany
Z	425	86234	Denmark
			Italy
<hr/>			<hr/>

**Figure 2.** Difference between semantic and syntactic transformations (Ilyas & Chu, 2019).

Another approach to data transformation includes data normalization, encoding, and discretization. Data normalization is a technique used to transform data (Frye et al., 2021; Njeri, 2022; Singh & Singh, 2020). It is used to scale and change the numerical values of raw data without changing the data itself, so that all data has the same scale (Singh & Singh, 2020). Normalization is used to equalize the effect of features in the learning process of a machine learning model so that all features affect learning equally (Singh & Singh, 2020).

Datasets often contain features that are strings or some data types other than numeric values (Cerde & Varoquaux, 2022). Machine learning algorithms work better with numerical inputs, so encoding the categorical variables of the data into a numerical form is a very important step in data transformation (Cerde & Varoquaux, 2022; Potdar et al., 2017). The traditional method for encoding categorical variables is the one-hot encoding method, which creates different binary features from the categorical feature (Cerde & Varoquaux, 2022; Ilyas & Chu, 2019; Potdar et al., 2017). One-hot encoding is simple and

widely used, but it can lead to a significant increase in dimension if the original categorical variable contains many unique categories (Cerde & Varoquaux, 2022).

Another way to encode categorical variables is label encoding, which assigns a unique number to each value in one feature when machine learning models understand the correlation between variables (Al-shehari & Alsowail, 2021). Label encoding changes categories into numbers, while one-hot encoding creates a new binary feature for every categorical value, which increases dimensionality much more (Cerde & Varoquaux, 2022). Figure three presents the difference between those encoding methods.

One-Hot Encoding				Label Encoding	
ID	Vehicle			ID	Vehicle
1	Car			1	Car
2	Bike			2	Bike
3	Motorbike			3	Motorbike
4	Car			4	Car

ID	Is_Car	Is_Bike	Is_Motorbike	ID	Vehicle
1	1	0	0	1	1
2	0	1	0	2	2
3	0	0	1	3	3
4	1	0	0	4	1

**Figure 3.** One-Hot encoding and Label encoding. (Adapted from Al-shehari & Alsowail, 2021)

High-cardinality categorical variables, however, can cause problems for encoding (Al-Shehari & Alsowail, 2021; Bolikulov et al., 2024; Cerde & Varoquaux, 2022; Pargent et al., 2022). Features, like variables with lots of unique categories, often make traditional methods such as one-hot encoding and label encoding tricky to use, due to the challenges between class relationships and the fact that the number of categories can keep growing as new data is added (Bolikulov et al., 2024; Pargent et al., 2022). Using one-hot encoding with high-cardinality features can also massively increase the number

of features, making models bigger and slower to train (Al-Shehari & Alsowail, 2021; Cerda & Varoquaux, 2022)

Bolikulov et al. (2024) and Pargent et al. (2022) studied several encoding techniques regarding machine learning models and AI models' performance in high-cardinality categorical variable encoding. Bolikulov et al. (2024) state that no single encoding method is the best, but the choice depends on the characteristics of the data. Generally, the best methods for encoding high cardinality variables are those that reduce the dimension, e.g., gray encoding and hash encoding (Bolikulov et al., 2024). However, Pargent et al. (2022) note in their research that hash encoding can reduce dimensionality but may lead to collisions, which may lead to loss of information. On the other hand, Pargent et al. (2022) note that regularized target encoding combined with cross-validation (5-fold cross-validation) was better than traditional coding methods in most data sets and machine learning models. It avoids over-adaptation and is effective in multiclass classification, regression, and binary classification problems (Pargent et al., 2022). Both studies note the ineffectiveness of the one-hot encoding method in the context of high cardinalities due to the high increase in dimensionality.

Embedding layers are used to convert high-cardinality categorical variables into dense vector representations instead of traditional one-hot encoding (Tarekegn et al., 2021; Wu et al., 2019). This technique reduces dimensionality, improves the learning efficiency of neural networks, and enables the description of relationships between different categories (Wu et al., 2019). While the one-hot encoding technique creates sparse, high-dimensional vectors, embedding layers map categorical values to a low-dimensional, continuous space (Guo & Berkhahn, 2016). Similar values are placed close together in there. Technically, the embedding layer is implemented as a matrix  $E \in R^{n \times d}$ , where  $n$  is a number of unique items, and  $d$  is the embedding dimension.  $R^{n \times d}$  represents a real value matrix with  $n$  rows and  $d$  columns, and each row in the matrix corresponds to the learned vector representation of category (Guo & Berkhahn, 2016)

The last part of the second approach to data transformation is data discretization. Data discretization refers to the process of changing information into small groups or intervals to simplify information, which improves the clarity and efficiency of models (Lin et al., 2022; Njeri, 2022; Wang et al., 2024). The goal of data discretization is to convert data into an easier-to-understand format by grouping data into intervals to eliminate complexity (Lin et al., 2022; Wang et al., 2024). The process especially improves the performance of classifiers (Wang et al., 2024). For example, age values can be grouped into meaningful categories like young, middle, and senior.

#### **3.1.4 Feature engineering**

A feature in machine learning and data analytics means a measurable characteristic of a single data point (Dong & Liu, 2018). Feature engineering is part of a data preprocessing process that prepares datasets for more efficient machine learning by creating new features or changing existing features (Carpesato, 2023; Dong & Liu, 2018; Verdonck et al., 2024). It aims to transform data into meaningful features, significantly improving the performance of machine learning (Abdelouahed et al., 2024; Carpesato, 2023; Verdonck et al., 2024). Feature engineering techniques are commonly used after data collection and cleaning. In general, missing values, errors, anomalies, and duplicates are dealt with in the data cleaning phase (Ilyas & Chu, 2019). However, some cleaning steps, such as advanced anomaly detection or imputation, are classified in some contexts as feature engineering (Verdonck et al., 2024).

The Feature engineering process usually consists of feature transformation, generation, selection, and extraction (Dong & Liu, 2018). The feature selection differs slightly from the other steps, as it only selects the most important existing features and does not create or modify the features (Abdelouahed et al., 2024; Verdonck et al., 2024). The feature selection process reduces dimension and mitigates overfitting by removing irrelevant and unnecessary data from the dataset, allowing focus only on significant features. (Dong & Liu, 2018). Unlike feature selection, feature transformation,

generation, and extraction modify existing features and create new features (Verdonck et al., 2024).

The purpose of feature transformation is to change or modify variables to a suitable presentation format for machine learning models (Abdelouahed et al., 2024; Dong & Liu, 2018; Verdonck et al., 2024). The purpose of generating features is to create new relevant features to improve the performance of machine learning models (Dong & Liu, 2018). New features are created based on existing data or expertise, highlighting relationships between variables. Feature extraction, on the other hand, aims to reduce dimensionality by creating new features by simplifying or summarizing existing features (Verdonck et al., 2024). Feature transformation addresses individual features, while the previously mentioned data transformation addresses the entire data set. Data normalization and encoding can sometimes be categorized as part of feature engineering, but in this study, they are categorized as part of data transformation, emphasizing the importance of raw data conversion for analysis (Verdonck et al., 2024). Verdonck et al. (2024) state that it is sometimes unclear which techniques directly fall under feature engineering and which do not.

### **3.1.5 Data Reduction**

In addition to cleaning the data and other preparation steps, reducing the data is a very important work step (Frye et al., 2021). It aims to reduce the amount of information while maintaining the integrity and relevance of the dataset (Bokonda et al., 2020). Efficiently implemented data reduction simplifies modelling and improves the overall performance of machine learning algorithms (Obaid et al., 2019).

Data reduction can be done by aggregating, removing unnecessary features, and reducing dimensionality (Abidin et al., 2020; Masmoudi et al., 2021). The feature selection presented below the feature engine is a simple method of reducing data by selecting only the most relevant features from the data (Abdelouahed et al., 2024;

Verdonck et al., 2024). A commonly used dimensionality reduction technique is principal component analysis (PCA). PCA is a feature extraction technique that creates new linear combinations of features, mapping them to a lower-dimensional subspace (Obaid et al., 2019). By identifying the essential components of the data set, the PCA tries to ensure the preservation of the data while reducing the number of features (Bharadiya, 2023). PCA aims to improve the efficiency of advanced analytics by minimizing computational complexity and removing noise from data (Bharadiya, 2023).

### **3.2 MLP Neural Network**

As the amount of data increases, the use of machine learning and deep learning has become increasingly common in predictive analytics (Savadatti et al., 2022). Machine learning is a sub-branch of artificial intelligence that allows different systems to learn patterns and relationships from large amounts of data to make predictions and decisions (Alpaydin, 2020; Johnston & Mathur, 2019; Verdonck et al., 2024). It is often used to analyze large data sets and make predictions.

Machine learning models include a variety of training models that understand input data, learn from it, and adapt over time (Bokonda et al., 2020). The success of the results produced by machine learning strongly depends on the quality of the information used for training (Abdelouahed et al., 2024; Fernandes et al., 2023; Njeri, 2022; Verdonck et al., 2024). Predictive analytics is a method that relies on machine learning algorithm to make accurate predictions and perform analyses (Bokonda et al., 2020; Savadatti et al., 2022). Predictive machine learning focuses on predicting future outcomes using algorithms that can analyze and learn complex patterns from historical data to generate predictions (Aljohani, 2023; Collins & Moons, 2019; Johnston & Mathur, 2019; Verdonck et al., 2024).

Neural networks are particularly effective at identifying patterns in large datasets because they can model complex dependencies and interactions between variables (Abiodun et al., 2018). Neural networks can describe complex and non-linear relationships in data, but they require more computational resources and careful tuning of hyperparameters (Chen et al., 2020). Therefore, neural networks are well-suited to problems where the relationships between features are not well defined (Abiodun et al., 2018). Neural networks can be enhanced by embedding layers, which improve the processing of highly cardinal categorical data (Hrinchuk et al., 2019). Another advantage of using a neural network is its ability to tolerate noisy data in classification (Lydia & Francis, 2020)

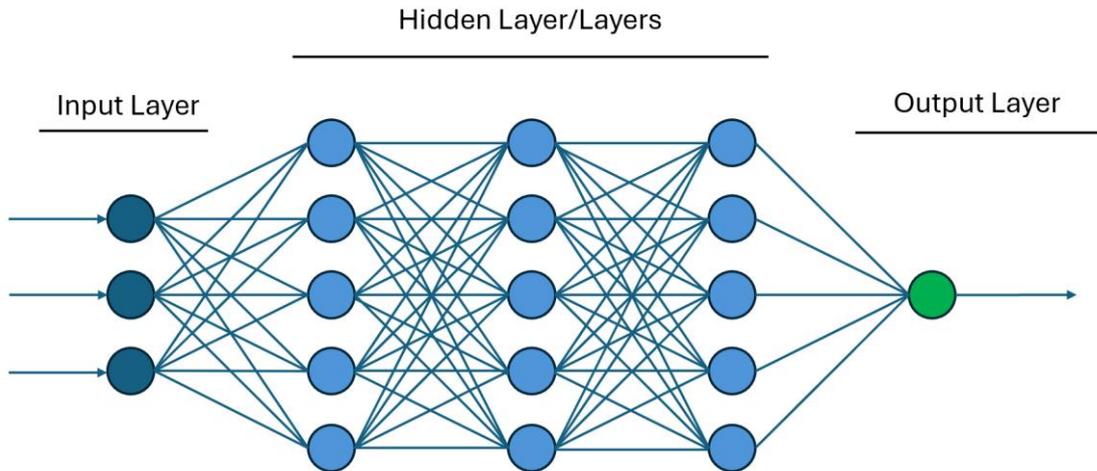
Neural networks are also particularly effective in unbalanced classification problems, as they can dynamically adjust training weights and loss functions to balance the predictions (Hrinchuk et al., 2019). Training weights refer to the specific importance given to certain samples during learning, and loss functions are mathematical formulas that measure the predictions of a model against actual values (Chen et al., 2020; Wu et al., 2019).

### **3.2.1 MLP Architecture**

MLP is a feed-forward artificial neural network (FFANN) with the ability to identify complex and nonlinear relationships (Shomope et al., 2025). The term feedforward means that each layer receives input from the previous layer, and the output works as input for the next layer (Raschka & Mirjalili, 2019). MLP includes input and output layers with one or more hidden layers with non-linear activation functions in between (Ayodele et al., 2021). Each hidden layer contains a defined set of neurons that are connected to neurons in the next layer (Lydia & Francis, 2020).

The input layer, shown in Figure four, receives structured input features and passes them through the network for processing (Comesaña et al., 2020). Hidden layers are designed

to identify relationships and complex patterns between features of input data (Lydia & Francis, 2020). The number of hidden layers and neurons determines the complexity of the neural network and the ability to identify different relationships and patterns in the data (Ayodele et al., 2021).



**Figure 4.** MLP neural network architecture (Adapted from Comesaña et al., 2020).

In FFANN, neurons in hidden layers are only connected forward to neurons in the next layer. Neurons in the same layer do not interact with each other (Shomope et al., 2025). Training a neural network requires careful parameter tuning. Parameters to be adjusted include the number of hidden layers, the number of neurons per layer, and the specification of activation functions (Comesaña et al., 2020).

The forward propagation process is used to calculate the output of the MLP (Shomope et al., 2025). It involves passing input data through the network layers to generate predictions. The process starts at the input layer, where the model receives the inputs (Comesaña et al., 2020). These inputs are multiplied by a layer-wise weight matrix, and a bias term is added. The output passes through a nonlinear activation function, after which it is passed as input to the next layer (Ertuğrul, 2018). The process is repeated

layer by layer until the final output is produced at the output layer. The process is presented in formula one:

$$\mathbf{z}_1^{(h)} = \mathbf{a}_0^{(in)} \mathbf{w}_{0,1}^{(h)} + \mathbf{a}_1^{(in)} \mathbf{w}_{1,1}^{(h)} + \dots + \mathbf{a}_m^{(in)} \mathbf{w}_{m,1}^{(h)} \quad (1)$$

The formula above calculates the net input of the first neuron in the hidden layer. In the formula,  $z_1^{(h)}$  represents the total input for the first hidden neuron (Raschka & Mirjalili, 2019). Each  $a_i^{(in)}$  is the activation value of the  $i$ th neuron in the input layer, but  $a_0^{(in)}$  is always a bias unit with value one. It allows a neuron to be activated if the weighted sum of the inputs is zero. The weights  $w_{i,1}^{(h)}$  represents the strength of the connection between input neuron  $i$  and hidden neuron one (Raschka & Mirjalili, 2019). The formula computes the weighted sum of all input activations and their weights. The result is then passed through an activation function to produce the final output of the neuron (Ertuğrul, 2018). The output is used as input for the next layer.

### 3.2.2 Activation functions

Activation functions are mathematical operations applied to the output of each neuron before moving on to the next neuron (Raschka & Mirjalili, 2019). They decide how much of the neuron's information is passed on, helping the neural network to learn complex patterns (Ertuğrul, 2018). During the training, the neural network learns to tune the weights between the interconnected input and output units to make accurate predictions from the input data (Lydia & Francis, 2020).

The activation functions used by the neural networks introduce non-linearity into the model (Ertuğrul, 2018). A multilayer neural network would behave as efficiently as a single-layer linear model if there were no non-linearity in the model. In this case, the model would only be able to learn limited complex patterns and interactions between features (Raschka & Mirjalili, 2019). By using non-linear activation functions after each

layer, the network can learn complex real-world functions and complex relationships in the data (Ertuğrul, 2018; Raschka & Mirjalili, 2019). Rectified linear unit (ReLU) is an activation function that is often used in NNs. It is a nonlinear function that is good with complex functions and relationships (Raschka & Mirjalili, 2019). ReLU is defined as:

$$f(z) = \max(0, z) \quad (2)$$

This function returns the input directly if it is positive, and zero otherwise. ReLU is quick and easy to calculate, which speeds up the learning process of the model. It also avoids problems where the model stops learning because the gradients become too small in deeper networks.

The sigmoid activation function is typically used in the output layer of models designed for multi-label classification tasks. It maps a real-valued input between zero and one, producing output values that can be interpreted as independent probabilities for each target label. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3)$$

Using the sigmoid activation function, the model can predict multiple positive labels simultaneously. A threshold value can be applied to each output neuron to define whether label is predicted as active (1) or not (0).

### 3.2.3 Loss function

Loss functions are an important part of neural network training (Lydia & Francis, 2020). They determine the difference between the predictions generated by the model and the actual target values. The loss function provides feedback to the model on how well it is

performing and how its parameters should be adjusted (Raschka & Mirjalili, 2019). For classification problems, the appropriate loss function to train the model must be chosen according to the type of problem. The problem can be binary or multiclass, and the output type can be log or probabilistic (Ertuğrul, 2018). Binary cross-entropy is a loss function for binary classification, and categorical cross-entropy is a loss function for multiclass classification (Lydia & Francis, 2020).

The binary cross-entropy quantifies the difference between two probabilities (Guo et al., 2022). The predicted value of the model is typically  $p \in (0,1)$  and the actual target label  $t \in \{0,1\}$ . The binary cross-entropy loss function is defined as:

$$\mathcal{C}(p, t) = -(t \log(p) + (1 - t) \log(1 - p)) \quad (4)$$

For a multi-label classification problem with  $n$  labels, the loss is calculated for every output node:

$$\mathcal{C}(p, t) = - \sum_{i=1}^n [t_i \log(p_i) + (1 - t_i) \log(1 - p_i)] \quad (5)$$

This loss function is ideal for neural networks with a sigmoid function in the output layer (Lydia & Francis, 2020). The binary cross-entropy loss function penalizes incorrect predictions more heavily, especially when the model is confident in a wrong answer (Raschka & Mirjalili, 2019).

### 3.2.4 Model optimization

In model optimization, the goal is to minimize loss function by adjusting weights and biases using optimization algorithms (Raschka & Mirjalili, 2019). It is typically done using an iterative process called gradient-based optimization. The model calculates the loss gradient for each weight and updates the weights in the direction that reduces the loss (Raschka & Mirjalili, 2019). The simplest form of this is gradient descent, where the rule is defined as:

$$\theta = \theta - n \times \nabla L \quad (6)$$

The model updates each parameter  $\theta$  moving it in the direction of the negative gradient of the loss function  $L$  (Raschka & Mirjalili, 2019). In the rule,  $n$  represents the learning rate controlling the size of every update step.  $\nabla L$  denotes the gradient of the loss function concerning the parameter  $\theta$  (Raschka & Mirjalili, 2019). Gradient descent is a simple and effective method in some cases, but can be inefficient with complex, high-dimensional neural networks (Rajput, 2023).

Adam (Adaptive Moment Estimation) optimizer improves basic gradient descent by combining both momentum and adaptive learning rate methods (Bowles, 2015; Comesaña et al., 2020). Adam uses exponentially decaying averages of the gradients and their squares (Raschka & Mirjalili, 2019). These are defined as:

$$\mathbf{m}_t = \beta_1 \times \mathbf{m}_{t-1} + (1 - \beta_1) \times \mathbf{g}_t \quad (7)$$

$$\mathbf{v}_t = \beta_2 \times \mathbf{v}_{t-1} + (1 - \beta_2) \times \mathbf{g}_t^2 \quad (8)$$

where  $\mathbf{m}_t$  is the first moment estimate, and  $\mathbf{v}_t$  is the second moment estimate. In the formulas,  $\mathbf{g}_t$  is the gradient at time step  $t$ , and  $\beta_1, \beta_2$  are decay rates for the moment estimates (Raschka & Mirjalili, 2019). The estimates are initialised as zero vectors and are biased towards zero in the first few steps, so Adam applies a bias correction:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{(1 - \beta_1^t)} \quad (9)$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{(1 - \beta_2^t)} \quad (10)$$

The final rule for parameter updating in Adam is:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \times \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}} \quad (11)$$

where  $\epsilon$  is a small constant added, so that division by zero is prevented. Adam optimizer is well suited for training neural networks with noisy and sparse datasets (Rajput, 2023; Raschka & Mirjalili, 2019). It is generally flexible in the choice of hyperparameters, but sometimes the learning rate must be changed from the suggested default (Raschka & Mirjalili, 2019).

In addition to the optimizer, the learning process is influenced by several training parameters, including the number of epochs, early stopping, and batch size (Rajput, 2023; Raschka & Mirjalili, 2019; Shomope et al., 2025). The epoch means a complete pass through the entire training dataset (Raschka & Mirjalili, 2019). Training of the model is normally performed over multiple epochs, so that the model has time to reduce the loss and improve its performance. The large number of epochs can lead to overfitting of the model, when performs poorly on unseen data but well on training data (Abiodun et al., 2018; Comesaña et al., 2020). Early stopping can be used to mitigate overfitting caused by a large number of epochs. It monitors the model's performance on the validation set during the training process (Raschka & Mirjalili, 2019). The training process is stopped if no improvement is observed during a predefined number of epochs.

Batch size is also an important parameter in training. It determines the number of samples to be processed before the model updates the parameters (Raschka & Mirjalili, 2019). Training the model using mini-batches replaces updating the weights after each individual sample or after a whole dataset. Using mini-batches results in faster computation and more robust learning (Rajput, 2023).

### **3.2.5 Model Evaluation and Hyperparameter Optimization**

The learning process of NNs is based on matrix algebra, so the input and output data are presented in matrix form (Raschka & Mirjalili, 2019). Each layer of the NN performs mathematical operations between the input data and the weight matrix associated with the layer (Ertuğrul, 2018; Raschka & Mirjalili, 2019). An activation function is then used before the data is passed to the next layer (Ayodele et al., 2021). Activation functions play an important role in NNs in describing the input data so that the model learns non-linear relationships (Ertuğrul, 2018). The weighted sum calculated for the neuron input is transformed by mathematical operations on the activation function before passing to the next layer (Raschka & Mirjalili, 2019). The weights determine the strength of the connection between neurons and are automatically adjusted during training to minimize prediction error.

The learning process of the machine learning model consists simply of training and testing (Alpaydin, 2020; Johnston & Mathur, 2019; Paleyes et al., 2023). The data is typically divided into training and test sets, where the quality of the data directly affects the effectiveness of the model (Nasteski, 2017). The model is fed with data from the selected dataset to learn patterns and relationships from the dataset (Paleyes et al., 2023). After training, it tries to predict the labels of the test set, and the number of incorrect predictions is calculated to evaluate the performance of the model (Nasteski, 2017).

Cross-validation and holdout are commonly used techniques in training machine learning models (Raschka, 2018). In cross-validation, k-fold cross-validation is the most widely used method, where a dataset is divided into k subsets (folds) (Jiang et al., 2020). One-fold is reserved for testing, while the remaining folds are used for training. As shown in Figure five, this process is repeated k times, with each fold serving as the test set once (Jiang et al., 2020). Cross-validation is effective in reducing overfitting but needs a lot of computational power. Holdout splits the data set once into training and testing sets (Alpaydin, 2020). The split is usually done with a weighting where 80% of the data is the training set and 20% is the testing set.

■ Training fold      ■ Testing fold

1.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6
2.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6
3.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6
4.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6
5.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6
6.	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6

**Figure 5.** 6-Fold Cross-validation (Jiang et al., 2020)

An essential part of machine learning projects is to evaluate and measure the performance of the model after training (Jiang et al., 2020; Raschka, 2018; Weerts et al., 2020). Machine learning models can be modified by adjusting hyperparameters to optimize performance, which significantly impacts the model's efficiency and accuracy (Arnold et al., 2024; Jiang et al., 2020). Hyperparameters are set before training and guide the learning process, but the model cannot change them during learning (Raschka, 2018). Parameters are values that the model learns and updates throughout training, and the final model consists of these learned parameters (Yang & Shami, 2020). Common

hyperparameters in NNs are number of hidden layers, number of neurons/layers, learning rate, dropout value and batch size (Du et al., 2021).

Hyperparameter optimization is the iterative process of selecting the best by testing several hyperparameter combinations, often through cross-validation (Morales-Hernández et al., 2023). The process starts by choosing a set of hyperparameters and assigning values to them. Next, a machine learning algorithm is trained using the selected hyperparameter values. Finally, the performance of the model is evaluated using various performance metrics. The process is repeated by changing the values of the hyperparameters until satisfactory values are reached (Morales-Hernández et al., 2023; Yang & Shami, 2020). The most common techniques for optimizing hyperparameters are Grid search, Random search, and Bayesian optimization (Arnold et al., 2024; Yang & Shami, 2020). Table one shows the different hyperparameters of general machine learning models, which can be adjusted to tune the model's learning.

In regression and classification problems, the metrics differ due to the differences in the output values (continuous and categorical values). The most used measures in regression problems are Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and  $R^2$  Score (Johnston & Mathur, 2019). When it comes to classification problems, the most used measures are Accuracy, Precision, Recall, F1-score, Receiver Operating Characteristic (ROC), and Area Under the Curve (AUC) (Bowles, 2015; Weerts et al., 2020). The F1 result measures the harmonic mean of the precision and recall values, creating a single metric. AUC is another measure created from the precision and recall values, combining the values to describe the performance of the model (Johnston & Mathur, 2019). In addition to AUC, another visualization technique is the ROC curve, which shows the ratio of True Positive Rate (TPR) and False Positive Rate (FPR) values (Bowles, 2015).

The Confusion matrix in Table one compares predicted and actual values and is used to help evaluate the performance of the classification model (Johnston & Mathur, 2019).

Accuracy calculates the proportion of correctly predicted cases, Precision measures how many actual positive cases were correctly identified as positive, and Recall indicates the proportion of predicted positive cases that were truly positive (Badal & Sungkur, 2023).

**Table 1.** Confusion Matrix (Johnston & Mathur, 2019)

	Predicted value 1	Predicted value 0
Actual value 1	False negatives	True positives
Actual value 0	True negatives	False positives

## 4 Research Plan

The work is implemented as a constructive research approach, where a practical solution based on a theoretical background is implemented in order to solve the identified practical problem.

In order to solve the problem of the case company and the study, the necessary datasets must be collected from the company's Snowflake database, after which the datasets need to be prepared for the machine learning model. The datasets must include component lists of historically ordered product configurations, order variant codes, and basic codes of ordered products. After preparing the datasets, an MLP neural network is implemented to predict the component lists of product configurations. The performance of the machine learning model is evaluated using various tests, as well as the hyperparameter optimization performed on the model.

### 4.1 Dataset creation and preparation

The case company uses the Snowflake database as its data warehouse. Therefore, all the data needed to perform the prediction task must first be retrieved from there. The historical data required includes sales order numbers and their positions, the components used to manufacture the products, product variant codes, and the basic product code. This information is stored in many different tables in the database, so the first step is to build SQL queries to retrieve the data. The queries made to the database are limited to the specific products and components. For example, all bulk materials are excluded, as they should always be available. Product types are excluded so that only normal products are processed. In addition, all products with a short delivery time, 12 days or less, are excluded from the dataset.

The final results of the queries are exported from the database as CSV files, which are further processed in Visual Studio Code. The datasets must be merged, and all order positions with a variant code value of 999 must be removed. The variant code with a value of 999 contains any free text field information, so it cannot be processed in this prediction task. The text field values contained in the dataset must be encoded for the machine learning model. Text field values that do not have high cardinality can be one-hot encoded, but classes with high cardinality are encoded into vector form (embeddings) so that the dimensionality of the dataset doesn't increase too much. The numerical values in the dataset are scaled so that the model understands their meaning.

The study analyzes the dataset used for training, and particularly the component column in the dataset. The class is analyzed based on the frequency of individual values, as the aim of this study is to exclude components that occur very frequently in product structures. Frequently occurring components don't need to be predicted, as the predictive power of the model is intended to be focused on components that occur less frequently in product structures. By deleting frequently occurring components, their dominance in model training is avoided.

## **4.2 Model building**

The built model is a multi-input neural network, which means it can handle categorical sequence data and structured table data as input. The first input branch handles sequences of product variant codes. The second input handles numeric and one-hot encoded structured features.

To input variant codes, categorical sequences must first be edited to be of uniform length. After length editing, each token must be converted into a dense vector representation for training using an embedding layer. The embedding layer allows the model to understand the relationships between the variant codes. To control the length of the vector representation, a global average pooling layer is used to transform a variable-

length vector representation into a fixed-size vector representation. The structured input contains numeric features and one-hot encoded categorical variables, which are combined with a pooled variant embedding.

The combined input is passed through a fully connected feedforward MLP neural network. The model consists of two or three hidden layers, each with a ReLU activation function and a dropout layer. The output layer has one neuron for each target label, and it uses a sigmoid activation. This allows the model the ability to predict multiple labels at once. The model is trained using Adam optimization and binary cross-entropy as the loss function.

### **4.3 Model testing and hyperparameter optimization**

The model's hyperparameters are adjusted using cross-validation. Five-fold cross-validation is used to train the model with all data and test it with varying test data. The results for each test case are the average of the results from each cross-validation round. Normal machine learning model performance metrics are used to evaluate the effectiveness of the model. Evaluating the model's performance in this thesis, precision, recall, F1 score, and binary cross-entropy loss metrics are used during hyperparameter tuning.

The MLP model's performance is tested with different combinations of hyperparameter values to first determine the appropriate hyperparameters for the model to maximize its performance. Test cases are created from combinations of hyperparameter values to be tested, which are then executed during the implementation phase. Performance metrics are used to select the appropriate hyperparameters for each case. When testing the hyperparameters of the model structure, the number of epochs is set to 80, and the early stopping method is used. Early stopping interrupts model training if the model's performance does not improve, even if the number of epochs has not yet been reached.

The batch size variable is set to 128 for the model structure hyperparameter test cases. Training parameters, such as the number of epochs and batch size, are tested with different values after testing the model structure. Table X shows the test plan for testing different hyperparameter combinations of the model. In the test plan, the MLP neural network is tested with only two hidden layers.

**Table 2.** Test plan for hyperparameter optimization.

Case ID	Embedding size	Dropout	Hidden layer 1	Hidden Layer 2	Learning rate
1	32	0.3	512	256	0.001
2	64	0.3	512	256	0.001
3	64	0.5	512	256	0.001
4	64	0.2	512	256	0.001
5	64	0.1	512	256	0.001
6	64	0.1	1024	512	0.001
7	64	0.1	512	256	0.0005
8	64	0.3	512	256	0.0005
9	128	0.1	512	256	0.001

After optimizing the hyperparameters, the model is tested with different training parameters, such as different batch sizes and numbers of epochs. In addition, the limitation of the dataset used for training the model needs to be tested. The aim of the work is to predict components that have rarely occurred in history as accurately as possible, so the dataset used for training the model should be tested in terms of the components to be included. When limiting the training dataset, the performance of the model is examined in terms of rarely occurring component classes to see if the prediction

of rarely occurring components suffers if more common components are included in the training.

The study also analyzes the created model in terms of how quickly its predictive power declines as the number of occurrences of components in the training data decreases. The analysis of the model's predictive performance to the sample size is performed for components with a maximum of 100 occurrences in the training data. This limitation is imposed because the study aims to examine the model's ability to predict components that occur infrequently in product structures.

## **5 Dataset creation for the machine learning model**

This chapter collects and prepares the dataset needed to train and test the machine learning model. Data is retrieved from the target company's database and processed into the format required by the machine learning model.

### **5.1 Building the dataset**

The first step in creating an artefact is to make the dataset needed to train and test the machine learning model. The case company has switched to using the Snowflake database as its data repository, where the necessary historical data can be found in various views. To build a training dataset for the machine learning model, two different SQL queries are run on the Snowflake views to obtain all the data needed for predictive machine learning. The materials appearing in the product structure, the base code of the ordered product, and the variant codes appearing in the order must be found in the database. In addition, the order number and item are needed to link the two tables together. The final training data must answer the question of which materials are present in the product structure based on the base code and variant codes. The final artifact implementation will be done in a Visual Studio Code project using the Python programming language, and the dataset construction and formatting will be done inside the project. The results of the queries are downloaded as CSV files to be processed in Visual Studio Code.

### **5.2 Historical data collection**

To predict accurately, the structure of a product requires careful data collection so that the model can be trained and tested with relevant information. This artifact retrieves data from the company's Snowflake database using SQL queries. First, the database is

queried for the structure of manufactured products (BoM). In the D2B - BoM view, a single BoM number contains each material appearing in the product structure as a separate row. In the D2B - Production Orders view, one production order contains one BoM number, which is used to retrieve from the D2B - BoM view the materials that appear in the structure of one manufactured product. The O2C - Sales Order Item view is also included in the query, which provides the sales order number and item used by the order management system (OMS), which are finally needed to merge the two tables. In addition, this view provides the basic code of the product. The Dimension Material and Dimension Material Plant views are added to the query to contain the master data information for the materials.

```

SELECT DISTINCT
    soi."Material Number Used by Customer" AS BasicCode,
    soi."Customer Purchase Order Number - Business Data" AS
ORDERNO,
    LTRIM(soi."Item Number of the Underlying Purchase Order
- Item Data", '0') AS Item,
    dm."Material Number"
FROM
    PROD_MO_UDM.REUSABLE_SAP_D2B."D2B - BoM" b
INNER JOIN
    PROD_MO_UDM.REUSABLE_SAP_D2B."D2B - Production Orders"
po
ON
    b."Bill of material" = po."Bill Of Material"
LEFT JOIN
    PROD_MO_UDM.REUSABLE_COMMON."Dimension Material" dm
ON
    dm."Material SID" = b."Material SID"
LEFT JOIN
    PROD_MO_UDM.REUSABLE_SAP_O2C."O2C - Sales Order Item"
soi
ON
    soi."Sales Document Number" = po."Sales Order Number" AND
soi."Sales Document Item" = po."Item Number In Sales Order"
LEFT JOIN
    PROD_MO_UDM.REUSABLE_COMMON."Dimension Material Plant"
dmp
ON
    b."Material Plant SID" = dmp."Material Plant SID"
WHERE
    dmp."Indicator Bulk Material" <> 'X' AND dmp."Plant" =
'0800' AND (soi."Material Group" = '3HC' OR soi."Material
Group" = '3HD') AND soi."Item Number of the Underlying
Purchase Order - Item Data" <> ' ' AND soi."Material Number

```

```
Used by Customer" <> ' ' AND SUBSTR(soi."Order Item Created  
On", 1, 4) >= '2020' AND dmp."Planned Delivery Time in Days" >  
12;
```

**Algorithm 1.** SQL query for product BOMs.

Only four attributes needed for prediction are selected using the SQL language's distinct command for the result of the query, which are the product basic code, material number, order number, and item. To validate the query, many different features from the joined views were needed to verify that only relevant data was processed in the final output. All irrelevant information, called noise, is attempted to be removed at this stage by filtering the database query carefully. The filter in the SQL query aims to reduce the number of materials to be processed to focus on only those relevant materials for operational purchasing.

The first database query in algorithm one has been filtered out of bulk material that should always be available. Materials master data information is used to select only materials with plant information for the target unit. The material group information in the O2C - Sales Order Item view has been used to filter only products of a specific type. In addition, all cases where the trade item number or engine base code is blank are filtered out. All 2025 sales order items will be included in the query, so that the data volume is not too large for the design of the dataset. As a last filter, only materials with a delivery time of more than 12 days in the master data are selected. To make it easier to handle the value in the future, all the prefixes have been removed from the sales order item data.

In the second database query, all the variant codes of the order items were extracted from the database in the OMS schema. Variant codes are stored in the company database in a view called F\_ORDERLINEVARIANT. In this view, each order item variant code has its own row. For example, if an order item has seven different variant codes, then there are seven rows for the order item in this view. The Dimension Order and Dimension Orderline views are included in the query to provide the OMS system order

number and item for each variant code row. Filtering in the second database query is only filtered by order year and target unit in the algorithm two. The result of the query is that only three columns are needed in the final dataset: order number, item, and variant code.

```
SELECT DISTINCT
ORDERNO,
"Order line no" AS Item,
VARIANTCODE
FROM PROD_MO_UDM.REUSABLE_OMS.F_ORDERLINEVARIANT v
inner join PROD_MO_UDM.REUSABLE_OMS."Dimension Order" o
on v.ORDERNO = o."Order no"
inner join PROD_MO_UDM.REUSABLE_OMS."Dimension Orderline" l
on o."ORDER_ID" = l."ORDER_ID"
where LEFT(l."PU", 5) = 'FIMOT' AND "Order Year" >= 2020
```

**Algorithm 2.** SQL query for all variant codes of sales order positions.

### 5.3 Data preparation

Several pre-processing steps are applied to the data collected in the previous step to create a structured and efficient dataset for the machine learning model. As a result of the previous step, two different datasets were obtained from Snowflake and uploaded as CSV files for further processing. Figures six and seven show the top five rows of both datasets to clarify their structure.

	<u>A</u> BASICCODE	<u>A</u> ORDERNO	<u>A</u> ITEM	<u>A</u> Material Number
1	3GKP082440-BDK	W14022959	1	3GZF273008-77
2	3GKP082440-BDK	W14022959	2	3GZF273008-77
3	3GKP082440-BDK	W14022959	2	3GZF334708-12
4	3GKP082440-BDK	W14022959	2	3GZF293730-13
5	3GKP082440-BDK	W14022959	2	3GZF234008-204

**Figure 6.** Top five rows from Snowflake BoM query.

	A ORDERNO	# ITEM	A VARIANTCODE
1	W14022959	2	145
2	W14022959	2	105
3	W14022959	1	754
4	W14022959	1	145
5	W14022959	1	126

**Figure 7.** The top five rows from the Snowflake variant code query.

The creation of the final dataset starts in Visual Studio Code by reading CSV files using the pandas library. Then, all order and item number combination rows are deleted where the variant code value 999 appears because they include free text information. In addition, all rows containing variant codes that do not affect the product's structure are removed from the variant code dataset. The variant codes to be removed are listed in a separate variable. Dataset loading, orders deleting and variant code deletion are done in algorithms three and four.

```
# Datasets loading from CSV files
BoM = pd.read_csv('BoM.csv')
Variants = pd.read_csv('VariantCodes.csv')

# Orders to remove where ORDERNO + ITEM combination includes
variantcode 999
orders_to_remove = Variants[Variants['VARIANTCODE'] ==
999][['ORDERNO', 'ITEM']].drop_duplicates()
variants_df = Variants[~Variants.set_index(['ORDERNO',
'ITEM']).index.isin(orders_to_remove.set_index(['ORDERNO',
'ITEM']).index)]
```

**Algorithm 3.** Deletion of sales order positions where variant code 999 appears.

```
# List of variant codes to be removed (doesn't affect the
structure of the product)
codes_to_remove = [2, 4, 24, 25, 26, 27, 49, 50, 51, 95, 96,
98, 105, 114, 115, 126, 135, 138, 139, 141, 145, 146, 147,
148, 149, 150, 159, 160, 161, 163, 168, 179, 181, 186, 208,
241, 331, 332, 333, 360, 370, 374, 417, 425, 481, 483, 484,
491, 492, 493, 494, 496, 497, 528, 529, 530, 537, 552, 560,
561, 562, 585, 586, 599, 630, 648, 675, 676, 695, 696, 760,
761, 762, 763, 764, 777, 795, 813, 816]

# Remove rows with unvalid variant codes
```

```

variants_final =
variants_df[~variants_df['VARIANTCODE'].isin(codes_to_remove)]

```

**Algorithm 4.** Deletion of variant codes that don't affect the structure.

Figures 9 and 10 are the order number and item number, which are used to combine the two datasets. Many rows were removed from the variant code dataset, but their data is still in the BoM dataset. Datasets are merged by order number and item number using an inner merge. In this case, only the rows of order number and item number combinations that appear in the variant code dataset are taken from the BoM dataset. After merging, the missing value cells in the material and variant code columns are replaced with the specified value in algorithm five.

```

# Merge the datasets based on ORDERNO and ITEM using inner
merge
merged_df = pd.merge(BoM, variant_df, on=['ORDERNO', 'ITEM'],
how='inner')

# Replace empty cells
merged_df['VARIANTCODE'] =
merged_df['VARIANTCODE'].fillna('NoVariant')
merged_df['Material Number'] = merged_df['Material
Number'].fillna('NoMaterial')

```

**Algorithm 5.** Merging variant codes to the BoM dataset and handling empty values.

Figure eight shows that there are a lot of rows after merging because of the many-to-many relationship between material codes and variant codes. In the BoM dataset, each row is uniquely identified by the order number, item, and material number. Each row is uniquely identified by the order number, item, and variant code in the variant code dataset. This means that the number of rows written for one material code equals the number of variant codes for the order item under consideration.

	BASICCODE	ORDERNO	ITEM	Material Number	VARIANTCODE
0	3GJP133280-BSK	WW1229350	1	3GZF314713-57	531
1	3GJP133280-BSK	WW1229350	1	3GZF314713-57	73
2	3GJP133280-BSK	WW1229350	1	3GZF314713-57	682
3	3GJP133280-BSK	WW1229350	1	3GZF314713-57	540
4	3GJP133280-BSK	WW1229350	1	3GZF314713-57	125

**Figure 8.** The top five rows of the printed table after inner merging.

In this data preparation step, the dataset is structured as shown in Figure eight, where each material has as many rows at the order item level as there are variant codes. Variant codes and the encoding of other features will be discussed later, as their handling depends on the machine learning model used. The number of rows can be reduced by processing variant codes so that each material code for an order item appears only in one row.

The basic code can be divided into several features because the code contains a lot of different information according to the company's specifications. Extracting new features improves the prediction of the bill of materials by increasing the number of classes that can be identified (Verdonck et al., 2024). Dividing the base code into structured parts helps the model to identify patterns more efficiently and allows for better generalizability. Ten new features are created from the basic code according to the company's specifications in algorithm six, and added as columns to the dataset in Figure eight.

```
#BASICCODE broken down into meaningful parts
final_case_df['BASICCODE'] =
final_case_df['BASICCODE'].astype(str)
final_case_df['Enclosure'] =
final_case_df['BASICCODE'].str[2]
final_case_df['Type'] = final_case_df['BASICCODE'].str[3]
final_case_df['FrameCode'] =
final_case_df['BASICCODE'].str[4:6]
final_case_df['PolePairs'] =
final_case_df['BASICCODE'].str[6]
final_case_df['FrameLength'] =
final_case_df['BASICCODE'].str[7]
```

```

final_case_df['StatorPackLength'] =
final_case_df['BASICCODE'].str[8]
final_case_df['Speed'] = final_case_df['BASICCODE'].str[9]
final_case_df['MountingArrangement'] =
final_case_df['BASICCODE'].str[11]
final_case_df['Voltage/Frequency'] =
final_case_df['BASICCODE'].str[12]
final_case_df['Design'] = final_case_df['BASICCODE'].str[13]

```

**Algorithm 6.** Basic code extraction to meaningful features.

All order positions with the variant code 999 have been removed from the dataset. In addition, all variant codes that do not affect the structure of the product have been removed. The order number and item information have also been dropped from the final data set, although they are always unique values in the background with the material code on a single line. The data has also been cleaned up a lot already in the SQL statements, where a lot of unnecessary information has been removed using delimiters. At this stage, the dataset was validated by checking that all order positions containing variant code 999 were removed from the dataset.

The processed dataset was analyzed after preparation to understand the characteristics of the dataset and the predicted values. This step is done because the dataset's characteristics influence the choice of the machine learning model. First, the number of classes in the label-encoded target column was analyzed. The dataset contains the product configurations of the target company's sales orders from 2020 onwards. The dataset contains 3,563 different material codes that the model should be able to predict for a given product configuration (basic code and variant codes). The class imbalance can be immediately observed because the column contains 595 classes with only one sample. The largest class contains 53,700 samples, making it dominant relative to the few classes. As shown in Figure nine, the median value of the classes is 17, and 2613 classes occur less than 100 times. It can also be noted that there is a significant number of classes with fewer than five or ten occurrences.

```

Total classes: 3563
Min samples: 1
Max samples: 53700
Median samples: 17.0
Classes with > 2000 samples: 187
Classes with > 1000 samples: 288
Classes with > 500 samples: 443
Classes with > 100 samples: 950
Classes with ≤ 100 samples: 2613
Classes with ≤ 20 samples: 1897
Classes with ≤ 10 samples: 1541
Classes with ≤ 5 samples: 1195
Classes with ≤ 3 samples: 987
Classes with ≤ 2 samples: 823
Classes with ≤ 1 samples: 595

```

**Figure 9.** Material code class imbalance analysis.

As seen in Figure 10, the top categories are dominant because of their number of occurrences. What is clear is that the risk of overfitting increases for dominant material code values. In turn, infrequently occurring values may pose a problem in terms of their predictability. When implementing a machine learning model, these material code column properties should be considered, as high cardinality and class imbalance are important factors for prediction accuracy (Pargent et al., 2022; Tarekegn et al., 2021).

```

Top 20 Most Frequent Classes:
Material Number  Sample Count
0      3GZF131      53700
1      3GZF121      33966
2      3GZF284708-2  32853
3      3GZE284005-1  24424
4      3GZF284716-6  20341
5      3GZF243128-5  19186
6      3GZF243228-4  19172
7      3GZF293720-149 18350
8      3GZF243028-2  18160
9      3GZF284731-3  17153
10     3GZF364030-150 16934
11     3GZF234028-319 16260
12     3GZF234009-205 15882
13     3GZF364030-210 15738
14     3GZF284720-4  15676
15     3GZF293730-14  15268
16     3GZF264730-117 14408
17     3GZF175030-1  13616
18     3GZF234096-316 13419
19     3GZF273116-29 12818

```

**Figure 10.** Top 20 most frequent classes in material code feature.

The purpose of the model is to try to predict the product structures of future order items so that difficult-to-source materials can be identified in time. As noted above, historical data on product structures of past orders contains a very large number of different material codes. At this stage, it makes sense to limit the number of material codes to be considered based on the reliability of the suppliers' on-time delivery.

All material codes with an unsatisfactory level of on-time delivery (OTD) are retrieved from the company's database. From the prepared dataset used to train the model, all rows with OTD above 95 % are removed. This reduces the number of materials to be considered for the machine learning model, which helps the model to focus only on relevant material codes.

## 6 MLP Neural Network implementation

After the data preparation phase, the next stage of the machine learning workflow involves model selection, training, and evaluation (Paleyes et al., 2023; Studer et al., 2021). Model selection is a very important step to ensure its ability to handle the characteristics of the dataset (Abbasi et al., 2022; Raschka, 2018; Savadatti et al., 2022). The goal is to develop a machine learning model that can predict the occurrence of components based on historical data using different combinations of variant codes and product basic codes. The implementation of the model requires creating a model architecture, splitting the data into training and testing sets and tuning the hyperparameters (Studer et al., 2021). After training, appropriate evaluation metrics are selected based on the task to evaluate the model's performance (Paleyes et al., 2023).

To solve the multi-label classification problem, choosing the right machine learning model is very important. The model must be able to predict the probability of occurrence of a component named as material code class based on a given combination of basic and variant codes. In this classification problem, the special features are the high cardinality variant codes in the dataset. High cardinality means that a categorical variable has a significant number of different values (Cerdeira & Varoquaux, 2022). Feature vectors for such values have many dimensions, making it difficult to describe the relationships between the values.

Another challenge with material codes is that some of them occur very rarely and others more frequently, causing imbalances within the class (Tarekegn et al., 2021). To achieve high prediction accuracy and generalizability to unseen data, it is essential to carefully select a machine learning model that can efficiently handle categorical features with high cardinality, unbalanced classes, and complex relationships between inputs (Raschka, 2018). Using embeddings, neural networks improve their ability to process categorical features, making them a valuable tool for understanding the relationships between variant codes, material codes, and basic codes.

In this work, the multilayer perceptron (MLP) architecture is used in a multi-label classification task, where several output data need to be predicted for a single input. Before training a neural network, the dataset needs to be properly prepared, as the data preprocessing requirements vary between machine learning models. In this study, the dataset has already undergone preliminary preprocessing steps, where irrelevant data has been removed based on its relevance for prediction. The resulting dataset contains the product structures of all sales orders of the target company from 2020 onwards. To train the model, the data is prepared specifically for the neural network and then divided into training and test sets. After training, the performance of the model will be evaluated using different metrics, which will be used to adjust parameters to improve the model.

### **6.1 Dataset Final Preparation for the ANN**

The first step in building the final dataset for NN is to define the frequency threshold variable. Frequency threshold variable limits how often occurring classes are included in the training of the model. This step is necessary when dealing with unbalanced data sets, and it is necessary to be able to predict rare classes. The dataset is then transformed into a format in which the order item is used to express on a single line all the product configuration information (material codes, variant codes, and extracted basic code features) that appear on it, as presented in Figure 11.

```

1 Material_labels,VariantCode_list,Enclosure,Type,FrameCode,PolePairs,FrameLength,StatorPackLength,S
2 "[2780, 2588, 2795, 2590]", "[20, 162]", "A,A,16,3,4,2,0,A,D,K
3 "[673, 1061]", "[290, 81, 288]", "B,P,35,2,2,1,0,A,D,G
4 "[2474, 345, 196, 1662, 93]", "[37, 71, 6]", "B,P,08,3,3,2,2,A,D,B
5 [473], "[35, 246, 167, 16, 89, 288, 188, 50, 37, 36, 79, 81]", "B,P,28,2,2,2,0,A,D,K
6 "[699, 298, 335]", "[35, 36, 167, 16, 81, 50, 188, 288, 246, 37, 79, 89]", "B,P,09,2,5,2,0,A,D,L
7 "[1950, 473, 1194]", "[167, 16, 89, 188, 37, 36, 79, 50, 35, 246, 288, 81]", "B,P,28,2,2,2,0,A,D,G
8 "[413, 2324]", "[180, 39, 288, 283, 290]", "B,P,35,2,2,3,0,A,E,L
9 [1921], "[288, 190]", "K,P,31,3,4,2,0,A,D,L
10 "[2795, 2944, 2780, 2590, 2656, 2919, 2928, 2716]", "[92, 165, 20]", "A,A,16,3,4,1,0,A,D,K
11 "[2308, 2481, 2246, 216, 432]", "[188, 0, 35, 190]", "K,P,10,2,5,4,0,B,D,J
12 "[2647, 2554, 1898]", "[288, 162]", "B,P,16,1,4,1,0,A,D,L

```

**Figure 11.** Single-row representation of an order item with all features.

The components with column name material codes that appear on the order item are placed in the list, as well as the variant codes. The previously created dataset format creates duplicates for material codes and variant codes. Therefore, only separate values are imported from each batch of trade values, and the data set is converted into a multi-character structure so that the multi-character classification task can be performed.

From the dataset in Figure 11, the next step is to encode the categorical features and perform feature normalization for numeric values. Categorical features have a reasonable number of unique values, so they can be encoded using the one-hot encoding method. One-hot encoding creates a column for each unique value of a feature, but the dimensionality does not increase too much when there are a reasonable number of unique values for each feature (Cerdeira & Varoquaux, 2022). Numerical values are scaled using StandardScaler from the scikit-learn library. StandardScaler changes the values of the features so that the mean of each feature is 0 and the standard deviation is 1 (Raschka & Mirjalili, 2019). This helps learning algorithms work because all values are on the same scale. A structured matrix of scaled numerical and one-hot encoded features is created in Algorithm seven, which is the second input to the future model.

```

# Numerical values of extracted basiccode features are scaled
using standard scaler
    numeric_features = ['FrameCode', 'PolePairs',
    'FrameLength', 'StatorPackLength', 'Speed']
    scaler = StandardScaler()

```

```

df_nn[numeric_features] =
scaler.fit_transform(df_nn[numeric_features])

# Identify one-hot encoded columns
onehot_features = [col for col in df_nn.columns if
any(prefix in col for prefix in [
'Enclosure_', 'Type_', 'MountingArrangement_',
'Voltage/Frequency_', 'Design_'])]

# Combine numeric and one-hot features
X_structured = df_nn[numeric_features] +
onehot_features.astype('float32').values

```

**Algorithm 7.** A structured matrix creation of scaled numbers and one-hot encoded features.

The variant codes are extracted at this stage as a list of label-encoded values for each order line. Since the number of variant codes per order line varies, these lists are padded to a fixed length using a post-padding method to keep the input dimensions of the neural network model consistent. For variant codes, one-hot encoding would increase the dimensionality of the dataset due to the high cardinality (Cerde & Varoquaux, 2022). An embedding layer is applied to the variant codes to process them efficiently. Variant codes handling is implemented in Algorithm eight.

The embedding layer represents each unique variant code in a dense, low-dimensional vector space (Hrinchuk et al., 2019). Similar variant codes are placed closer together in the vector space. The use of embedding layers reduces dimensionality and helps the model to understand the impact of variant codes on product BoMs during training (Hrinchuk et al., 2019). Each variant code in the order item list is embedded individually, after which those embeddings are averaged together using a global average pooling layer. The result of the variant code list is one feature vector of all the variant codes occurring in the list.

```

# Prepare variant code sequences and structured features
variant_lists = df_nn['VariantCode_list'].tolist()

# Split data into training and testing sets

```

```

X_var_train, X_var_test, X_struct_train, X_struct_test,
y_train, y_test = train_test_split(
    variant_lists, X_structured, y, test_size=0.2,
    random_state=42
)

# Pad variant code sequences
padded_variantcodes_train = pad_sequences(X_var_train,
padding='post')
max_len = padded_variantcodes_train.shape[1]
padded_variantcodes_test = pad_sequences(X_var_test,
maxlen=max_len, padding='post')

# Define variant vocabulary size
variant_size = max(max(seq) for seq in variant_lists if
seq) + 1
# Embeddings to represent variantcodes
variant_input = Input(shape=(max_len,))
name="variant_input")
variant_emb = Embedding(input_dim=variant_size,
output_dim=128)(variant_input)
variant_repr = GlobalAveragePooling1D()(variant_emb)

```

**Algorithm 8.** Preparation of variant codes using embeddings.

At this point, the feature vectors of the variant codes and structured features are in separate matrices. These matrices are later merged and passed through the hidden layers of the NN. The components in material code column still need to be transformed into a multi-label binary target matrix. Transforming material codes (components) into a target matrix was implemented by using a multi-label binarizer from the scikit-learn library in Algorithm nine. Each row represents which material codes are associated with each order item.

```

# Apply MultiLabelBinarizer to material labels
y_raw = df_nn['Material_labels']
mlb = MultiLabelBinarizer()
y = mlb.fit_transform(y_raw)

```

**Algorithm 9.** Material codes transformation into the target matrix

## 6.2 MLP Model Hyperparameter Optimization, Training and Evaluation

In this chapter, the final architecture of the MLP neural network is selected using various tests, and the constructed MLP model is trained. Before the model's actual training, hyperparameter optimization is performed using the test plan created in the research plan section. This is done using cross-validation to set the hyperparameters that best suit the problem for the model. After hyperparameter optimization, the model is trained, and the training parameters are tested with different values.

When training the model, it is essential to consider the imbalance of the component class which is named as material code. The impact of handling rare and frequently occurring classes needs to be evaluated regarding model performance. An iterative process of training and testing allows for testing the impact of different treatments on model performance.

The model using the MLP architecture is trained using the Adam optimizer and the BCE loss function. The parameters and strategies tested will be class frequency threshold, MLP architecture, and training parameters. The class frequency threshold is used to filter out the most common components so that the model can better predict less common components. It is especially useful for imbalanced multi-label classification tasks because the frequent classes are otherwise too dominant (Tarekegn et al., 2021). For example, rare categories can be oversampled to increase their representativeness in the training data.

### 6.2.1 MLP model architecture implementation

The final MLP model has two input matrices that combine a variant matrix and structured tabular features. The variant matrix is combined with structured data to form a single combined data set for input. It is passed through two hidden layers, using ReLU

activation to increase non-linearity. In addition, a dropout value is used after each layer to regularize and reduce overfitting. The output layer has one neuron for each target label and uses sigmoid activation to model the probability of each label independently, enabling multi-label classification. The network uses the Adam optimizer and binary cross-entropy loss for training. The structure of the model is defined in Algorithm 10.

```
# Model structure definition
    struct_input = Input(shape=(X_struct_train.shape[1],),
name="structured_input")
    x = Concatenate()([variant_repr, struct_input])
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.3)(x)
    output = Dense(y.shape[1], activation='sigmoid')(x)

    model = Model(inputs=[variant_input, struct_input],
outputs=output)
                model.compile(optimizer=Adam(0.001),
loss='binary_crossentropy',
                metrics=[Precision(), Recall()])
```

**Algorithm 10.** Model structure definition.

The size of the mini batch, the number of epochs, and the validation split are specified for training. In addition, the early stopping method is used, which interrupts training if the model's performance does not improve. The parameters for training the model are defined in Algorithm 11.

```
# Training parameters
    model.fit(
        [padded_variantcodes_train, X_struct_train], y_train,
        validation_split=0.2,
        epochs=80,
        batch_size=64,
                callbacks=[EarlyStopping(patience=3,
restore_best_weights=True)],
        verbose=2
```

)

**Algorithm 11.** Training parameters definition.

### 6.2.2 Hyperparameter optimization

Hyperparameter tuning is an important step when optimizing an NN model to find the best performance of the model in the classification task (Yang & Shami, 2020). The NN model is developed through an iterative process of testing and evaluating its performance by changing different parameters. The tested parameters include the number and the size of the hidden layers, dropout value, embedding size and learning rate of the Adam optimizer. 5-fold cross-validation is used to evaluate the performance of a model with different hyperparameters. It ensures that the results are not dependent on a particular data distribution (Raschka & Mirjalili, 2019; Weerts et al., 2020).

**Table 3.** Test cases for hyperparameter optimization.

Case ID	Embedding size	Dropout	Hidden layer 1	Hidden Layer 2	Learning rate
1	32	0.3	512	256	0.001
2	64	0.3	512	256	0.001
3	64	0.5	512	256	0.001
4	64	0.2	512	256	0.001
5	64	0.1	512	256	0.001
6	64	0.1	1024	512	0.001
7	64	0.1	512	256	0.0005
8	64	0.3	512	256	0.0005
9	128	0.1	512	256	0.001
10	128	0.3	512	256	0.001

In hyperparameter optimization, the training parameters are the number of epochs, batch size, and threshold value based on the number of occurrences of components. The number of epochs was set to 80, the batch size to 128, and the threshold to 500. The model is therefore trained using a dataset containing a maximum of 500 components that have appeared in historical data.

**Table 4.** Hyperparameter optimization test case results.

Case ID	Recall	Precision	Cross entropy loss	F1 score
1	0.785	0.862	0.0011	0.822
2	0.792	0.863	0.0010	0.826
3	0.757	0.862	0.0011	0.806
4	0.795	0.868	0.0010	0.830
5	0.802	0.868	0.0011	0.834
6	0.793	0.863	0.0011	0.827
7	0.771	0.862	0.0011	0.814
8	0.745	0.853	0.0012	0.795
9	0.804	0.872	0.0011	0.837
10	0.810	0.872	0.0010	0.840

Table four shows the results of the hyperparameter testing cases. The results are averages of the results of each cross-validation round. The results show that changing the learning rate of the Adam optimizer and increasing the dropout value had the biggest impact on the F1 score. The loss function value is low in each test case, which is good. The best results were obtained by increasing the value of the embedding size variable, which determines the length of the vector to represent each categorical token.

### 6.2.3 Model training and evaluation

In hyperparameter optimization, several different parameter combinations were tested, and the performance of the model was evaluated. The MLP architecture in this work consists of two main inputs, one for the variant code matrix and another for the structured product features. The variant codes were processed through an embedding layer, which returned a single feature vector for the order item based on the variant codes in the list. The structured input contains normalized numeric and one-hot encoded categorical product features. The structured data is combined with the feature vectors of the correct variant codes using the sample index to combine correct rows.

A significant class imbalance in the target column (material code) may lead the model to overfit based on the most frequently occurring materials in the training data. The threshold variable is defined to limit material codes by frequency of occurrence, so it should be tested with different values. The class imbalance needs to be focused to ensure that the less frequent components can be predicted. In the hyperparameter optimization, the threshold variable is set to 500 because of the dataset class analysis in Figure 12. In this case, the most dominant material code classes are excluded.

In hyperparameter optimization, several different parameter combinations were tested, and the performance of the model was evaluated. The best results were achieved in test case number ten in the table four. In the best test case, the model consists of an input layer, two hidden layers, and an output layer. The input layer of the model is connected to the first hidden layer, which has 512 neurons. The second hidden layer has 256 neurons. Both hidden layers use the ReLu activation function, and a dropout layer with a value of 0.3 is used to reduce the overfitting problem and improve generalizability. The output size of the embedding layer is 128, the learning rate of the Adam optimizer is 0.001, and binary cross-entropy is used as a loss function. The output layer uses the sigmoid activation function, which is suitable for multi-label classification tasks and assigns a value between 0 and 1 to each label (Raschka & Mirjalili, 2019).

Next, the performance of the model will be tested with different threshold values for the number of occurrences of different components. The best hyperparameter values were tested to adjust the threshold value, which determines the components included in the training data based on their number of occurrences in the historical data. It was also tested how changing the threshold value affects the prediction performance of rare components that have occurred less than 100 times. Predicting rare components is inherently difficult due to the small sample size, but testing will reveal whether adding more common material classes to the training data affects the prediction of rare components. The model's performance is evaluated only for rare components that occur less than 100 times in the training data.

**Table 5.** Test results for predicting components occurring less than 100 times with different thresholds for training data.

Threshold value	Precision	Recall	F1 score	Binary cross-entropy loss
1200	0.680	0.634	0.634	0.0009
1000	0.642	0.595	0.594	0.0010
900	0.655	0.609	0.611	0.0010
800	0.649	0.589	0.595	0.0011
700	0.670	0.617	0.622	0.0010
600	0.643	0.593	0.595	0.0011
500	0.594	0.544	0.544	0.0013
400	0.601	0.554	0.556	0.0013
300	0.576	0.541	0.538	0.0016
200	0.461	0.427	0.422	0.0019

Lowering the threshold means that rare entries are proportionally more frequent in the training set. However, lowering it also increases the difficulty of the prediction task as the amount of data decreases. As a result, the model performance is significantly

degraded, especially when the threshold is below 500. The best balance between performance and label coverage appears to be between thresholds of 700 and 900, where the model maintains relatively good performance with the rarest components. Therefore the test was continued with threshold value of 700 and test the training parameters next. Table six tests the model's performance with different batch sizes and numbers of epochs.

**Table 6.** Test results with different model training parameters.

Batch size	Epochs	Recall	Precision	F1 score	BCE loss
128	40	0.689	0.847	0.760	0.0013
64	40	0.797	0.871	0.832	0.0011
128	60	0.783	0.868	0.823	0.0011
64	60	0.828	0.880	0.853	0.0010
128	80	0.807	0.876	0.840	0.0010
64	80	0.828	0.882	0.854	0.0010
128	100	0.817	0.871	0.843	0.0010

Table six tested the training parameters, and the early stopping parameter was set to three. Early stopping interrupted training in three tests in the table six. When the batch size was 64, training stopped at epoch 57 when the maximum was 60, and at epoch 64 when the maximum was 80. When the batch size was 128, training continued longer and stopped at epoch 84, when the maximum was 100. Smaller batch sizes led to earlier stopping, and larger batches required more epochs to reach the stopping criterion.

The results in Table six show that a smaller batch size of 64 often leads to better model performance compared to a batch size of 128 in all cases. In particular, the recall and F1 scores are significantly higher, which means improved generalizability and better detection of relevant features. Increasing the number of training epochs also improves

performance up to a certain point, but the improvements begin to level off at around 80 epochs. The best overall results were achieved using a batch size of 64 and 80 training epochs, with an F1 score of 0,854 and a binary cross-entropy loss of 0,0010. The results show that a smaller batch size and moderately extended training improve the model's performance without overfitting. Table seven shows the topology of the final MLP neural network and Table eight shows final training parameters.

**Table 7.** Topology of final MLP model.

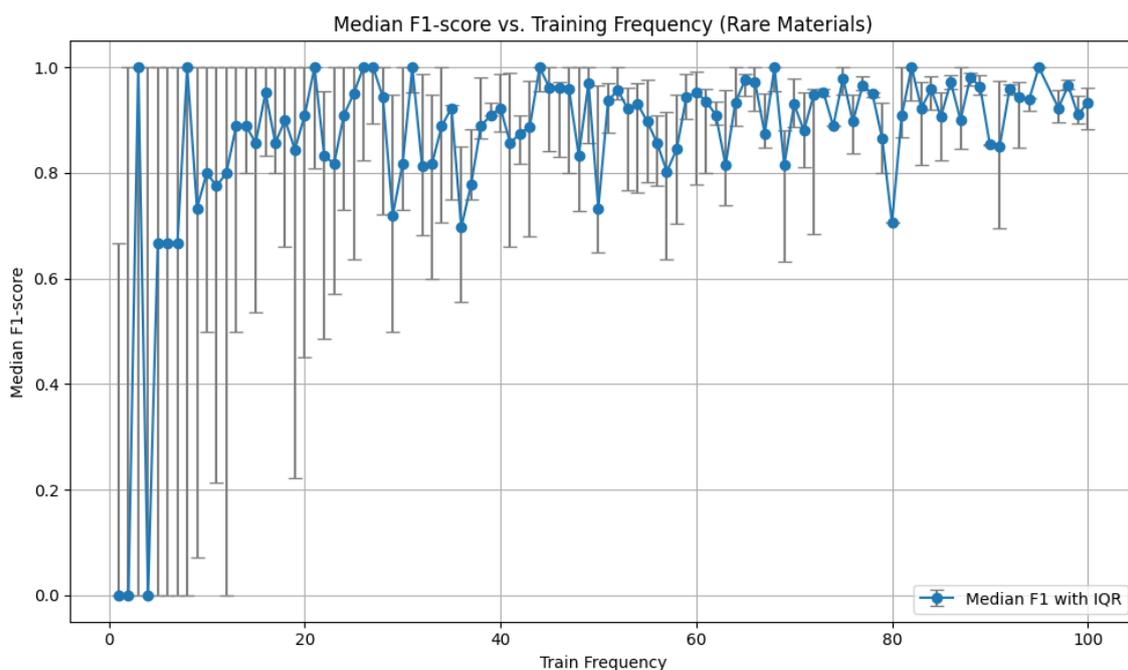
Layer	Details
Input 1	Variant sequence input
Embedding layer	Input dim = variant_size, Output dim = 128
Pooling	Global Average Pooling 1D, Creates vector representation of variants
Input 2	Structured features input
Concatenation	Merges variant representation and structured features
Hidden Layer 1	512 neurons, Activation function = ReLU
Dropout 1	Rate = 0,3
Hidden Layer 2	256 neurons, Activation function = ReLU
Dropout 2	Rate = 0,3
Output Layer	Fully connected layer with neurons equal to the number of target classes, Activation function = Sigmoid

**Table 8.** Final training parameters of the MLP model.

Parameter	Value
Optimizer	Adam optimizer, learning rate = 0,001
Loss function	Binary cross-entropy
Batch size	64
Epochs	80

Early stopping	Patience = 3, returns best weights
Validation split	0,2

In addition, the performance of the model is evaluated for the components that appeared least frequently in the training data. Components with fewer than 100 occurrences in the training data are examined. An analysis was performed to determine how many occurrences should be in the training data for the model's predictive power and stability to reach a reasonable level. The analysis is performed by training the MLP model using the topology and parameters of tables seven and eight.



**Figure 12.** Median F1-score and IQR error bars of materials with training frequency under 100.

The analysis reveals a clear relationship between label frequency and model effectiveness. In the Figure 12, the IQR (interquartile range) bars show how much the F1-score vary between the same training frequency. Longer bars mean bigger variation

in F1-scores at the same training frequency, while shorter bars mean more stable performance. Labels with fewer than 10 training examples are predicted with low and highly variable accuracy. Performance improves steadily as frequency increases, with reliable predictions where median F1 is bigger than 0,9 and low variance observed for labels appearing at least 40–50 times in the training data. This suggests that a minimum of 40 training samples per material may be needed for consistent performance.

The results show that the model is not perfect and that predicting rare components is challenging. The model already produces satisfactory median F1 results in cases where the training frequency is above ten, but there is considerable variation in the accuracy of the predictions. The results show that the rarity of components is a significant challenge for predictability, especially for very rare components, as they occur very rarely in the training data.

## 7 Conclusions

This thesis was conducted as a commission for the case company, aiming to create a machine learning model to predict the occurrence of rarely occurring components of the BOM of the products ordered. The objective was to create a MLP neural network model to predict infrequently occurring and procurement problematic parts from a delay perspective. During the work, an understanding of the type of classification task and the limitations caused by the dataset's characteristics has been developed. The historical data was carefully prepared for the machine learning model in order to ensure the performance of the model. After preparing the data, a suitable machine learning model for the multi-label classification task was implemented using a MLP neural network. The performance of the model was tested using an iterative process, in which the model was tested with different inputs and the results were used to fine-tune the model for subsequent tests.

The study was conducted using a constructive research method. The study identified a problem, explored a theory to solve it, and finally implemented and tested a solution. Using a constructive research method, a neural network was created that achieved reasonable results in evaluating performance in a highly challenging prediction task.

Two research questions were defined at the beginning of the research and answered to reach the result. The first research question, "Is the performance of the MLP neural network sufficient for predicting rare components?" was addressed by training the model with various parameters and evaluating its performance in predicting components that appeared fewer than 100 times in the training data. The results show that predicting rare components is challenging. The MLP model achieves reasonable median F1 scores when the component occurs more than ten times in the training data. However, the model's performance varies greatly if the component's frequency in the training data is very low. The model produces more reliable and consistent predictions only when the component occurs at least 40–50 times in the training data. In this case,

the median F1 scores exceed 0,9 and the variation is smaller. Based on this, the MLP model is also capable of detecting patterns for rarer components, but its performance is significantly limited for very rare components. Based on the tests, the model can be considered sufficient for rare components, but not for the rarest ones, which occur less than 10 times in the training data.

The second research question, "How quickly does the predictive power of the model decline when the number of occurrences of components in the training data decreases?" was addressed by analysing a graph depicting the median F1 score and its variation (IQR) according to the training frequency of the components. The analysis shows a clear decline in predictability as the number of occurrences in the training data decreases. F1 scores drop significantly when the number of occurrences of components in the training data is less than 10. In these cases, predictions become unreliable and unstable, and there are large differences in predictive power between classes. Between 10 and 40 occurrences, the model's performance improves, but there is still considerable variation. After 40–50 occurrences, the prediction performance of the model stabilizes. Threshold tests confirm that including more frequently occurring components in the training data may weaken the model's ability to predict rare components, as general categories start to guide learning. The threshold tests also showed that the model's performance for rare components deteriorates relatively quickly when the number of occurrences drops to a very low level.

In summary, the thesis shows that MLP neural networks can be applied reasonably well to predicting rare BOM components, but sufficient training data must be available. Predicting extremely rare components remains difficult due to the low sample size, but the study highlights how data preparation, threshold selection, and parameter optimization can improve performance in moderately rare cases. These findings provide the case company with an understanding of how prediction models can be leveraged to support procurement.

In the future, the prediction of rare components could be improved by applying oversampling methods that create artificial occurrences of the rarest classes in the training data. Such techniques would enable the model to learn more effectively from classes that currently occur too infrequently for reliable prediction. In addition, the data used in this study was limited to five years of historical data, as older data is not available from the company's database. Obtaining training data from a longer period would provide a larger sample size, which could improve the model's ability to detect patterns of rare components.

## References

- Abbasi, E., Alavi Moghaddam, M. R., & Kowsari, E. (2022). A systematic and critical review on development of machine learning based-ensemble models for prediction of adsorption process efficiency. *Journal of Cleaner Production*, 379, 134588-. <https://doi.org/10.1016/j.jclepro.2022.134588>
- Abdelouahed, S. M., Abla, R., Asmae, E., & Abdellah, A. (2024). Harnessing feature engineering to improve machine learning: A review of different data processing techniques. *2024 International Conference on Intelligent Systems and Computer Vision, ISCV 2024*. <https://doi.org/10.1109/ISCV60512.2024.10620105>
- Abidin, D. Z., Nurmaini, S., Firsandava Malik, R., Erwin, Rasywir, E., & Pratama, Y. (2020). RSSI Data Preparation for Machine Learning. *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*, 284–289. <https://doi.org/10.1109/ICIMCIS51567.2020.9354273>
- Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. E., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), 938. <https://doi.org/10.1016/J.HELIYON.2018.E00938/ASSET/85B25AED-7CEE-48F9-8F25-73F74B8D991E/MAIN.ASSETS/GR6.JPG>
- Aljohani, A. (2023). Predictive Analytics and Machine Learning for Real-Time Supply Chain Risk Mitigation and Agility. *Sustainability*, 15(20), 15088-. <https://doi.org/10.3390/su152015088>
- Alpaydin, Ethem. (2020). *Introduction to Machine Learning, Fourth Edition*. MIT Press.
- Al-shehari, T., & Alsowail, R. A. (2021). An insider data leakage detection using one-hot encoding, synthetic minority oversampling and machine learning techniques. *Entropy (Basel, Switzerland)*, 23(10), 1258-. <https://doi.org/10.3390/e23101258>
- Arnold, C., Biedebach, L., Küpfer, A., & Neunhoeffer, M. (2024). The role of hyperparameters in machine learning models and how to tune them. *Political Science Research and Methods*, 12(4), 841–848. <https://doi.org/10.1017/PSRM.2023.61>
- Ayodele, B. V., Alsaffar, M. A., Mustapa, S. I., Adesina, A., Kanthasamy, R., Witoon, T., & Abdullah, S. (2021). Process intensification of hydrogen production by catalytic

- steam methane reforming: Performance analysis of multilayer perceptron-artificial neural networks and nonlinear response surface techniques. *Process Safety and Environmental Protection*, 156, 315–329. <https://doi.org/10.1016/j.psep.2021.10.016>
- Badal, Y. T., & Sungkur, R. K. (2023). Predictive modelling and analytics of students' grades using machine learning algorithms. *Education and Information Technologies*, 28(3), 3027–3057. <https://doi.org/10.1007/s10639-022-11299-8>
- Bharadiya, J. P. (2023). A Tutorial on Principal Component Analysis for Dimensionality Reduction in Machine Learning. *International Journal of Innovative Science and Research Technology*, 8(5). [www.ijisrt.com](http://www.ijisrt.com)
- Bokonda, P. L., Ouazzani-Touhami, K., & Souissi, N. (2020). Predictive analysis using machine learning: Review of trends and methods. *2020 International Symposium on Advanced Electrical and Communication Technologies, ISAECT 2020*. <https://doi.org/10.1109/ISAECT50560.2020.9523703>
- Bolikulov, F., Nasimov, R., Rashidov, A., Akhmedov, F., & Cho, Y. I. (2024). Effective Methods of Categorical Data Encoding for Artificial Intelligence Algorithms. *Mathematics (Basel)*, 12(16), 2553-. <https://doi.org/10.3390/math12162553>
- Bowles, Michael. (2015). *Machine Learning in Python: Essential Techniques for Predictive Analysis*. John Wiley & Sons, Incorporated.
- Campeato, Oswald. (2023). *Python 3 and Feature Engineering*. Mercury Learning & Information.
- Cerda, P., & Varoquaux, G. (2022). Encoding High-Cardinality String Categorical Variables. *IEEE Transactions on Knowledge and Data Engineering*, 34(3), 1164–1176. <https://doi.org/10.1109/TKDE.2020.2992529>
- Chen, Y., Song, L., Liu, Y., Yang, L., & Li, D. (2020). A Review of the Artificial Neural Network Models for Water Quality Prediction. *Applied Sciences*, 10(17), 5776-. <https://doi.org/10.3390/app10175776>
- Collins, G. S., & Moons, K. G. M. (2019). Reporting of artificial intelligence prediction models. *The Lancet (British Edition)*, 393(10181), 1577–1579. [https://doi.org/10.1016/S0140-6736\(19\)30037-6](https://doi.org/10.1016/S0140-6736(19)30037-6)

- Comesaña, M. M., Febrero-Garrido, L., Troncoso-Pastoriza, F., & Martínez-Torres, J. (2020). Prediction of Building's Thermal Performance Using LSTM and MLP Neural Networks. *Applied Sciences*, *10*(21), 7439-. <https://doi.org/10.3390/app10217439>
- Côté, P. O., Nikanjam, A., Ahmed, N., Humeniuk, D., & Khomh, F. (2024). Data cleaning and machine learning: a systematic literature review. *Automated Software Engineering*, *31*(2), 54-. <https://doi.org/10.1007/s10515-024-00453-w>
- Dong, Guozhu., & Liu, Huan. (2018). *Feature Engineering for Machine Learning and Data Analytics*. CRC Press.
- Ertuğrul, Ö. F. (2018). A novel type of activation function in artificial neural networks: Trained activation function. *Neural Networks*, *99*, 148–157. <https://doi.org/10.1016/j.neunet.2018.01.007>
- Fernandes, A. A. A., Koehler, M., Konstantinou, N., Pankin, P., Paton, N. W., & Sakellariou, R. (2023). Data Preparation: A Technological Perspective and Review. *SN Computer Science*, *4*(4). <https://doi.org/10.1007/s42979-023-01828-8>
- Frye, M., Mohren, J., & Schmitt, R. H. (2021). Benchmarking of Data Preprocessing Methods for Machine Learning-Applications in Production. *Procedia CIRP*, *104*, 50–55. <https://doi.org/10.1016/J.PROCIR.2021.11.009>
- Goyal, M., & Mahmoud, Q. H. (2024). A Systematic Review of Synthetic Data Generation Techniques Using Generative AI. *Electronics (Basel)*, *13*(17), 3509-. <https://doi.org/10.3390/electronics13173509>
- Guo, C., & Berkahn, F. (2016). *Entity Embeddings of Categorical Variables*. <https://arxiv.org/pdf/1604.06737>
- Guo, C., Chen, X., Chen, Y., & Yu, C. (2022). Multi-Stage Attentive Network for Motion Deblurring via Binary Cross-Entropy Loss. *Entropy (Basel, Switzerland)*, *24*(10), 1414-. <https://doi.org/10.3390/e24101414>
- Hrinchuk, O., Khrulkov, V., Mirvakhabova, L., & Oseledets, I. (2019). *Tensorized Embedding Layers for Efficient Model Compression*. <https://doi.org/10.48550/arxiv.1901.10787>
- Huang, Q., & Zhao, T. (2024). *Data Collection and Labeling Techniques for Machine Learning*. <https://arxiv.org/abs/2407.12793v1>

- Ilyas, I. F. ., & Chu, Xu. (2019). *Data Cleaning*. Association for Computing Machinery and Morgan & Claypool Publishers.
- Ilyas, I. F., & Rekatsinas, T. (2022). Machine Learning and Data Cleaning: Which Serves the Other? *ACM Journal of Data and Information Quality*, 14(3). <https://doi.org/10.1145/3506712>
- Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised Machine Learning: A Brief Primer. *Behavior Therapy*, 51(5), 675–687. <https://doi.org/10.1016/j.beth.2020.05.002>
- Johnston, Benjamin., & Mathur, Ishita. (2019). *Applied Supervised Learning with Python : Use scikit-learn to build predictive models from real-world datasets and prepare yourself for the future of machine learning*. Packt Publishing.
- Li, P., Rao, X., Blase, J., Zhang, Y., Chu, X., & Zhang, C. (2021). CleanML: A study for evaluating the impact of data cleaning on ml classification tasks. *Proceedings - International Conference on Data Engineering, 2021-April*, 13–24. <https://doi.org/10.1109/ICDE51399.2021.00009>
- Lin, W. C., Tsai, C. F., & Zhong, J. R. (2022). Deep learning for missing value imputation of continuous data and the effect of data discretization. *Knowledge-Based Systems*, 239, 108079-. <https://doi.org/10.1016/j.knosys.2021.108079>
- Lydia, A. A., & Francis, F. S. (2020). Multi-Label Classification using Deep Convolutional Neural Network. *2020 International Conference on Innovative Trends in Information Technology, ICITIIT 2020*. <https://doi.org/10.1109/ICITIIT49094.2020.9071539>
- Masmoudi, O., Jaoua, M., Jaoua, A., & Yacout, S. (2021). Data Preparation in Machine Learning for Condition-based Maintenance. *Journal of Computer Science*, 17(6), 525–538. <https://doi.org/10.3844/JCSSP.2021.525.538>
- Morales-Hernández, A., Van Nieuwenhuyse, I., & Rojas Gonzalez, S. (2023). A survey on multi-objective hyperparameter optimization algorithms for machine learning. *The Artificial Intelligence Review*, 56(8), 8043–8093. <https://doi.org/10.1007/s10462-022-10359-2>
- Nasteski, V. (2017). An overview of the supervised machine learning methods. *HORIZONS.B*, 4, 51–62. <https://doi.org/10.20544/HORIZONS.B.04.1.17.P05>

- Njeri, N. R. (2022). Data Preparation For Machine Learning Modelling. *International Journal of Computer Applications Technology and Research*, 11(06), 231–235. <https://doi.org/10.7753/IJCATR1106.1008>
- Obaid, H. S., Dheyab, S. A., & Sabry, S. S. (2019). The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning. *IEMECON 2019 - 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference*, 279–283. <https://doi.org/10.1109/IEMECONX.2019.8877011>
- Paleyas, A., Urma, R. G., & Lawrence, N. D. (2023). Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Computing Surveys*, 55(6), 1–29. <https://doi.org/10.1145/3533378>
- Pargent, F., Pfisterer, F., Thomas, J., & Bischl, B. (2022). Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, 37(5), 2671–2692. <https://doi.org/10.1007/s00180-022-01207-6>
- Potdar, K., S., T., & D., C. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, 175(4), 7–9. <https://doi.org/10.5120/IJCA2017915495>
- Rajput, Vishal. (2023). *Ultimate Neural Network Programming with Python: Create Powerful Modern AI Systems by Harnessing Neural Networks with Python, Keras, and TensorFlow (English Edition)*. Orange Education Pvt Ltd.
- Raschka, S. (2018). *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. <https://doi.org/10.48550/arxiv.1811.12808>
- Raschka, S., & Mirjalili, V. (2019). *Python machine learning : machine learning and deep learning with Python, scikit-learn, and TensorFlow 2* (Third edition). Packt Publishing Ltd.
- Roh, Y., Heo, G., & Whang, S. E. (2021). A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4), 1328–1347. <https://doi.org/10.1109/TKDE.2019.2946162>

- Savadatti, M. B., Dhivya, M., Meghanashree, C., Navya, M. K., Lokesh, Y., & Kawri, N. (2022). An Overview of Predictive Analysis based on Machine learning Techniques. *Proceedings - IEEE International Conference on Advances in Computing, Communication and Applied Informatics, ACCAI 2022*. <https://doi.org/10.1109/ACCAI53970.2022.9752630>
- Schock, C., Dumler, J., & Doepper, F. (2021). Data Acquisition and Preparation - Enabling Data Analytics Projects within Production. *Procedia CIRP, 104*, 636–640. <https://doi.org/10.1016/j.procir.2021.11.107>
- Shomope, I., Tawalbeh, M., Al-Othman, A., & Almomani, F. (2025). Predicting biohydrogen production from dark fermentation of organic waste biomass using multilayer perceptron artificial neural network (MLP–ANN). *Computers and Chemical Engineering, 192*, 108900-. <https://doi.org/10.1016/j.compchemeng.2024.108900>
- Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing, 97*, 105524-. <https://doi.org/10.1016/j.asoc.2019.105524>
- Steinberg, F., Burggräf, P., Wagner, J., Heinbach, B., Saßmannshausen, T., & Brintrup, A. (2023). A novel machine learning model for predicting late supplier deliveries of low-volume-high-variety products with application in a German machinery industry. *Supply Chain Analytics, 1*, 100003. <https://doi.org/10.1016/J.SCA.2023.100003>
- Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K. R. (2021). Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *Machine Learning and Knowledge Extraction 2021, Vol. 3, Pages 392-413, 3(2)*, 392–413. <https://doi.org/10.3390/MAKE3020020>
- Tarekegn, A. N., Giacobini, M., & Michalak, K. (2021). A review of methods for imbalanced multi-label classification. *Pattern Recognition, 118*, 107965-. <https://doi.org/10.1016/j.patcog.2021.107965>
- Verdonck, T., Baesens, B., Óskarsdóttir, M., & vanden Broucke, S. (2024). Special issue on feature engineering editorial. *Machine Learning, 113(7)*, 3917–3928. <https://doi.org/10.1007/s10994-021-06042-2>

- Wang, S., Ren, J., Bai, R., Yao, Y., & Jiang, X. (2024). A Max-Relevance-Min-Divergence criterion for data discretization with applications on naive Bayes. *Pattern Recognition*, *149*, 110236-. <https://doi.org/10.1016/j.patcog.2023.110236>
- Weerts, H. J. P., Mueller, A. C., & Vanschoren, J. (2020). *Importance of Tuning Hyperparameters of Machine Learning Algorithms*. <https://arxiv.org/abs/2007.07588v1>
- Wu, L., Li, S., Hsieh, C.-J., & Sharpnack, J. (2019). *Stochastic Shared Embeddings: Data-driven Regularization of Embedding Layers*. <https://doi.org/10.48550/arxiv.1905.10630>
- Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295–316. <https://doi.org/10.1016/j.neucom.2020.07.061>

## Appendix

### Appendix 1. Python code for data preparation and model implementation

```

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder,
StandardScaler, MultiLabelBinarizer
from sklearn.model_selection import KFold, train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding,
Concatenate, Dropout, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import f1_score, recall_score,
precision_score

def prepare_nn_inputs(BoM_path="BOM2020.csv",
Variants_path="Variantti2020.csv", otd ="OTD Materials.csv",
threshold=700):
    BoM = pd.read_csv(BoM_path)
    Variants = pd.read_csv(Variants_path)

    # Remove invalid orders
    orders_to_remove = Variants[Variants['VARIANTCODE'] ==
999][['ORDERNO', 'ITEM']].drop_duplicates()
    variants_df = Variants[~Variants.set_index(['ORDERNO',
'ITEM']).index.isin(orders_to_remove.set_index(['ORDERNO',
'ITEM']).index)]

    # Remove variantcodes which doesn't affect to the product
structure
    codes_to_remove = [2, 4, 24, 25, 26, 27, 49, 50, 51, 95,
96, 98, 105, 114, 115, 126, 135, 138, 139, 141, 145, 146,
147, 148, 149, 150, 159, 160, 161, 163, 168, 179, 181, 186,
208, 241, 331, 332, 333, 360, 370, 374, 417, 425, 481, 483,
484, 491, 492, 493, 494, 496, 497, 528, 529, 530, 537, 552,
560, 561, 562, 585, 586, 599, 630, 648, 675, 676, 695, 696,
760, 761, 762, 763, 764, 777, 795, 813, 816]
    variants_final =
variants_df[~variants_df['VARIANTCODE'].isin(codes_to_remov
e)]

    # Merge datasets

```

```

merged_df = pd.merge(BoM, variants_final, on=['ORDERNO',
'ITEM'], how='inner')
merged_df['VARIANTCODE'] =
merged_df['VARIANTCODE'].fillna(-1).astype(int)
merged_df['Material Number'] = merged_df['Material
Number'].fillna('NoMaterial')

# Extract features from BASICCODE
merged_df['BASICCODE'] =
merged_df['BASICCODE'].astype(str)
merged_df['Enclosure'] = merged_df['BASICCODE'].str[2]
merged_df['Type'] = merged_df['BASICCODE'].str[3]
merged_df['FrameCode'] = merged_df['BASICCODE'].str[4:6]
merged_df['PolePairs'] = merged_df['BASICCODE'].str[6]
merged_df['FrameLength'] = merged_df['BASICCODE'].str[7]
merged_df['StatorPackLength'] =
merged_df['BASICCODE'].str[8]
merged_df['Speed'] = merged_df['BASICCODE'].str[9]
merged_df['MountingArrangement'] =
merged_df['BASICCODE'].str[11]
merged_df['Voltage/Frequency'] =
merged_df['BASICCODE'].str[12]
merged_df['Design'] = merged_df['BASICCODE'].str[13]

# Filter out high supplier OTD material codes
df_otd = supplier_OTD_calculation(otd)
df_otd.columns = df_otd.columns.str.strip()
merged_df.columns = merged_df.columns.str.strip()
merged_df = merged_df[~merged_df['Material
Number'].isin(df_otd['Material Number'])]

# Encode variant and material codes
le_variant = LabelEncoder()
merged_df['VariantCode_int'] =
le_variant.fit_transform(merged_df['VARIANTCODE'])
le_material = LabelEncoder()
merged_df['Material_encoded'] =
le_material.fit_transform(merged_df['Material Number'])

# Define frequency threshold
material_counts =
merged_df['Material_encoded'].value_counts()
materials_to_keep = material_counts[material_counts <=
threshold].index
merged_df =
merged_df[merged_df['Material_encoded'].isin(materials_to_k
eep)]

# Group to multi-label format based on ORDERNO and ITEM,
and delete duplicate values
group_cols = ['ORDERNO', 'ITEM']
material_grouped = (

```

```

        merged_df.drop_duplicates(subset=group_cols +
['Material_encoded'])
        .groupby(group_cols)['Material_encoded']
        .agg(list)
        .reset_index(name='Material_labels')
    )
    variant_grouped = (
        merged_df.drop_duplicates(subset=group_cols +
['VariantCode_int'])
        .groupby(group_cols)['VariantCode_int']
        .agg(list)
        .reset_index(name='VariantCode_list')
    )

    extracted_basiccode_features =
merged_df.drop_duplicates(subset=group_cols)[group_cols + [
        'Enclosure', 'Type', 'FrameCode', 'PolePairs',
'FrameLength', 'StatorPackLength', 'Speed',
        'MountingArrangement', 'Voltage/Frequency', 'Design']]

    # Merge grouped variants and extracted basiccode features
to grouped material codes
    df_nn = material_grouped.merge(variant_grouped,
on=group_cols).merge(extracted_basiccode_features,
on=group_cols)
    df_nn.to_csv("nn_training_data.csv", index=False)
    df_nn = df_nn.drop(columns=['ORDERNO', 'ITEM'])

    # Categorical values of extracted basiccode features are
one-hot encoded
    df_nn = pd.get_dummies(df_nn, columns=['Enclosure',
'Type', 'MountingArrangement', 'Voltage/Frequency',
'Design'])

    # Numerical values of extracted basiccode features are
scaled using standard scaler
    numeric_features = ['FrameCode', 'PolePairs',
'FrameLength', 'StatorPackLength', 'Speed']
    scaler = StandardScaler()
    df_nn[numeric_features] =
scaler.fit_transform(df_nn[numeric_features])

    return df_nn, numeric_features

def build_and_train_model1(df_nn, numeric_features):

    # Identify one-hot encoded columns
    onehot_features = [col for col in df_nn.columns if
any(prefix in col for prefix in [
        'Enclosure_', 'Type_', 'MountingArrangement_',
'Voltage/Frequency_', 'Design_'])]

    # Combine numeric and one-hot features

```

```

X_structured = df_nn[numeric_features +
onehot_features].astype('float32').values

# Apply MultiLabelBinarizer to material labels
y_raw = df_nn['Material_labels']
mlb = MultiLabelBinarizer()
y = mlb.fit_transform(y_raw)

# Prepare variant code sequences and structured features
variant_lists = df_nn['VariantCode_list'].tolist()

# Split data into training and testing sets
X_var_train, X_var_test, X_struct_train, X_struct_test,
y_train, y_test = train_test_split(
    variant_lists, X_structured, y, test_size=0.2,
    random_state=42
)

# Pad variant code sequences
padded_variantcodes_train = pad_sequences(X_var_train,
padding='post')
max_len = padded_variantcodes_train.shape[1]
padded_variantcodes_test = pad_sequences(X_var_test,
maxlen=max_len, padding='post')

# Define variant vocabulary size
variant_size = max(max(seq) for seq in variant_lists if
seq) + 1
# Embeddings to represent variantcodes
variant_input = Input(shape=(max_len,),
name="variant_input")
variant_emb = Embedding(input_dim=variant_size,
output_dim=128)(variant_input)
variant_repr = GlobalAveragePooling1D()(variant_emb)

# Model structure definition
struct_input = Input(shape=(X_struct_train.shape[1],),
name="structured_input")
x = Concatenate()([variant_repr, struct_input])
x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
output = Dense(y.shape[1], activation='sigmoid')(x)

model = Model(inputs=[variant_input, struct_input],
outputs=output)
model.compile(optimizer=Adam(0.001),
loss='binary_crossentropy',
metrics=[Precision(), Recall()])

# Training parameters
model.fit(

```

```

        [padded_variantcodes_train, X_struct_train], y_train,
        validation_split=0.2,
        epochs=80,
        batch_size=64,
        callbacks=[EarlyStopping(patience=3,
restore_best_weights=True)],
        verbose=2
    )

    # Evaluation
    results = model.evaluate([padded_variantcodes_test,
X_struct_test], y_test)

    loss = results[0]
    precision = results[1]
    recall = results[2]

    print(f"\nTest Binary Crossentropy Loss: {loss:.4f}")
    print(f"Test Precision: {precision:.4f}")
    print(f"Test Recall: {recall:.4f}")

    # Predict probabilities
    y_pred_prob = model.predict([padded_variantcodes_test,
X_struct_test])
    y_pred_bin = (y_pred_prob >= 0.5).astype(int)

    f1 = f1_score(y_test, y_pred_bin, average='micro')
    print(f"F1 Score (Micro): {f1:.4f}")

    return model

# Function to calculate supplier OTD for every delivered
material from year 2019
def supplier_OTD_calculation(otd = "OTD Materials.csv"):
    df = pd.read_csv(otd)

    df["Max GR Date"] = pd.to_datetime(df["Max GR Date"])
    df["Requested Delivery Date"] =
pd.to_datetime(df["Requested Delivery Date"])
    df["OTD"] = (df["Max GR Date"] <= df["Requested Delivery
Date"]).astype(int)

    material_otd_summary = df.groupby("Material
Number").agg(
        total_orders=("OTD", "count"),
        on_time_orders=("OTD", "sum")
    )
    material_otd_summary["OTD_percent"] = (
        material_otd_summary["on_time_orders"] /
material_otd_summary["total_orders"] * 100
    )
    material_otd_summary = material_otd_summary.reset_index()

```

```

        materials_otd =
material_otd_summary[material_otd_summary["OTD_percent"] >
95.0]
    return materials_otd

def main():
    df_nn, numeric_features = prepare_nn_inputs()
    build_and_train_model1(df_nn, numeric_features)

if __name__ == "__main__":
    main()
```