
Predictive machine learning on maintenance of Thermoplan mechanical parts

- Predict mechanical breakdown in coffee machines and relevant
spareparts -

Project Report
Group 3

Aalborg University
Department of Architecture, Design, and Media technology



Department of Architecture, Design, and
Media technology
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Predictive machine learning on maintenance of Thermoplan mechanical parts

Abstract:

Here is the abstract

Theme:

Development of predictive machine learning model on historical data

Project Period:

Spring Semester 2025

Project Group:

3

Participant(s):

Peter Gravgaard Andersen
Nanna Krogh Jensen
Lanja Mozafar Khorshid
Thomas Rye Pedersen
Kasper Ferdinand Siebrands
Mathilde Vibæk

Supervisor(s):

Andreas Møgelmoose

Page Numbers: 85

Date of Completion:

May 21, 2025

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

1	Introduction	1
1.1	About Bentax	2
1.1.1	Thermoplan	3
1.1.2	Challenges Experienced by Bentax	3
1.1.3	Use of Artificial Intelligence at Bentax	3
1.2	Initial Problem Statement	4
2	Problem Analysis	5
2.1	Maintenance and service planning	5
2.1.1	Cost of breakdowns	5
2.1.2	Part replacement	6
2.1.3	Service planning	6
2.1.4	Predictive maintenance	7
2.2	Time estimation and route optimization	8
2.2.1	Driving	8
2.2.2	A known problem	8
2.2.3	CO2	9
2.2.4	Impact On Customer	9
2.3	Data availability	10
2.3.1	Dataset Description	10
2.3.2	Data and features	10
2.3.3	Availability of data	12
2.4	State of the Art	13
2.4.1	Predictive Maintenance	13
2.4.2	Traveling Salesman Problem and Route Optimization	14
3	Problem description	15
3.1	Chosen problem	16
3.2	Final problem statement	16

4	Requirement specification	18
4.1	Use Case	18
4.2	Requirements Overview Tables	19
5	Technical overview	21
5.1	From Words to Vectors, an Introduction to Text-Based Data in Machine Learning	21
5.1.1	Tokenization: From text to Numerical Inputs	22
5.1.2	Statistical Vectorization Methods: Bag-of-Words as an example	23
5.1.3	Semantic Vector Representations: Word Embeddings	24
5.1.4	Why Represent Text as Vectors?	25
5.2	Overview of Classification Models	28
5.2.1	Random Forest Classifier	28
5.2.2	Extreme Gradient Boosting	28
5.2.3	Support Vector Classifier	29
6	Experimental phase	30
6.1	Data preprocessing for the experimental phase	30
6.1.1	Chosen data	30
6.1.2	Data Cleaning	31
6.1.3	Column Selection	31
6.1.4	Asset Filtering: Focusing on Thermoplan	31
6.1.5	Instruction Filtering and Data Reduction	32
6.1.6	Instruction Text Analysis	32
6.1.7	Distribution of Fault Types	32
6.1.8	Data Partitioning	33
6.2	Evaluation Metrics	33
6.2.1	Data Visualization	36
6.3	Methods and implementation of the experimental phase	38
6.4	Data approach	38
6.4.1	Splitting the dataset in categories	38
6.4.2	Data imputation methods	41
6.5	Text approach	43
6.5.1	TF-IDF	43
6.5.2	Sentence Transformers	47
6.5.3	LLM	50
6.6	Results Experimental Phase	59
6.6.1	Data approach	59
6.6.2	Text approach	59
6.6.3	Sentence Transformers	59
6.6.4	LLM	59
6.7	Discussion of experimental phase	60

6.7.1	data	60
6.7.2	text	60
6.7.3	conclusion of experimental phase	60
6.7.4	Choice of classifier	61
6.7.5	Choice of Classifier	61
7	Final approach	62
7.1	Design of the model	62
7.2	Data selection	63
7.2.1	Feature - Work order	63
7.2.2	Feature - Work order type	64
7.2.3	Feature - Primær Asset Produkt	64
7.2.4	Feature – Instruction	66
7.2.5	Feature – Product ID	66
7.2.6	Feature – Quantity	66
7.2.7	Min samples per part	67
7.2.8	Summary of Features	69
7.3	Modelling	70
7.3.1	evaluation	71
8	Results	72
8.1	Introduction	72
8.2	Performance Matrix	72
8.3	Analysis	74
8.4	Error Analysis	74
8.5	Visualizations	74
8.6	Evaluation	74
8.6.1	Type of Faults	74
9	Discussion	75
9.1	Discussion of results	75
9.1.1	Has the Model Leaned or Is It Just Guessing?	75
9.1.2	Strong vs. Weak Prediction Scenarios	76
9.1.3	Prediction Table	77
9.1.4	Performance Against Requirements	77
9.2	Discussion of Methods	79
9.2.1	TF-IDF	79
9.2.2	Sentence Transformer	80
9.2.3	LLM, Lanja	80
9.2.4	Discussion of relevance	81
9.3	Discussion of ethics	82

10 Conclusion	84
10.1 Introduction	84
10.2 Problem Analysis	84
10.3 Methods and Implementation	84
10.4 Final Model	84
10.5 Results	84
10.6 Requirement Specification	84
10.7	85
A Appendix	i
A.1 Bentax Service Documentation	i
A.2 Technical Theory	v

Preface

Aalborg University May 21, 2025

Here is the preface. You should put your signatures at the end of the preface.

Name
<mail>

Name
<mail>

Name
<mail>

Name
<mail>

Name
<mail>

Name
<mail>

Abbreviations

A list of the abbreviations used in this report, sorted in alphabetical order:

Chapter 1

Introduction

Efficient service and maintenance are essential for companies delivering complex technical solutions. For Bentax A/S, a Danish provider of professional coffee systems, minimizing downtime on their coffee machines is crucial for maintaining customer satisfaction and operational reliability. This is especially true for their fully automatic Thermoplan machines, which form the core of their business model. To achieve this, Bentax invests in both preventive maintenance and rapid technician support, but this approach is costly and often inefficient, as spare parts are frequently replaced without clear evidence of failure.

As the amount of maintenance data continues to grow, Bentax sees potential in leveraging machine learning to optimize their repair workflows. Specifically, there is interest in predicting which spare parts are most likely needed for a given service task, based on historical work orders and free-text descriptions of faults. Such a system could reduce the number of technician visits, lower downtime, and contribute to a more sustainable and cost-effective service process.

This project explores how historical repair data and technician-written descriptions can be used to develop a machine learning model capable of suggesting the top five most probable spare parts for a reported issue. Unlike traditional predictive maintenance, which relies on real-time sensor data, this solution focuses on what can be achieved with currently available information. The goal is to build a prototype that supports smarter service planning and lays the foundation for future, more advanced data-driven maintenance strategies.

1.1 About Bentax

Bentax is a Danish company founded in 2004 initially providing vending machines with coffee, candy and similar items. In the wake of the financial crisis, Bentax, like many others, had to reevaluate their business model and began focusing on quality-oriented customers such as hotels and cafés [bentax_growth]. A key turning point in this change, was their exclusive distribution agreement with the Swiss coffee machine manufacturer Thermoplan, which became the foundation of their new business strategy [bentax_growth]. Today, Bentax offers professional coffee machines, mainly from Thermoplan, roasts coffee in-house, and provides nationwide technical service, positioning itself as an all-around coffee solution for various industries such as hotels, restaurants, cafés, and offices, while also catering to individual consumers[bentax_roasting].

Bentax offers an all inclusive service model which includes installation, maintenance, spare parts and technical support. This is a core competitive advantage for Bentax as it offers customers full cost transparency. [bentax_roasting][bentax_guide].

The company's mission is to offer high-quality coffee solutions with a focus on sustainability and innovation, particularly through responsible supply chain management and ethical sourcing. Bentax operates by the core values of integrity, decency, and expertise. These are reflected, for example, in their fair trade sourcing, efforts to reduce CO2 emissions, respectful supplier-consumer relationships, and fast response technical service [bentax2025].

1.1.1 Thermoplan

Thermoplan is a Swiss company known for high quality, fully automatic coffee machines used by professionals worldwide. With their focus on precision engineering of their components and digital features with monitoring through ThermoplanConnect, they have become a leading name in the industry. With Thermoplan playing a key role in Bentax's business model, this project will focus specifically on optimizing service and support for these machines[Thermoplan2025].

1.1.2 Challenges Experienced by Bentax

Like any company, Bentax encounters operational challenges that impact efficiency and service quality.

A key issue lies in the maintenance of their coffee machines. The process is costly, and components are often replaced before the end of their useful life to prevent failures and downtime. While this approach can enhance short-term reliability, it leads to inefficient use of parts and undermines goals of material sustainability[BentaxProblemDescription].

Furthermore, the lack of transparency around technician arrival times leads to customer dissatisfaction and increases pressure on customer service teams. This issue is closely related to suboptimal route planning and technician dispatching. Improving these aspects could not only enhance the customer experience, but also reduce CO2 emissions [BentaxProblemDescription].

1.1.3 Use of Artificial Intelligence at Bentax

To address these challenges and strengthen their service model, Bentax has started exploring the use of artificial intelligence (AI) technologies to improve efficiency and customer satisfaction. Two promising approaches have been identified: predictive maintenance and real-time arrival estimation with route optimization [BentaxProblemDescription].

Predictive maintenance

Predictive maintenance uses historical data, real-time monitoring, and statistical models to predict when components are likely to fail. This allows maintenance to be scheduled just in time, minimizing unplanned downtime, reducing maintenance costs, and extending component lifespan.

Route optimization and real-time arrival estimation

Route optimization and real-time arrival estimation aims to improve the efficiency of technician travel planning by accounting for factors such as traffic, customer location, CO2 impact, and task urgency. When combined with real-time tracking, this can provide customers with live updates on technician arrival times, enhancing satisfaction and reducing the burden on support teams.

1.2 Initial Problem Statement

The problems and possible solutions Bentax has described lead to the initial problem statement:

How can Bentax utilize artificial intelligence to improve their efficiency and/or customer satisfaction?

Chapter 2

Problem Analysis

2.1 Maintenance and service planning

Proper maintenance and service planning are important to ensure the continuous operation of coffee machines, especially for businesses that rely on them for daily revenue. With a diverse customer base, including companies whose core business is selling coffee, minimizing machine downtime is very important for Bentax [**BentaxProblemDescription**].

2.1.1 Cost of breakdowns

When machines break down, it results in a loss not only for Bentax, which must handle repairs and reputational risks, but also for their customers. For many of these businesses, especially those whose revenue is directly tied to coffee sales, machine downtime means they are unable to serve their customers, leading to potential income loss and lower customer satisfaction. Over time, repeated interruptions in service can lower the customer's trust and loyalty, which could affect their business. Recent research highlights the important role that service quality plays in shaping customer satisfaction and loyalty, particularly within the coffee industry. Factors such as product consistency, service reliability, and availability significantly influence how customers perceive their experience. When coffee machine operations are disrupted due to technical problems or maintenance delays, it not only affects the immediate quality of the product, but also negatively impacts the customer's emotional response and their willingness to return[**ProQuest**]. Given that disruptions can have consequences for customer retention, maintenance and service planning can be viewed as a tool for not only equipment but also as a way to manage customer assets.

In addition, maintenance represents a big and often underestimated component of total operational expenditure. As highlighted in another research, maintenance-related expenses can comprise anywhere between 15 to 60% of total operational costs in both the industrial and service-oriented sectors. These costs are based upon factors such as the chosen maintenance strategy and the criticality of the equipment involved [**ScienceDirectMaintenanceCost**].

2.1.2 Part replacement

Currently, Bentax technicians often replace (multiple) machine parts as a precaution. Even if the components still are functioning properly, extended use could increase the likelihood of failure, prompting early replacement. While this approach helps prevent unexpected breakdowns, it also leads to higher maintenance costs and longer service times. Moreover, preventive maintenance can sometimes result in unnecessary part replacements, contributing to inefficiencies and resource waste [**UpkeepPreventiveMaintenance**].

Although Bentax is not alone in facing such challenges, the broader impact of unnecessary part replacements extends beyond the boundaries of the company. Inefficient use of materials, through early disposal and replacement of still-functional components, leads to increased resource extraction, energy consumption, and waste. Material use is directly linked to global greenhouse gas emissions, meaning that inefficient maintenance practices can indirectly contribute to climate change [**InterEconomics**].

Overall, these inefficiencies not only affect Bentax's own operational costs, but may also impact their customers, for example, through longer service times or, depending on the service agreements, potentially higher service costs [**BentaxProblemDescription**].

2.1.3 Service planning

Another challenge for Bentax lies in the current structure of service planning. The company experiences long feedback cycles, often up to a year, before the impact of changes in service programs becomes visible [**BentaxProblemDescription**]. Since service strategies are typically evaluated at the individual machine level, it is difficult to analyze their overall effectiveness.

As a result, decisions regarding service frequency and timing are often based on static schedules rather than actual machine conditions, which can lead to inefficient use of resources and an increased risk of unplanned downtime.

2.1.4 Predictive maintenance

To improve the issue Bentax is currently experiencing, a more data-driven approach such as predictive maintenance (PdM) could be introduced. By monitoring machine usage and performance in real-time, Bentax can detect early signs of wear or failure and adjust service schedules dynamically. This would allow for more efficient resource allocation, reduce unnecessary interventions, and shorten feedback cycles by providing continuous insight into machine health.

2.2 Time estimation and route optimization

The challenges discussed so far, particularly the costs associated with unplanned breakdowns and inefficient service cycles, are closely related to another key issue: unreliable time estimation and route optimization.

2.2.1 Driving

Every service visit requires a technician to travel to the customer, which in almost all cases involves driving. For Bentax, as noted in their internal material, one of the problems lies in how routes are planned and adjusted. The lack of route planning practices can lead to longer travel distances and inefficient sequences of visits, as highlighted in Bentax's problem description [**BentaxProblemDescription**].

The driving from A to B, as a central component of service delivery, is only one aspect of a much broader optimization problem. The effectiveness of route planning is also influenced by multiple factors, including traffic conditions, road network structure, time constraints, and environmental considerations.

2.2.2 A known problem

In scientific literature, these kinds of optimization problems are often called variants of the Vehicle Routing Problem (VRP) or the Travelling Salesman Problem (TSP). The classical TSP seeks the shortest possible route that visits each location exactly once and returns to the origin, while the VRP extends this by considering vehicle capacities, time windows, and other real-world constraints. [**Braekers2016GreenVRP**]

Additionally, on-site repairs often require extra trips because technicians do not always have the correct parts available during the first visit. Even with efficient route planning and optimization, this results in longer overall service times. The need for multiple visits not only delays the resolution of the issue but also adds to the customer's uncertainty and inconvenience [**BentaxProblemDescription**].

2.2.3 CO2

The direct consequence of more driving is a higher release of carbon dioxide (CO₂). Although CO₂ emissions are not discussed in detail in the Bentax document, the company does mention a growing focus on environmentally friendly transportation through route planning. **[BentaxProblemDescription]** Efficient routing not only reduces fuel consumption and emissions, but also contributes to improved service reliability and lower operational costs.

2.2.4 Impact On Customer

Another consequence is the direct impact on the customer experience. Delays caused by traveling routes or missing parts can result in technicians arriving late or needing multiple visits to complete a repair. This creates uncertainty regarding arrival times and increases the downtime for the customer.

Unreliable service windows can cause frustration and disrupt the customer's daily operations. For many of Bentax's clients, coffee machine availability is essential to their business **[BentaxProblemDescription]**. Any interruption in service can affect their revenue, productivity, or customer satisfaction on their side.

As noted in 2.1.1, a decrease in perceived service quality can negatively affect customer retention.

2.3 Data availability

To better understand both problems, the data provided by Bentax is analyzed to gain an overview of its structure, features, and content for subsequent analysis.

2.3.1 Dataset Description

The dataset provided by Bentax has a comprehensive range of variables related to work orders, fault classifications, spare part utilization, and both spatial and temporal attributes. Each work order in the dataset documents service activities and repairs, capturing details including service location (address, postal code, and geographical coordinates), service type (standard or urgent) and multiple time stamps (order creation, technician arrival, and completion times).

Faults are categorized according to incident types, such as equipment breakdowns and service, and are linked to specific customer assets.

Additionally, the dataset includes information on spare part usage, including product identifiers, supplier item numbers, quantities, unit prices, and warehouse origins..

2.3.2 Data and features

The dataset offers structured insights into machine failures, service workflows, and resource usage, which are key to improving troubleshooting efficiency. A confidential subset of the dataset concentrates on error diagnostics and troubleshooting procedures. It contains error codes, descriptions, suggested solutions, and IDs for replacement components. It also specifies whether a given issue can be addressed by the customer or requires intervention by a technician. The types of errors are classified into categories such as component failures, cleaning-related issues, and sensor failures.

Below in table [we need to add table numbers] is an overview of the available data:

Features	Meaning
Work Order	A unique identifier assigned to a specific work task
Street 1	Street name of the work order.
City	Work order city
Postal Code	Work order postal code.

Longitude	The longitude of the location.
Latitude	The latitude of the location.
System Status	The status of the work order in the system (e.g., "Posted", "In Progress", "Unscheduled" or "Completed").
Resource (Booking) (Bookable Resource)	The technician or employee assigned to the task and available for booking.
Created On	Date and time when the work order was created.
First Arrived On	Date and time when the first technician arrived at the location.
Completed On	Date and time when the task was completed.
Primary Incident Costumer Asset	The asset which the work order concerns
Customer Asset	The specific asset owned by the customer that needs to be repaired or serviced.
Line Status	Status of reparation
Incident Type	The category for the incident or the specific module
Primary Asset Category	Product
Primary Asset Product	Specific model-name of the asset.
Work Order Type	The type of work order (e.g., standard, urgent).
Primary Incident Type	Category of task
Instructions	Message from customer service technicians get before task
Temp Work Order Summary	Summary of work order.
Internal Note	Technician notes.
Koptæller	Number of cups produced by machine.
Service Territory	Region of task.
Product ID	Unique product ID.
Supplier Item	Supplier item ID number.
Name	Product Namm
Quantity	Number of units used
Unit Cost (Base)	Cost of unit used
Name (Warehouse)	Warehouse

2.3.3 Availability of data

It is clear that while maintenance records are available, such as the feature 'Instructions', there are no detailed machine condition data. The available records focus on service, break downs and part replacement, but do not include real-time performance metrics or specific indicators of machine health. This is further elaborated in the Problem description 3.

2.4 State of the Art

2.4.1 Predictive Maintenance

Predictive maintenance is, as stated earlier, a method designed to balance maintenance costs and operational reliability. In a research about predictive maintenance in the industry [ScienceDirectMaintenanceCost] proposed a framework that integrates machine learning algorithms for maintenance prediction. Their approach employs Random Forest and Support Vector Machines trained on historical failure data, combined with real-time monitoring. Through feature selection and anomaly detection, the framework demonstrates a reduction in unplanned downtime and improvements in maintenance scheduling and inventory management.

Although the authors of the paper [ScienceDirectMaintenanceCost] also discuss the dependency on extensive, high-quality datasets, a limitation that is present in the context of Bentax, where only historical service data without real-time IoT metrics are available, their findings indicate that historical work orders can be utilized as an initial step.

A study addressing challenges in predictive maintenance describes the use of decision trees combined with 'SHapley Additive exPlanations (SHAP)' values to make model outputs more interpretable.[ChallengesinPMD] The study highlights that providing explanations for model predictions is important for adoption by users, such as technicians, who must understand the reasoning behind suggested actions.

These studies show that maintenance strategies based on historical fault and replacement records are possible even in the absence of real-time data, but that it is difficult to achieve full predictive maintenance without additional real-time information.

2.4.2 Traveling Salesman Problem and Route Optimization

The optimization of technician routes is related to mathematical models such as the Traveling Salesman Problem and Vehicle Routing Problem (VRP). A study about the VRP [VRPstateoftheart] address VRPs with environmental objectives, aiming to minimize both cost and CO2 emissions. They apply multi objective optimization techniques and use dynamic traffic information to achieve efficient routes.

Another research, focused on UAV's [UAVstateoftheart] investigates adaptive route optimization in last-mile delivery, where real-time data updates the routes to enhance arrival time estimation. Although their case involves drones rather than technicians, the underlying concept of adapting to dynamic conditions is useful for Bentax.

A study about the multiple traveling salesman problem [mTSPstateoftheart] provides an overview of TSP and VRP research, highlighting that hybrid solution methods, such as the combination of genetic algorithms with local search, are effective for solving routing problems. The study also describes the integration of time windows and service level agreements, which introduce additional constraints into the routing problems by requiring that service be delivered within specific time intervals and according to predefined quality standards.[mTSPstateoftheart] This corresponds to Bentax's customer-related requirements, such as avoiding visits during peak operational hours and ensuring timely responses in the event of urgent breakdowns.

These studies indicate that effective route optimization in practice requires more than just minimizing travel distance. It highlights the importance of the ability to dynamically adjust routes based on real-time traffic conditions and to comply with customer-specific time constraints and service agreements.

Chapter 3

Problem description

The problems Bentax experiences are closely connected. The current approach to minimizing machine downtime is preventive replacement of parts. While this reduces the risk of failure, it also increases material usage and cost, resulting in a greater environmental impact 2.1.

Furthermore, Bentax's service plans are not planned based on their overall effectiveness, but rather on individual levels of the machines. This makes service cycles harder to assess and adapt. As a result, schedules remain static instead of being based on actual machine conditions, leading to inefficient resource use and a higher risk of unplanned downtime 2.1.3.

As addressed in 2.2, the challenges with time estimation in Bentax's data are rooted in the way maintenance and service are currently handled. Rather than focusing solely on optimizing routes or schedules, the real issue lies in ensuring that repairs and maintenance are done effectively and in time.

When maintenance is performed properly, it results in less driving partially due to returns, fewer breakdowns, lower CO2 emissions, and a improved customer satisfaction 2.2.4. Tackling the problem at its source not only improves efficiency but also significantly lowers overall costs.

3.1 Chosen problem

Given that predictive maintenance and real time-estimation is not possible at the current state of d This project takes inspiration from predictive maintenance methodologies. The objective is not to prevent breakdowns, but to improve the efficiency of the repair process after a breakdown has occurred. The focus lies on predicting the most likely spare parts required for a reported issue.

Rather than relying on real-time IoT or sensor data, which is absent in the available dataset, the approach utilizes historical repair data to support better decision making before a technician is dispatched. This can reduce unnecessary visits, minimize downtime, and lower operational costs.

As explained in subsection 2.1.2, many repairs involve replacing multiple parts. Therefore, an effective solution should be capable of recommending a set of spare parts, not just one.

The dataset includes an “Instructions” column containing the customer-reported symptoms at the time of breakdown. These descriptions, in combination with historical repair outcomes, hold potential for training a machine learning model that can suggest likely part replacements based on text-based symptom input.

The aim is to develop a product that generates a Top X list of recommended spare parts for each reported breakdown. This solution allows for smarter, data-driven service planning, reduces unnecessary travel, lowers maintenance costs, and contributes to a more sustainable service process.

As one might argue, although technicians currently perform this task intuitively, implementing a machine learning solution offers a more consistent and scalable alternative capable of reducing human error and enabling data-driven decision-making. This represents a first step in digitalizing Bentax’s maintenance processes and setting the foundation for a future solution for the initial problems provided by Bentax.

(Vi vil gerne præcisere at det er vigtigt fordi teknikerne ikke er afhængige af deres erfaring når vi bruger ML, noget med hvordan det kan skaleres)

3.2 Final problem statement

How can a machine learning model be developed to predict the top 5 most likely spare parts needed for Thermoplan coffee machine repairs based on historical data, reducing service time, lowering operational costs, and minimizing the environmental impact.

[short notes:] [the idea of including a bit of data in the problem analysis is because the limitations of Predictive maintenance (pmd) with the available data. this way we can conclude and analyse the problem and make a decision]

Chapter 4

Requirement specification

4.1 Use Case

Before beginning the development of a program or system, it makes sense to first define what is actually required. By clearly outlining the expected capabilities and functionality of the system, so the development process can be kept focused and aligned with the intended outcomes.

The following use case serves to describe the core functionality that the spare part prediction system should be able to provide. It outlines how the system is expected to interpret natural language maintenance descriptions and suggest the most likely spare parts needed.

As part of a prediction strategy for Thermoplan coffee machines, a system should be developed to automatically interpret natural language descriptions of technical problems and recommend the most appropriate spare parts. These descriptions are typically informal, written in Danish, and found in maintenance logs created by Bentax service technicians.

When a maintenance issue is described—such as “Kaffemaskinen laver ikke mælkeskum” (“The coffee machine is not producing milk foam”)—the system should be able to process the text, semantically understand the issue, and identify a ranked Top-5 list of spare parts that are most likely required to resolve it. This should be achieved by comparing the input to historical maintenance cases and known part associations.

The system should be able to intelligently handle language variation, ambiguous inputs, and non-standard phrasing by using techniques such as TF-IDF, large language models, or a hybrid approach. When applicable, it should be able to retrieve references to similar past service cases to support its recommendations.

This functionality should reduce troubleshooting time, improve part selection accuracy, and minimize machine downtime during repairs. It should support data-driven decision making, reduce the reliance on technician intuition, and lay the foundation for future integration into technician-facing platforms. In the current phase, the focus is on backend model development without a direct user interface.

4.2 Requirements Overview Tables

The following tables provide an overview of the functional and non-functional requirements described in this document.

Functional Requirements Overview

Table 4.1: Functional Requirements Overview

Requirement Name	Description	Section Reference
Natural Language Input	The system should be able to accept user queries in free-form natural language, allowing users to describe issues in ordinary terms.	Natural Language Input
Spare Part Recommendation	The system should be able to predict a Top-5 list of likely spare parts based on historical data.	Spare Part Recommendation
Semantic Text Comparison	The system should be able to compare textual information, enabling it to match queries with relevant documents.	Semantic Text Comparison
Relevant Document Retrieval	The system should be able to return the most relevant documents or records related to the query, helping identify what part solved similar issues in the past.	Relevant Case Retrieval
Danish Language Support	The system should support Danish for both input and processing.	Danish Language Support
Top-5 Prediction Handling	The system should return a ranked list of possible parts or indicate low confidence when appropriate.	Handling Ambiguity and Uncertainty

Non-Functional Requirements Overview**Table 4.2:** Non-Functional Requirements Overview

Requirement Name	Description	Section Reference
User-Friendly Output	The system should provide a structured output format that is easy to interpret.	Usability
Response Time	The system should return predictions within approximately 120 seconds or less on average hardware.	Performance and Response Time
Maintainability	The system should be designed for easy maintainability and future updates, with a modular structure and clean documentation.	Maintainability
Performance Accuracy	The system should aim to include the correct spare part in the Top-5 list in over 80% of tested cases.	Accuracy and Evaluation

Chapter 5

Technical overview

(Nanna)

This section, is valuable to review in order to give the methodological parts of the experimental phase the proper depth and understanding. The section will cover the fundamental theory that is shared across the different approaches used in the 'experimental phase', (the following section in this report), the common theoretical foundation that applies when working with text and computers.

5.1 From Words to Vectors, an Introduction to Text-Based Data in Machine Learning

When dealing with human language in machine learning (ML), one of the first challenges is that ML models require input in numeric form. In other words, models can only process numbers. Text, however, consists of sequences of words or characters. The computer can understand words, by transforming text into vectors, meaning, a mathematical representation that an algorithm can operate on. This section provides an overview of how raw text is transformed into vectors, introducing simple approaches like bag-of-words and more semantic representations like word embeddings. By balancing intuitive explanations with technical depth to be accessible to non-specialists while satisfying technically inclined readers.

5.1.1 Tokenization: From text to Numerical Inputs

The following section is primarily based on the explanations provided in the Hugging Face LLM course [huggingface2025].

Tokenization is the process of breaking text into smaller units called 'tokens' as a preliminary step toward numerical representation. Before tokenization, raw text is often normalized, removing uppercased letters, accents or extra white space, to ensure consistency. Once normalized, the text can be split into tokens. The most intuitive tokens are whole words, for example, "kaffemaskinen drypper med vand!" could be tokenized into ["kaffemaskinen", "drypper", "med", "vand", "!"]. This word-level tokenization is pretty straightforward but has its limitations. It (typically) struggles with unknown or rare words (any word not seen in training would be impossible to handle if we rely on fixed vocabulary of whole words).

An alternative is character-level tokenization, where each character is treated as a token. For instance, "bentax" becomes ["b", "e", "n", "t", "a", "x"]. Character tokenization completely avoids unknown words (since every word is composed of characters which are likely known), and it can handle languages with complex morphology or spelling variants. However, purely character-based sequences become very long and sparse vectors, making learning more difficult and computationally expensive (model has to learn patterns from scratch over a long sequence).

A popular compromise in modern NLP (natural language problem) is subword tokenization, which splits words into meaningful pieces (subwords) that are shorter than whole words but longer than single characters. The guiding principle of subword tokenization is to keep common words intact while breaking rare or complex words into constituent parts. For example, a rare word like "kaffemaskine" might be split into subwords "kaffe" and "maskine". Each subword carries meaning (so the model isn't treating each letter independently), and by combining them, the full word's meaning is preserved. This approach dramatically reduces the problem of out-of-vocabulary words. Virtually any word can be represented as a sequence of subwords. It also keeps the total number of unique tokens (the vocabulary size) at a reasonable level for the model. In practice, subword tokenization allows models to handle a rich variety of text inputs (including rare or compound words) with high efficiency and accuracy.

Modern Transformer-based language models typically (always) use subword tokenization. There are different algorithms to generate subwords, for example, 'Byte-Pair Encoding' (BPE) and 'WordPiece' are two common techniques. GPT-2 uses a byte-level BPE tokenizer, while BERT uses WordPiece. These algorithms learn a vocabulary of subword units from a large text corpus, aiming for an optimal balance between too coarse (whole word) and too fine (character-level) granularity. In practice, subword tokenization enables a model

to handle text in a flexible way: Common words stay as single tokens, while uncommon words are split into pieces that the model has seen. This yields a robust encoding scheme with few truly “unknown” tokens. Once text is tokenized (into words, characters, or subwords), each token is typically mapped to a numeric ID. We then need to convert these IDs into a numeric vector representation that a model can understand, as described in chapter 16, page 588 [**Hands-on-Machine-Learning**]

In summary, tokenization is a crucial first step that translates text into data a model can “ingest and later digest”, laying the groundwork for all subsequent vectorization steps in the ML pipeline. How the model ‘digest’ these words, highly depends on which methods applied. The next subsections will explain Statistical vectorization and semantic vectorization.

5.1.2 Statistical Vectorization Methods: Bag-of-Words as an example

Statistical vectorization methods, are methods that transform documents into numerical vectors, based on the statistical properties of word occurrences. This approach assumes that each word in a document contributes independently to its meaning, allowing a document to be represented as a vector in a high dimensional space. A simple and illustrative example is the Bag-of-Words (BoW) method. BoW builds a vocabulary from the corpus and represents each document by counting how many times each word from the vocabulary appears in it. The result is a sparse vector where most elements are zero, especially in large corpora. However, BoW method has key limitations. It treats all words as equally important, ignoring how frequently they appear across the corpus, which can reduce the effectiveness of the representation. Additionally, BoW fails to capture the semantic meaning of words or their order in a sentence, leading to a loss of contextual information. Furthermore, the model produces very high-dimensional and sparse vectors, a problem known as the “curse of dimensionality” which can pose significant challenges for many machine learning algorithms, as explained in chapter 8 page 238 [**Hands-on-Machine-Learning**]. Despite these limitations, BoW is an important stepping stone toward more refined vectorization techniques. One such refinement is TF-IDF, which adjust BoW counts to reflect the uniqueness of terms, because it considers the frequency of each words as it importance, therefore the name “Term Frequency-Inverse Document Frequency”. TF-IDF plays a central role in our implementation and is explained in more detail in the method and implementation section (see section TF-IDF Approach).

Statistical vectorization differ fundamentally from semantic vector representations like word embeddings, which attempt to capture the contextual or relational meaning of words. While statistical methods rely on frequency-based signals, embeddings are typically learned from large corpora using predictive models and capture more nuanced relationships be-

tween words.

5.1.3 Semantic Vector Representations: Word Embeddings

In machine learning, embedding refers to a process of representing discrete objects (like words, sentences, images, or audio) as vectors in a lower-dimensional space.

While the statistical vectorization of BoW and TF-IDF treat each word as an independent feature, they do not capture the meaning of words or relationships between words beyond raw counts or frequency. Word Embeddings address this limitation by learning dense, low-dimensional vectors for words such that those vectors encode meaningful (semantic) information. In an embedding, each word is represented by a vector much smaller than the full vocabulary size, and crucially, similar words will have similar vectors. This idea stems from the linguistic theory of distributional semantics, often summarized as "words that appear in similar contexts tend to have similar meanings", the concept is further described in [**The-Distributional-Hypothesis-in-NLP**]. In practice, this means an embedding algorithm will try to place words like "espresso", "kaffe" and "latte" close together in the vector space, whereas an unrelated word like "o-ring" might end up far apart.

Popular methods to learn word embeddings from large text corpora include Word2Vec and GloVe. These algorithms scan through billions of words of text and adjust word vector space. The end result is a vector for each word (for example, of dimension 50, 100, or 300) where distances or angles between vectors reflect semantic similarity. For instance, one classic result from Word2Vec is that the vector arithmetic operation $\text{vector}(\text{"king"}) - \text{vector}(\text{"man"}) + \text{vector}(\text{"woman"})$ yields a result close to $\text{vector}(\text{"queen"})$. This demonstrates that the embedding space has captured gender relationships (among other types of relations) in its geometry. When words are represented in this way, even simple arithmetic on their vectors can produce meaningful results, indicating that the model has learned latent linguistic structure. [**Hands-on-Machine-Learning**] chapter 13, page 467-468.

There are several advantages of embedding over the earlier sparse representations. First, embeddings are dense and low-dimensional. Instead of a 10,000-dimensional sparse vector (mostly zeros) for a word, we might have a 100-dimensional dense vector (all values non-zero). This compactness tends to make machine learning models more efficient and less prone to overfitting, since the model has far fewer parameters to learn and the input space is more condensed. Secondly, embeddings capture semantic similarity in a way that sparse vectors cannot. In a TF-IDF vector, there's no "built-in" reason for coordinates for "king" and "queen" would be related, they're just independent features. But in an embedding space, if "king" and "queen" are used in similar contexts, their vectors will end up close together, reflecting that semantic relationship. This is extremely useful for any ML task

involving 'meaning', such as clustering similar documents, searching for related content, or improving generalization in text classification (since synonyms or related terms will be recognized as similar features, rather than being treated as completely unrelated inputs).

In summary, word embedding mark a shift from purely statistical representations to semantic representations. Each word's vector is learned based on context, such that the geometry of the vector space encodes meaning, (for example, vectors for "espresso", "kaffe", "latte" might cluster together as "kaffevarianter"). This semantic encoding allows algebraic reasoning on words.

5.1.4 Why Represent Text as Vectors?

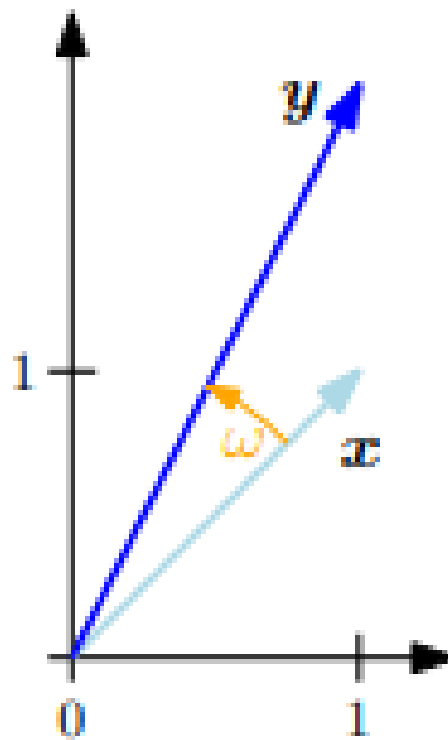
Representing words or documents as vectors makes them 'comparable'. Each word or text becomes a point in (potentially high-dimensional) space. We can then define distance or similarity measures in this space to ask questions like "how similar are these two pieces of text?". For example, we can compute the cosine-similarity between two word vectors, which is the cosine of the angle between the two vectors, x and y . (chapter 3, page 75 [Mathematics-for-Machine-Learning])

If the value is 1, the two vectors would point exactly the same direction (maximum similarity); if 0, they are completely unrelated. In theory, words like "espresso" and "kaffe" will have a smaller angle between their embedding vectors (considered high cosine similarity) if the embedding has learned that they are related coffees, whereas "espresso" and "touchskærm" would be far apart (possibly near 0 cosine similarity).

By computing similarities in vector space, we enable tasks like information retrieval (find the documents most similar to a query by comparing their vector representations), clustering (group documents by topical similarity using distance measures), and nearest-neighbor classification (classify a text by looking at the classes of its closest neighbors in vector space). None of these would be possible on raw text, because we can't meaningfully average or compare words directly, but with vectors, we can. In essence, numerical representations make language measurable.

Once text is represented as vectors, we can feed them to a standard classifier (logistic regression, SVM, Neural Network etc.). Consider an 'instruction', we can represent each 'instruction' as a vector (say, a TF-IDF vector of its words, or perhaps an average of its word embeddings) and then train a classifier to output whether that vector corresponds to a certain 'spare part' or 'categorical' (milk-issue, electronic etc.). The learning algorithm will find patterns in the vector (instruction) that in the end can correlate with the correct spare part needed. This is only possible because we turned the 'instruction' into numbers. If left as raw text, the machine learning model would simply not process it. Vectors are

Figure 3.5 The angle ω between two vectors x , y is computed using the inner product.



Example 3.7 (Orthogonal Vectors)

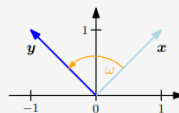


Figure 3.6 The angle ω between two vectors x , y can change depending on the inner product.

the common language between human text and mathematical models.

5.2 Overview of Classification Models

This section presents three machine learning models commonly used for classification tasks. Each model takes a different approach to learning patterns from data and has specific strengths and weaknesses.

5.2.1 Random Forest Classifier

The Random Forest Classifier is a tree based model that builds a collection of decision trees. It uses a technique called bagging, where multiple trees are trained independently on different random samples of the training data. These samples are drawn with replacement. After training, each tree makes a prediction, and the final output is based on majority voting among all trees. This approach helps reduce overfitting and increases the model's stability.

Each tree in the forest is also trained using only a random subset of the features at each split. This encourages diversity among the trees and improves generalization by reducing correlations between them. The Random Forest Classifier is easy to implement and handles a variety of data types well.

However, the model may encounter difficulties when applied to high-dimensional and sparse input data. Representations where most features are zero, can make it harder for trees to find informative splits. Although the model remains functional, its efficiency and accuracy may be limited under these conditions.

5.2.2 Extreme Gradient Boosting

Extreme Gradient Boosting builds upon the general idea of using multiple decision trees but differs from Random Forest by how it combines them. Instead of training all trees independently and aggregating their outputs, this method constructs trees sequentially. Each new tree is trained to correct the prediction errors made by the previous trees.

This is achieved through gradient boosting, where the model minimizes a loss function using gradient descent techniques. As a result, each tree is focused on reducing the specific mistakes made so far. This step-by-step learning process allows the model to capture complex patterns and refine its predictions.

Despite these strengths, Extreme Gradient Boosting faces similar challenges as other tree-based models in high-dimensional and sparse data spaces. Sparse input can limit the

model's ability to find strong and consistent splits. Tree-based methods generally perform best when the input features are dense and contain clear patterns.

5.2.3 Support Vector Classifier

The Support Vector Classifier works by finding a boundary, called a hyperplane, that separates data points from different classes. It does this by trying to keep the boundary as far away from the closest points on either side as possible. These closest points are called support vectors.

This method works especially well when the number of features is large, which is often the case in text classification. It is also effective when the data is sparse because it focuses only on the most important points in the dataset. SVC uses all of the input dimensions at once to find a single best boundary, instead of making many small decisions like trees do.

A linear version of SVC is often used when the data is already in a high-dimensional space. In such cases, the model is both simple and efficient.

Chapter 6

Experimental phase

For this project, a variety of approaches were tested. These can be categorized into two main sections: a data-centric approach, which involved experimenting with different methods of utilizing the data, and a text-centric approach, which focused on processing and analyzing the textual content. This is later in the chapter evaluated and discussed. The first section is about the data used for this experimental phase.

6.1 Data preprocessing for the experimental phase

6.1.1 Chosen data

The data we received from Bentax in order to complete this project consisted of three datasets: -Alle Arbejdsordre.xlsx, rows: 70162, columns: 23, Unique Work Order numbers: 70162 Arbejdsordre med afskrevet reservedel.xlsx, rows: 81467, columns: 33, Unique Work Order numbers: 0 Confidential:

To fulfill the aim of this project we will mainly be working with the dataset Arbejdsordre med afskrevet reservedel.

The core dataset for this project, "Arbejdsordre med afskrevet reservedele", differs from the broader dataset "Alle Arbejdsordre" by including explicit information about the spare parts that were used to complete a repair. This makes it more suitable for our machine learning objective, "predicting the correct spare part ('Product ID') based on a customer-provided fault description ('Instructions')". It is important to note that the work order numbers are not unique in this dataset, as each row represents the use of a specific spare

part. Consequently, multiple entries may refer to the same service case, just with different components involved.

6.1.2 Data Cleaning

During the preprocessing, several steps were taken to clean and simplify the dataset. First, two metadata columns marked as 'Do Not Modify' were removed. Additionally, column names were standardized by stripping repetitive suffixes such as 'Work Order' and by removing special characters.

6.1.3 Column Selection

As the projects primary aim is to predict the correct Product ID based upon a fault description. Therefore we can only include features that would be available before the technician arrives at the site. Features like 'intern note', 'name', 'supplier item number' and 'Temp work order summary' were excluded to prevent data leakage. A selection of other columns was removed because they either contained customer-identifiable information (e.g. address details), or data that would only be available post-prediction task that happens before the technician resolves the issue columns were excluded to prevent data leakage. Among the removed columns were: Street 1, City, Longitude, Latitude, System Status, Resource Booking, Created On, Temp Work Order Summary, Service Territory, Unit Cost, and Ware House Name. (oversættes til dansk, columns)

6.1.4 Asset Filtering: Focusing on Thermoplan

The primary filtering challenge was isolating only coffee machines from the Thermoplan brand, as these are the most prevalent in the dataset and best suited for model training due to data volume and consistency. Initially, attempts were made to filter by brand name but this proved insufficient. Upon closer inspection, it was discovered that Thermoplan models consistently include product names starting with "BW", followed by model indicators such as "3", "4", "4C", or "4Neo". Using this pattern within the column 'Primær Asset Produkt', a total of 47,755 relevant instances were extracted.

6.1.5 Instruction Filtering and Data Reduction

The model this project aims to build depends on the availability of two key pieces of information: the customer instruction and the spare part used. Therefore, all rows without a spare part or without an instruction were removed. A further inspection revealed that the majority of rows missing an instruction came from the internal workshop ('værksted') rather than customer-facing repairs. Since our focus is on customer-reported breakdowns ('Nedbrud'), these cases were excluded.

After filtering, the dataset contained approximately 670 unique product IDs. However, the distribution was highly imbalanced: many spare parts occurred only once or twice. To improve model robustness and avoid noise, rare product IDs were iteratively filtered out. Removing parts with fewer than 10 occurrences reduced the number of labels to about 250, offering a more balanced training target without discarding too much data. The threshold remains under consideration and may be adjusted based on business priorities or a list of actively stocked spare parts.

6.1.6 Instruction Text Analysis

A significant challenge in this dataset is the unstructured and inconsistent nature of the 'Instruction' field. These are free-text messages written by customers without access to internal error codes (in most cases also external error codes). As a result, descriptions range from highly informative ("powder error") to vague ("virker ikke", or simply "utæt"). Others contain mixed or irrelevant content, such as building access instructions or internal technician notes.

An initial analysis of word frequencies in instructions showed a high prevalence of stop-words, which were subsequently removed. After this preprocessing step, the most common remaining terms appeared more relevant to actual mechanical or operational issues and could potentially be linked to specific spare parts. This supports the use of text-based machine learning methods such as NLP-based classifiers.

6.1.7 Distribution of Fault Types

The column 'Primary Incident' captures the type of issue reported. As the aim is predictive maintenance, special attention was given to the subset labeled "nedbrud" (breakdown), which reflects real operational failures. Only instances with both an instruction and an associated spare part were included in the modeling pipeline. Exploratory analysis revealed that breakdowns not only constitute a large portion of relevant cases, but also exhibit greater

instruction length on average, suggesting more detailed customer input.

6.1.8 Data Partitioning

The filtered dataset was split into training and test sets, using a 80/20 split, resulting in 38,204 training samples and 9,551 test samples. Stratification was not applied at this stage, although this may be revisited depending on label imbalance and evaluation requirements.

6.2 Evaluation Metrics

When building a machine learning model, it is not enough to simply create predictions, it also matters how those predictions are evaluated. In this project, the model returns a ranked list of five spare parts. Because more than one of the predicted spare parts may be correct for a given input, this setup is treated as a multi-label problem with a Top-5 output. The evaluation must therefore reflect not just whether correct parts are predicted, but how many are found, and how well the predicted set matches the actual one.

To capture these aspects, we use recall and IoU as well as precision@5, F1@5, and quantity accuracy. In addition, we include quantity recall and quantity precision to evaluate how well the model predicts the number of parts required for a repair. Identifying the correct set of part types (Top-5) and estimating how many of each are needed (quantity) are separate things but both will be evaluated.

Recall measures the proportion of relevant parts successfully predicted by the model. It reflects how many of the true labels were retrieved within the Top-5 list.

$$\text{Recall} = \frac{|\text{True} \cap \text{Predicted}|}{|\text{True}|}$$

For example, if a repair requires three specific spare parts: {A, B, C}, and the model predicts the Top-5 list: {A, D, E, F, B}, then two out of the three true parts (A and B) are correctly included. The recall in this case is:

$$\text{Recall} = \frac{2}{3} \approx 0.67$$

IoU quantifies the overlap between the predicted and actual label sets. Unlike recall, IoU penalizes both missed and irrelevant predictions, making it a rather balanced indicator of prediction quality.

$$\text{IoU} = \frac{|\text{True} \cap \text{Predicted}|}{|\text{True} \cup \text{Predicted}|}$$

For example, if the true parts are {A, B, C} and the model predicts {A, D, E, F, B}, then the intersection is {A, B} and the union is {A, B, C, D, E, F}. Therefore, the IoU is:

$$\text{IoU} = \frac{2}{6} \approx 0.33$$

Precision@5 measures the proportion of predicted parts that are correct out of the five returned. This metric assesses how accurate the model's suggestions are within the Top-5 limit.

$$\text{Precision@5} = \frac{|\text{True} \cap \text{Predicted}|}{5}$$

To compute it, you take the five predicted parts, check how many of them are actually correct (i.e., appear in the true label set), and divide that number by 5.

F1@5 is the mean of precision@5 and recall.

$$\text{F1@5} = 2 \cdot \frac{\text{Precision@5} \cdot \text{Recall}}{\text{Precision@5} + \text{Recall}}$$

Quantity accuracy compares the number of predicted parts to the number actually used in the repair.

$$\text{Quantity accuracy} = \frac{\text{Number of correct quantity predictions}}{\text{Total predictions}}$$

Quantity recall measures how many of the actually required units were correctly predicted in terms of quantity, regardless of the predicted part types.

$$\text{Quantity Recall} = \frac{\sum_i \min(\text{True}_i, \text{Predicted}_i)}{\sum_i \text{True}_i}$$

Quantity precision measures the proportion of predicted units that were actually required. It focuses on how many of the predicted quantities were correct, and penalizes the model when it overestimates the number of parts needed.

$$\text{Quantity Precision} = \frac{\sum_i \min(\text{True}_i, \text{Predicted}_i)}{\sum_i \text{Predicted}_i}$$

6.2.1 Data Visualization

[needs checking: above]

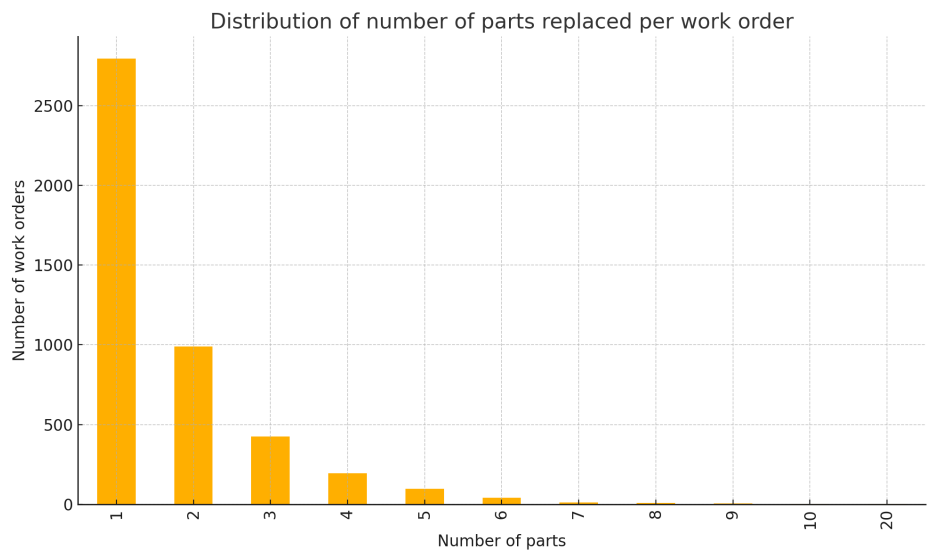


Figure 6.1: Number of Parts Replaced pr. Work Order

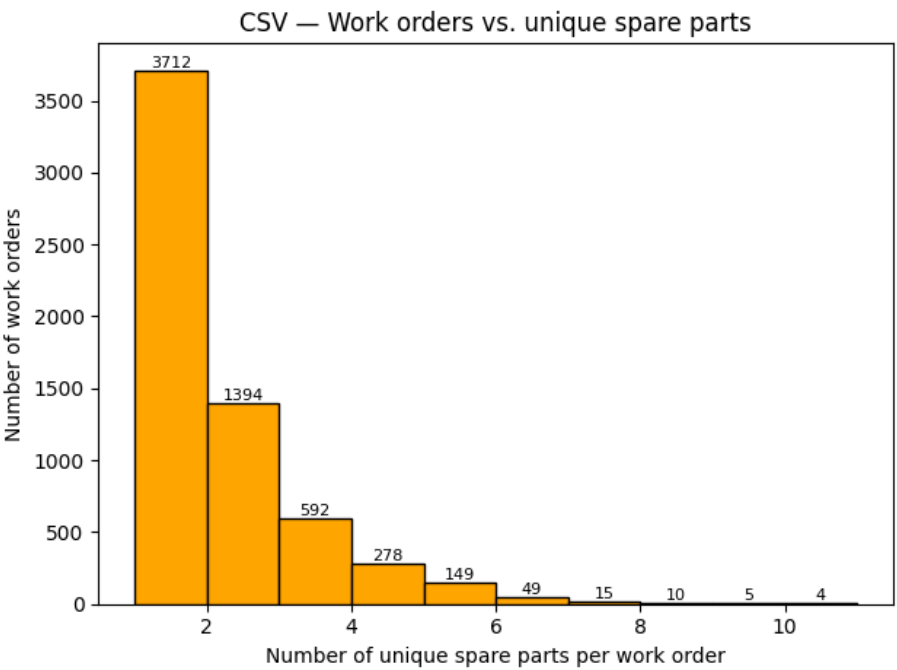


Figure 6.2: Number of unique Parts per Work Order

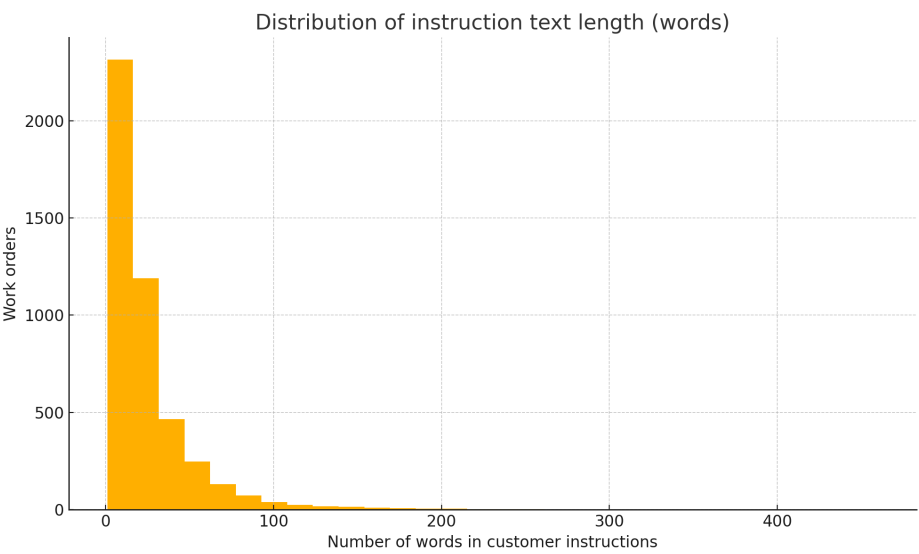


Figure 6.3: Instruction Lenght

6.3 Methods and implementation of the experimental phase

6.4 Data approach

This sections covers different methods to use the anvaible data. The first is about a method to split the data in different categories instead of using the dataset togheter as a whole, this is later reviewed in the results chapter. Another method experimented with focussed on solving the unbalanced data using SMOTE. At last LSA, a dimentionality reduction method, was experimented with to see if it had any impact on the models performance if it had fewer dimentions.

6.4.1 Splitting the dataset in categories

In this approach, a rule-based keyword classification approach was used to label textual work order instructions in Danish.

The goal of the dataprocessing pipeline was twofold: (1) to clean and normalize textual data, and (2) to assign interpretable multi-label classifications to each work order, based on domain-specific keyword sets.

The overall pipeline consists of structured data loading, cleaning, linguistic normalization, token-level filtering, keyword-based classification, and finally, grouped export of the annotated dataset.

Data Loading and Cleaning

The input data, sourced from the excel file from bentax, contains fields such as work orders, instruction texts, associated product IDs and machine descriptors, as earlier described in the 2.3.1. Initial preprocessing steps include the removal of missing or invalid entries. Specifically, work orders were excluded if they had missing product IDs, if the number of products did not match recorded quantities, or if the instruction field was empty. Numeric quantities were normalized to fall within a valid range (1 to 5), and machine identifiers were extracted using regular expressions. This was done to prepare the data for a split.

[implementation example?]

Text Normalization and Tokenization

Each instruction string was lowercased, stripped of punctuation and special characters, and normalized using Unicode NFKD decomposition. Numerals were replaced with a placeholder token, to reduce sparsity without losing semantic cues. Subsequently, the cleaned text was tokenized into word-level tokens, and Danish stopwords were removed. To improve robustness against inflectional variation, all tokens were passed through a Snowball stemmer for the Danish language, reducing them to their morphological root forms.

[implementation example?]

Rule-Based Keyword Classification

The core of the classification approach is a manually curated, domain-specific keyword system. Five functional categories were defined based on maintenance semantics:

- **WATER** (for example: “vand”, “drypper”, “flowfejl”)
- **MILK** (for example: “mælk”, “suge”, “køl”)
- **MECHANICS** (for example: “kværn”, “kaffebønne”, “fejl”)
- **CLEANING** (for example: “rens”, “grums”, “filterskifte”)
- **ELECTRONICS** (for example: “skærm”, “display”, “slukker”)

Each category includes both unigrams and common bigrams/or trigrams, enabling phrase-level classification (for example: “flow error” or “rense program”). All keywords were stemmed during initialization to ensure compatibility with the stemmed tokens derived from instruction texts. Also a tool was made to visualize the most frequent words, to include in each category.

[implementation example?]

For each instruction, classification was performed by matching stemmed tokens and n-grams (up to trigrams) against the precompiled keyword lists. This process resulted in a multi-label assignment per work order, as individual instructions could invoke multiple functional categories (for example: a work order involving both mechanical failure and

water leakage).

[implementation example?]

Grouping and Export

After classification, the data was grouped by work order ID to aggregate products and instructions at the task level. Each grouped entry retained its associated instruction, machine type, classified labels, and target products. For downstream use, the data was then exported into structured CSV files, partitioned both by machine model and assigned class label. This export strategy facilitates subsequent model training or manual inspection per functional subsystem, so that a model can be trained on only for example 'BW3'-cleaning associated instructions.

Model Training

Once the data is split per machine model and category, each CSV subset is used to train a Support Vector Classifier (SVC). Text features are transformed via a TF-IDF vectorizer (with 1–3 ngrams, limited vocabulary and Danish stopwords), and categorical features (machine type) are one-hot encoded. Training is performed with grouped 5-fold cross-validation to ensure that all instructions from the same work order remain in the same fold. For each fold:

1. The TF-IDF + one-hot encoding pipeline is fit on the training split.
2. A binary sigmoid-kernel SVC is trained for each part-ID label and calibrated to output probabilities.
3. A secondary SVC predicts part quantities (1–5), with a fallback to a constant dummy classifier when insufficient positive samples exist.

Finally, predictions across folds are used to compute precision@K, recall@K, F1@K, Hamming score, IoU, and quantity accuracy, providing a robust estimate of model performance before full retraining on the complete subset. The metrics are explained earlier in chapter 6 6.2.

6.4.2 Data imputation methods

SMOTE (Synthetic Minority Oversampling Technique)

As shown in Figure 6.1, the dataset is highly imbalanced, with many labels associated with only a single repair instance. This class imbalance negatively affects the performance of machine learning models, especially in multi-label settings. Therefore, we evaluated SMOTE as a resampling strategy to synthetically increase the presence of underrepresented labels.

SMOTE (Synthetic Minority Oversampling Technique) is a widely used oversampling method that generates synthetic samples for minority classes instead of merely duplicating existing ones [chawla2002smote]. The method reduces the risk of overfitting, which is common in naive oversampling, by introducing variation in the newly generated data.

Given a minority class instance $\mathbf{x} \in \mathbb{R}^n$, SMOTE generates a new synthetic instance \mathbf{x}_{new} using one of its k -nearest neighbors \mathbf{x}_{nn} , defined as:

$$\mathbf{x}_{\text{new}} = \mathbf{x} + \lambda \cdot (\mathbf{x}_{\text{nn}} - \mathbf{x})$$

where $\lambda \sim \mathcal{U}(0,1)$ is a random number drawn from a uniform distribution. This interpolation creates new instances along the line segments joining a sample and its neighbors in feature space, maintaining local characteristics of the minority class.

The core steps of the algorithm are:

1. **Identify minority class samples:** Select all instances belonging to the class(es) with low representation.
2. **Compute k-nearest neighbors:** For each minority instance, determine its k nearest neighbors (typically using Euclidean distance).
3. **Interpolate new samples:** For a desired number of synthetic examples, randomly select one of the k -neighbors and generate a synthetic point using the interpolation formula above.

[Example: drawing of how smote works.]

[ref is on its way:]

LDA and LSA for Dimensionality Reduction

Initial tests using text as features demonstrated the high dimensionality of the dataset when TF-IDF was applied. To possibly solve this issue, two dimensionality reduction techniques, LDA and LSA, were experimented with.

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique that seeks to maximize the separation between different classes while minimizing intra-class variance. Unlike Principal Component Analysis (PCA), which is unsupervised and focuses on maximizing variance, LDA leverages class labels to find linear combinations of features that best distinguish classes, [ref is on its way:]

The LDA process involves the following steps:

1. Compute the mean vectors for each class.
2. Calculate the within-class and between-class scatter matrices.
3. Derive the Fisher criterion to identify the optimal projection matrix that maximizes class separation.

Latent Semantic Analysis (LSA), on the other hand, is an unsupervised method that seeks to uncover latent structures in textual data by decomposing the term-document matrix using Singular Value Decomposition (SVD). It is particularly effective in reducing the dimensionality of TF-IDF matrices while retaining key semantic relationships [ref is on its way:]

The key steps of LSA include:

1. Construct the term-document matrix based on TF-IDF values.
2. Apply SVD to factorize the matrix into three matrices (U, Σ , and V).
3. Select the top k singular values and corresponding vectors to reduce dimensionality.

Both LDA and LSA are essential in text classification and clustering tasks, especially when dealing with sparse, high-dimensional data. The choice between them depends on whether labeled data is available and the specific nature of the dataset, [ref is on its way:]

6.5 Text approach

6.5.1 TF-IDF

Introduction

Introduction TF-IDF

Building on the foundational understanding of text vectorization covered in the technical overview, this section focuses on the practical application and implementation details of TF-IDF.

Term Frequency–Inverse Document Frequency (TF IDF) is a statistical method used to evaluate the importance of a word within a specific document relative to an entire collection of documents, commonly referred to as a corpus. It is widely applied in the fields of NLP and information retrieval due to its simplicity, efficiency, and effectiveness in identifying relevant terms [simha2021tfidf].

The method comprises two key components: Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures how often a specific word appears in a document, either as a raw count (e.g., “the word appears five times”) or as a relative frequency, calculated by dividing the word’s occurrence by the total number of words in the document. This provides an indication of the term’s importance within the context of the individual document [simha2021tfidf].

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

In contrast, IDF evaluates the degree to which a word is rare or unique across the entire corpus. The fewer documents a term appears in, the higher its IDF value. Common words such as “and” or “the” receive low IDF scores, as they appear in nearly all documents and do not meaningfully differentiate content[simha2021tfidf].

$$\text{IDF}(t) = \log \left(\frac{N}{1 + df(t)} \right)$$

By multiplying TF and IDF, one obtains the TF-IDF weight, which highlights words that are both frequent in a specific document and rare across the rest of the corpus. This

makes TF-IDF particularly suitable for identifying distinctive and meaningful terms while filtering out generic low-information words.

Old Implementation of TF-IDF

In practice, TF-IDF is commonly used as a vectorization technique, as seen in the snippet below, converting text into numerical vectors that can be processed by machine learning algorithms. Each document is represented as a vector, each element corresponding to the TF-IDF weight of a word from the overall vocabulary. These vectors can then be applied in various tasks such as document classification, clustering, similarity analysis, and keyword extraction [simha2021tfidf].

```
text_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(
        ngram_range=NGRAM_RANGE,
        min_df=MIN_DF,
        max_df=MAX_DF,
        max_features=max_features,
        stop_words=list(stop_words),
        sublinear_tf=True
```

Figure 6.4: example of TF used as a vector

As seen in 6.4, there are a few hyperparameters which can be adjusted, as seen in 6.1.

Feature	Description
ngram_range	Use word pairs or triples (e.g. unigrams, bigrams, trigrams)
min_df	Ignore terms that appear in fewer documents
max_df	Ignore very frequent terms (e.g. stopwords)
max_features	Limit the number of features in the vocabulary
stopwords	Remove predefined common words (e.g. Danish stopwords)
sublinear_tf	Apply logarithmic scaling to term frequency

Table 6.1: Adjustable TF-IDF hyperparameters and their effects.

Despite its advantages, TF – IDF has notable limitations. It does not capture semantic meaning or the context in which

Another challenge is the potential for memory inefficiency, especially when applied to large texts. Since the length of each vector equals the size of the vocabulary, TF-IDF is susceptible to the curse of dimensionality. The max features parameter, as seen in 7.8, can be quite high, which can degrade performance by its complexity [simha2021tfidf].



```
Full TF-IDF vocab size: 75607
```

Figure 6.5: Example of a high dimensional output of TF-IDF, where the vector size is equal to dimensions

6.5.2 Sentence Transformers

Introduction and Motivation

As an intermediate step between traditional vectorization techniques and , sentence transformers offer a way to embed entire sentences into dense numerical vectors that capture semantic meaning. Unlike TF-IDF, which is based on word frequency and lacks contextual understanding, sentence transformers are trained to reflect how similar in meaning two sentences are, even if they do not share surface-level vocabulary.

The model applied in this approach is paraphrase-multilingual-MiniLM-L12-v2. It produces a 384-dimensional embedding for each sentence and supports over 50 languages, including Danish. While it does not incorporate domain-specific tuning, the model's training on semantic similarity tasks provides general-purpose sentence representations that can be used as input features for classification tasks.

Modeling Approach

In this method, sentence embeddings are generated for each work order description and used as input to a SVC ?. The goal is to move beyond bag-of-words representations and introduce semantically meaningful features, while still maintaining a simple and interpretable classifier. This architecture serves as a conceptual and computational bridge between TF-IDF-based models and the more complex transformer-based fine-tuning approaches described in the next section.

Feature Engineering

Method Although sentence transformers such as MiniLM-L12-v2 abstract away traditional word-level feature engineering and tokenization some preprocessing is still essential to ensure that the input text is clean, consistent, and suitable for semantic embedding.

In the context of sentence transformers, feature engineering involves preparing text so it can be effectively encoded into dense semantic vectors that capture the overall meaning of an input sentence. These sentence-level embeddings can then be used as input features for downstream classifiers.

The key feature engineering steps for sentence transformer-based models include:

- Text normalization: Standardizing text by converting it to lowercase and removing

```
from sentence_transformers import SentenceTransformer
```

Figure 6.6: Enter Caption

```
!Sentence embeddings
print("Embedding instructions...")
model = SentenceTransformer(SENTENCE_MODEL_NAME)
X_train_embed = model.encode(X_train['Instructions'].tolist(), batch_size=BATCH_SIZE, show_progress_bar=True)
X_val_embed = model.encode(X_val['Instructions'].tolist(), batch_size=BATCH_SIZE, show_progress_bar=True)
```

Figure 6.7: Sentence Embedding Code Example

non-alphanumeric characters to reduce variability and noise.

- Field concatenation: Merging relevant text fields (e.g., instructions, internal notes, and summaries) into a single coherent input string to provide context.
- Encoding: Passing the cleaned and concatenated text through the `encode()` function of the sentence-transformers library, which returns a fixed-length vector representation (in our case, 384 dimensions).
- Label encoding: Transforming multi-label categorical targets (spare part IDs) into binary vectors using a `MultiLabelBinarizer`, making them suitable for classification.
- Rare class handling: Grouping infrequent labels into an “Other” class to mitigate the effects of label imbalance during training.

These steps ensure the input text is presented in a uniform format that makes use of the pretrained model’s semantic understanding. Unlike traditional methods that operate at the word level, sentence transformers process entire input strings holistically, capturing contextual meaning in a way that is especially useful for short, noisy, and domain-specific text, such as the Danish maintenance instructions in this project.

Implementation The sentence-transformers Python library was used to load the pretrained paraphrase-multilingual-MiniLM-L12-v2 model. Each work order instruction was passed to the `encode()` function, which produced a 384-dimensional dense embedding capturing the semantic meaning of the input. These sentence-level embeddings were then used as input features to a linear SVM classifier.

The figures below illustrate the model initialization and the embedding generation process.

Model Fit

Method The underlying transformer model was pre trained on a paraphrase detection task. It learns to position similar sentences close together in vector space. Internally, the model tokenizes the input, processes it with self-attention layers, and pools the token embeddings to generate a single sentence vector. The transformer's weights remained fixed throughout.

Implementation The 384-dimensional embeddings were input to a linear SVM classifier. Then the classifier's outputs are transformed into probability estimates.

Model Tuning

Method and Implementation The classifier was tuned by adjusting the regularization parameter of the SVC. See ??

6.5.3 LLM

Introduction and BERT-based Classification approaches

In this approach, we explore the automation of text-based work order classification using machine learning techniques. Although LLMs such as BERT are beyond the formal scope of our curriculum, we chose to include them as an experimental component. We believe that transformer-based architectures represent a cutting-edge and highly relevant approach for solving classification problems involving unstructured text. For references, we have used the Report of BERT Pretraining from Google AI Language [**BERT-Pre-Training-by-Google**] to understand how BERT works, and we gathered the Danish BERT version from HuggingFace.com [**danskbert-HuggingFace**].

As a proof of concept, two different methods have been tested that leverage BERT (Bidirectional Encoder Representations from Transformers), a transformer-based neural network architecture developed by Google that has achieved state-of-the-art performance across a wide range of NLP tasks such as sentiment analysis, question answering, and text classification. Unlike traditional machine learning models that require manual feature engineering, BERT is capable of learning context-rich representations from full sentences bidirectionally, capturing nuanced linguistic patterns that are often crucial in technical and unstructured work order descriptions [**devlin2018bert**].

There has been implemented two separate BERT-based approaches to classify spare part types from Bentax's domain-specific text data:

1. **Fine-tuning a pre-trained Danish BERT model:**

In this setup, we take a publicly available pre-trained Danish BERT model (vesteinn/DanskBERT) and fine-tune it directly on our task. Several relevant text fields are combined into a single input sequence and used to train the model to predict spare part categories. We treat this as a supervised classification task and utilize Hugging Face's Trainer API to manage the training loop, evaluation metrics, and model checkpointing. This approach is theoretically powerful, as it allows the model to adjust its internal parameters specifically to our dataset, thereby optimizing performance through end-to-end learning.

2. **Using BERT as a feature extractor with a Random Forest classifier:**

In contrast to the first method, this approach does not modify the BERT model's weights. Instead, we use Danish BERT to generate fixed vector embeddings for each input sample, and train a classical machine learning model (Random Forest classifier) on top of these embeddings. This hybrid approach offers a more interpretable and computationally efficient alternative, especially useful when resources are limited or

explainability is important.

Both approaches rely on a consistent and shared preprocessing pipeline that includes normalization of text fields, handling of rare classes, and label encoding. This allows us to isolate the effect of the modeling strategy itself and fairly compare the two methods in terms of performance, complexity, and applicability to real-world use cases.

By evaluating both techniques, we aim to understand the trade-offs between deep learning and classical machine learning when applied to Danish industrial data. Ultimately, our goal is to assess which method better suits Bentax's practical requirements for automated text classification.

Feature Engineering

Method Feature engineering is the process of transforming raw data into structured input suitable for machine learning algorithms. In the context of LLMs like BERT, this process differs from traditional approaches because the model itself captures complex linguistic patterns through contextual learning. Nevertheless, certain preprocessing steps are still necessary to ensure compatibility with the model's input requirements and to facilitate effective training.

In natural language processing (NLP), common feature engineering steps include:

- Text normalization: Converting all text to lowercase and removing non-alphanumeric characters to maintain consistency.
- Field concatenation: Merging multiple text fields into a single sequence (so that the model can consider all relevant information at once).
- Tokenization: Breaking the text into tokens (handled by BERT's built-in tokenizer, which maps words and sub-words to numeric indices).
- Label encoding: Converting categorical labels (like spare part categories) into numeric class indices for classification.
- Embedding generation: Producing vector representations of text. For example, if using BERT as a fixed feature extractor instead of fine-tuning it, one would generate embeddings from the model without updating its weights.

These steps ensure consistency across all samples and allow the model to focus on meaningful textual patterns during training and inference. The feature engineering strategy

```

59 # Kombiner alle relevante tekstfelter
60 def combine_text_fields(df):
61     # Kombiner alle tekst-kolonner med vagt
62     combined = []
63     for i, row in df.iterrows():
64         combined_text = ""
65         # Giv instructions højere vagt ved at gentage dem
66         if not pd.isna(row.get('Instructions')):
67             combined_text += preprocess_text(row['Instructions']) + " "
68             # Gentag instructions for at give dem mere vagt
69             combined_text += preprocess_text(row['Instructions']) + " "
70
71         # Tilføj temp work order summary
72         if not pd.isna(row.get('Temp work order summary')):
73             combined_text += preprocess_text(row['Temp work order summary']) + " "
74
75         # Tilføj intern note
76         if not pd.isna(row.get('Intern note')):
77             combined_text += preprocess_text(row['Intern note']) + " "
78
79         # Tilføj incident type information
80         if not pd.isna(row.get('Incident Type (Work Order Incident)')):
81             combined_text += preprocess_text(row['Incident Type (Work Order Incident)']) + " "
82
83     combined.append(combined_text.strip())
84     return combined

```

Figure 6.8: Enter Caption

must also account for language-specific challenges, in our case, handling informal and unstructured Danish service descriptions.

Implementation For the spare part classification task with Bentax data, we developed a shared preprocessing pipeline applied to both modeling approaches described in this report. The following steps were implemented for all samples before modeling:

- Field concatenation: Key text fields from each work order (such as Instructions, Incident Type, Temporary Work Order Summary, and Internal Note) were concatenated into one combined string per sample.
- Text normalization: The combined text was lowercased and stripped of special characters to reduce noise.
- Label encoding: Spare part labels (the target classes) were converted into numeric codes.
- Rare class handling: Very infrequent labels were grouped into a single “Other” category. This was done to address class imbalance and to improve model stability, ensuring that the classifier does not poorly fit extremely rare classes.

After this preprocessing, the data was ready for modeling. We pursued two different BERT-based modeling approaches:

Approach 1: Fine-Tuning DanskBERT

In the first approach, we take a pre-trained Danish BERT model (vesteinn/DanskBERT) and fine-tune it end-to-end on our classification task. All the preprocessed text for a work order is fed into the model, and the model’s weights are adjusted during training to

```
96 model = AutoModelForSequenceClassification.from_pretrained(  
97     model_name,  
98     num_labels=num_labels,  
99     problem_type="single_label_classification"  
100 )
```

Figure 6.9: Enter Caption

predict the correct spare part category. We utilize Hugging Face’s Trainer API to handle the training loop, evaluation, and checkpointing:

This fine-tuning approach leverages the full capacity of the transformer model by allowing it to adapt its internal representations to our specific dataset in a supervised fashion.

Approach 2: BERT as Feature Extractor with Random Forest

The second approach uses the same Danish BERT model only to generate text embeddings, which are then used as features for a classic machine learning classifier (a Random Forest). In this setup, the BERT model’s weights remain frozen – it is not fine-tuned on our data. Instead, for each preprocessed text sample, we obtain a fixed-length vector (the embedding of the [CLS] token from BERT) that captures the sample’s linguistic information:

These BERT-based features are then input to a `RandomForestClassifier` (from `scikit-learn`) which we train to predict the spare part category. This hybrid approach combines BERT’s powerful language representations with the simplicity and interpretability (the ability of human to understand and comprehend the reasoning behind a models’s predictions and decisions) of a traditional classifier. It is also computationally more efficient to train, since only the Random Forest must be fitted to the data. Moreover, using a Random Forest allows us to examine feature importance, giving insight into which dimensions of the BERT embeddings are most informative for the classification.

Both approaches use the same input pipeline, ensuring a fair comparison. This design allows us to explore the trade-offs between an end-to-end deep learning model and a two-step hybrid model. In summary, Approach 1 offers a theoretically optimal solution by fine-tuning all of BERT’s parameters to our task (potentially achieving higher accuracy given enough data), whereas Approach 2 emphasizes practical benefits like faster training, easier hyperparameter tuning, and model transparency (important for explaining results to stakeholders).

Model Fit

Method In traditional machine learning, “model fitting” refers to the process of training a model on a dataset (for example, calling a `model.fit(X, y)` function to make the model learn the relationship between inputs and outputs). In our project, we are working with a

```

123 # --- Funktion til at lave BERT embeddings ---
124 def get_bert_embeddings(texts, model_name='vesteinn/DanskBERT', max_length=128, batch_size=32):
125     # Indlæs tokenizer og model
126     tokenizer = AutoTokenizer.from_pretrained(model_name)
127     model = AutoModel.from_pretrained(model_name)
128
129     # Sæt model til evaluation mode
130     model.eval()
131
132     all_embeddings = []
133
134     # Behandl tekster i batches for at spare hukommelse
135     for i in range(0, len(texts), batch_size):
136         batch_texts = texts[i:i+batch_size]
137
138         # Tokeniser batch
139         encoded_input = tokenizer(
140             batch_texts,
141             padding='max_length',
142             truncation=True,
143             max_length=max_length,
144             return_tensors='pt'
145         )
146
147         # Få embeddings
148         with torch.no_grad():
149             outputs = model(**encoded_input)
150
151         # Brug [CLS] token embedding som repræsentation for hele teksten
152         embeddings = outputs.last_hidden_state[:, 0, :].numpy()
153         all_embeddings.append(embeddings)
154
155     # Kombiner alle batches
156     return np.vstack(all_embeddings)

```

Figure 6.10: Enter Caption

large pre-trained language model, so the notion of model fitting is slightly different. We did not train a language model from scratch. Instead, we leveraged a pre-trained model that is called within our code. This means the heavy-lifting of learning language patterns was already done during pre-training. However, we can still fine-tune the model on our specific task and data (as described in the model tuning section). Fine-tuning adjusts the model's pre-trained weights slightly via additional training on our domain-specific data, effectively fitting the model to the nuances of our task.

To understand how our chosen language model can be adapted, it's helpful to know how BERT was originally trained[[devlin2018bert](#)]. BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based neural network architecture developed by Google that was pretrained on a massive corpus (over 3 billion words from sources like BooksCorpus and English Wikipedia). The model was trained to learn a deep understanding of language without supervised labels by using two self-supervised tasks:

1. Masked Language Modeling (MLM): BERT randomly masks some percentage of words in each input sentence and learns to predict these masked words using the surrounding context. Approximately 15% of the words are masked. This forces the model to understand context in both directions (hence "Bidirectional" in BERT's name) to fill in blanks correctly.
2. Next Sentence Prediction (NSP): BERT learns to predict whether one sentence naturally follows another. During training, it is fed pairs of sentences and tasked with determining if the second sentence is a logical continuation of the first. In 50% of training pairs the second sentence is the actual next sentence, and in the other 50% it

is a random sentence from the corpus. This teaches the model about discourse-level coherence across sentences.

Through MLM, BERT acquires rich bidirectional context understanding, enabling it to derive meaning based on both left and right context for any given word. Through NSP, BERT learns about sentence-to-sentence relationships, improving its understanding of how larger passages of text flow. Together, these training objectives produce a model with deep semantic understanding and the ability to generate context-dependent representations of language. After this pre-training phase, two model sizes were released: BERTBASE (110 million parameters) and BERTLARGE (340 million parameters).

DanishBERT Model. DanskBERT is a version of BERT that has been trained specifically on Danish language data. Created by Vesteinn Jónsson and made available via Hugging Face, it shares the same architecture as BERTBASE and was trained with similar objectives (MLM and NSP) but on a large Danish corpus (e.g. Danish Gigaword Corpus). In essence, DanishBERT learns the nuances of Danish vocabulary and grammar, while maintaining BERT's general ability to understand context. We chose this model for our project because our data is in Danish, and using a model with Danish language understanding should yield better results than using an English-trained model. By starting with a model that already “speaks” Danish, we give our classifier a strong foundation before fine-tuning it on the Bentax-specific task.

Implementation Approach 1 (Fine-Tuning DanskBERT): In the fine-tuning approach, we do not call a simple `model.fit(X, y)` as one might with a scikit-learn model. Instead, we employ Hugging Face's Trainer to handle the training process. The Trainer takes our pre-processed text-inputs and label-outputs and iteratively updates the BERT model's weights to fit our data. Under the hood, this process optimizes the model just like standard model fitting: it computes loss (error) on our training examples and adjusts the model parameters (using back-propagation) to minimize this loss. One important detail is that we use `AutoModelForSequenceClassification` to initialize the model – this adds a classification layer on top of the pre-trained BERT (specifically on the [CLS] token output) to predict our spare part categories. All layers of the DanishBERT are fine-tuned, meaning the model gradually learns from the Bentax text examples and their labels, modifying its internal representations to better suit the classification task.

Approach 2 (Training the Random Forest on BERT Embeddings): In the feature-extraction approach, “model fitting” refers to the training of the Random Forest classifier on the embeddings produced by BERT. Here, the BERT model remains unchanged (no fine-tuning), and we use scikit-learn's `RandomForestClassifier` with the embeddings as input features. We chose a Random Forest as our classifier to “start small and simple.” This algorithm is robust, easy to use, and requires relatively minimal hyperparameter tuning compared to

```
166 rf_classifier = RandomForestClassifier(  
167     n_estimators=200,  
168     random_state=42,  
169     class_weight='balanced', # Håndterer ubalancerede klasser  
170     n_jobs=-1 # Brug alle CPU kerner  
171 )  
172 rf_classifier.fit(X_train_embeddings, y_train)
```

Figure 6.11: Enter Caption

other algorithms. In our implementation, we set the Random Forest to use 200 decision trees (`n_estimators=200`), and we enabled `class_weight='balanced'` to automatically adjust for class imbalances (so that classes with fewer examples are given more weight during training). The training process involves calling the classifier's `fit` method on the training embeddings and their corresponding labels:

Each decision tree in the forest learns a series of if-else rules on different subsets of the data. The ensemble of trees then votes to determine the final predicted class for a new sample. Training is complete when the Random Forest has built all trees and they have collectively adapted to the patterns in the training data. This approach is fast to train and easy to interpret – for instance, we can extract feature importances to see which dimensions of the BERT embedding most strongly influence the classifier's decisions.

Model Tuning

Method Model tuning refers to the process of optimizing a model's performance by adjusting its hyperparameters – the settings that govern the learning process but are not learned from the data itself. Hyperparameters include things like the learning rate (how quickly a model updates its weights), batch size (how many samples are processed at once during training), number of training epochs, optimizer type, or regularization factors. Unlike model weights (which are learned during training), hyperparameters must be chosen before or during training, and their values can significantly impact the model's ability to generalize.

In the context of deep learning and LLMs like BERT, tuning often focuses on selecting appropriate values for parameters such as the learning rate, batch size, number of epochs, and dropout rates. For example, a learning rate that is too high might cause the training process to diverge (the model overshoots optimal weights and fails to converge), while a learning rate that is too low can result in very slow training or getting stuck in a suboptimal solution. A large batch size can stabilize training but requires more memory and can potentially lead to poorer generalization, whereas a small batch size may introduce more noise but sometimes helps escape local minima. The number of training epochs needs to be balanced to avoid underfitting (too few epochs) or overfitting (too many epochs).

For a fine-tuned transformer model like BERT, hyperparameter tuning is particularly sensitive because these models are complex and our task-specific dataset may be relatively small. Careful tuning (especially of the learning rate and number of epochs) is critical to squeeze out the best performance without overfitting.

In classical machine learning models such as Random Forests, different hyperparameters are relevant. Key hyperparameters for a Random Forest include the number of trees in the forest, the maximum depth of each tree, the minimum samples required to split a node or be a leaf, and so on. These settings control the model's complexity. For instance, too many trees or very deep trees might model training data noise (overfitting), while too few trees or shallow trees might not capture enough signal (underfitting).

Hyperparameter tuning is typically done via a validation process. Common strategies include grid search (trying combinations from a predefined set of values), random search (trying random combinations within ranges), or more advanced methods like Bayesian optimization. The goal is to find a combination of hyperparameters that yields the best performance on a validation set (data not seen during training), which is a proxy for how well the model will perform on truly unseen data. Performance is measured with appropriate metrics (such as accuracy or F1-score for classification). Proper tuning strikes a balance between underfitting and overfitting, ensuring that the final model is robust when deployed on real-world data.

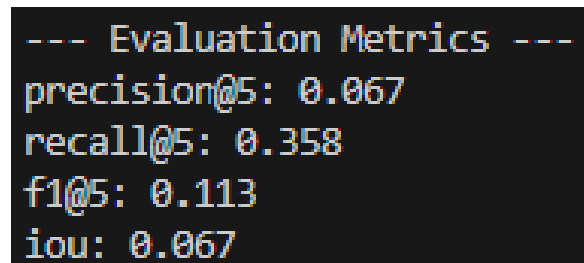
Implementation In our project we tuned each approach's hyperparameters to improve performance:

Approach 1 (Fine-Tuned BERT) Hyperparameters: When fine-tuning the Danish BERT model on the Ben-tax data, we had to configure several training hyperparameters. We used Hugging Face's Trainer with a specified learning rate, batch size, and number of epochs (based on initial trials and recommendations for fine-tuning BERT on small datasets). We also employed early stopping to prevent over-training: if the model's performance on a validation set stopped improving, training would be halted to avoid overfitting. The Hugging Face Trainer streamlined much of this process. Notably, `AutoModelForSequenceClassification` was used to automatically set up a classification layer, and the Trainer handled details like shuffling data each epoch, computing the loss and gradients, and performing optimization steps. We adjusted the learning rate carefully (BERT often requires a relatively low learning rate for fine-tuning) and chose an epoch count that allowed the model to see the data multiple times without memorizing it. All of these choices were guided by common practices in fine-tuning transformers and by observing the model's performance on validation data.

Approach 2 (BERT + Random Forest) Hyperparameters: For the Random Forest classifier

using BERT embeddings, we tuned more traditional parameters. We set the number of trees to 200 (`n_estimators=200`) as a balance between model complexity and training time. Having a substantial number of trees generally improves performance up to a point, but returns diminish beyond a certain number. We also configured `class_weight='balanced'` in the Random Forest. This hyperparameter automatically scales the weight of each class inversely proportional to its frequency in the training data, which helps the model pay sufficient attention to minority classes in our imbalanced dataset. Other Random Forest parameters (such as tree depth or splitting criteria) were left at their default reasonable values, since Random Forests tend to perform well out-of-the-box and we wanted to keep the solution simple. In our experimentation, this “small and simple” configuration was effective and did not show obvious signs of underfitting or overfitting, so we did not perform an exhaustive grid search.

Overall, the tuning process for both approaches was conducted with the goal of making each model perform reliably on unseen data. For the fine-tuned BERT, that meant carefully controlling the training process with an appropriate learning rate and stopping criteria. For the Random Forest, it meant choosing sensible defaults (like a sufficient number of trees and balanced class weights) to ensure robust performance without over-complexity. By tailoring the hyperparameters to each approach’s needs, we improved the models’ generalization while keeping the solutions practical and interpretable for our stakeholders.



```
--- Evaluation Metrics ---
precision@5: 0.067
recall@5: 0.358
f1@5: 0.113
iou: 0.067
```

Figure 6.12: Enter Caption

6.6 Results Experimental Phase

6.6.1 Data approach

[present results of data approach here]

also SMOTE results!

6.6.2 Text approach

TF-IDF

[present results of tests here]

6.6.3 Sentence Transformers

Results

6.6.4 LLM

[present results of tests here]

6.7 Discussion of experimental phase

6.7.1 data

[discuss data approaches]

6.7.2 text

[discuss text approaches] **old:** Rejection of LLM and Implementation of TF-IDf

Although LLM offer powerful capabilities for understanding and generating natural language, we chose not to use them for this task due to several practical considerations[spot2022llmdisadvantages]. LLMs typically require significant computational resources and can be more complex to fine-tune or interpret, especially for domain-specific classification tasks with structured labels like product IDs. Additionally deploying LLMs can introduce latency and increase costs, which was not aligned with the projects scope and constraints.

Instead, we implemented a Term Frequency-Inverse Document Frequency (TF-IDF) approach, which is both efficient and interpretable. TF-IDF transforms the input text data, specifically, the Instructions field in the dataset, into numerical vectors that reflect the importance of words within document relative to the entire corpus. We used uni grams and bi grams with a maximum feature limit of 5000 and applied a Danish stopwords list to reduce noise. Then these TF-IDF vectors were inputted into a X model to predict the corresponding product ID. This setup allowed us to build a fast transparent classification pipeline suitable for our use case.

6.7.3 conclusion of experimental phase

[choose methods for final approach] chosen: no LDA or LSA no smote Whole dataset approach

also TFIDF was chosen in combination with a linear SVC.

this is further implemented in final approach where the final model is further described and explained

6.7.4 Choice of classifier

Some machine learning models are better suited than others for datasets that are sparse and have many features, such as those produced using TF-IDF. Tree-based models like Extreme Gradient Boosting and the Random Forest Classifier may struggle when the features are spread thinly across many dimensions.

In contrast, a model like the Support Vector Classifier are often a better fit for sparse data, as discussed in 5.2. These models work well when the data can be separated by a straight line or plane and are less affected by the lack of dense patterns.

6.7.5 Choice of Classifier

Some machine learning models are better suited than others for datasets that are sparse and have many features, such as those produced using TF-IDF. Tree-based models like Extreme Gradient Boosting and the Random Forest Classifier may struggle when the features are spread thinly across many dimensions and when the information is sparsely distributed across thousands of potential input terms.

In contrast, a model like the Support Vector Classifier is often a better fit for this kind of data, as discussed in Section 5.2. SVC works well when the data can be separated by a linear boundary and is less affected by the lack of dense or structured feature relationships. Its margin-maximizing approach also gives strong generalization in high-dimensional spaces.

[Kan måske bruges som en god overgangstekst] Based on these theoretical and empirical considerations, TF-IDF in combination with a linear SVC was selected as the final approach. This is further implemented in the following chapter.

Chapter 7

Final approach

This chapter describes the chosen design and model used to solve the problem based on the problem .

7.1 Design of the model

The model is designed based on the knowledge acquired from the experimental phase. the flowchart in figure 7.1 shows the design of the flowchart. There are 2 main sections, preprocessing (lightgreen) and modeling (blue). The following sections in this chapter will describe each step in detail.

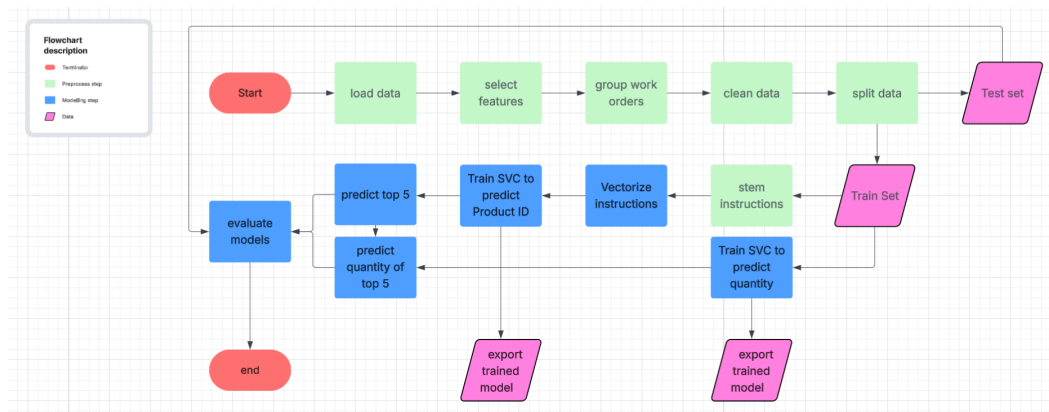


Figure 7.1: Flowchart showing the design of the final model

7.2 Data selection

As built on the knowledge acquired in the earlier phase, the following features were selected:

- *Work Order*
- *Instructions*
- *Product ID*
- *Quantity*
- *Primær Asset Produkt*
- *Work Order Type*

7.2.1 Feature - Work order

These features were chosen based on relevancy, usability and availability. which are further explained in subsections below.

The first feature included in the pipeline is the Work Order. As discussed in the earlier chapters on problem analysis and the experimental phase, this feature does not contain any direct information relevant to the nature of the repair and was therefore not used to train the machine learning model.

Instead, the Work Order is used to group rows that belong to the same repair job. A single work order may involve multiple spare parts and typically shares the same instruction text across all associated rows. By grouping rows with the same work order number, it becomes possible to consolidate all relevant parts and descriptions into a single entry, representing one complete repair.

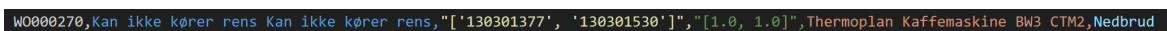


Figure 7.2: Example of a combined work order

As shown in Figure 7.2, each row in the raw dataset originally represented a single spare part (e.g., Product ID, shown in yellow). These rows appeared scattered throughout the dataset, even when they shared the same work order number and instruction. By grouping the data by work order, we create a clearer structure where each repair is treated as a single, unified record containing all associated parts.

This grouping step is crucial for consistent preprocessing. For example, consider the following simplified case:

Work Order 1: *"Milk is not working and water is leaking."* → [Product 2, 1x]

Work Order 1: *"Milk is not working and water is leaking."* → [Product 4, 1x]

Both entries belong to the same work order but appear on separate lines in the raw data. If, during preprocessing, a decision is made to remove all rows containing Product 4, this would result in an incomplete representation of the repair, only one product would remain, while the instruction clearly indicates two issues that likely required two spare parts.

Therefore, grouping work orders before filtering or modifying features ensures that repairs are preserved as complete units, reducing the risk of partial or misleading records.

Finally, it's worth noting that the Work Order number can potentially be used to trace a repair back to the customer. However, no other personal or identifying information was included in the final dataset, helping to preserve user privacy.

In the dataset after grouping resulted in: 34227 unique Workorders in total.

7.2.2 Feature - Work order type

The next step is part of clean data steps, also seen in the flowchart earlier represented in figure 7.1. The Feature work order type was used to include only the breakdowns that are solved in-store. This was done by selecting rows where Work Order Type equals 'Nedbrud'. This step helps to make sure that the model is working with actual customer-reported faults. After this step, the data had removed 16195 work orders, resulting in 18032 workorders left.

7.2.3 Feature - Primær Asset Produkt

In the problem analysis, the decision was made only to include coffee machines made by the Thermoplan Brand. This meant that these coffee machines needed to be filtered out. The machines in the dataset included where thermoplan machines BW3 and BW4. These were filtered using a pattern as seen in the code snippet in figure 7.3.

```
PATTERN_BW = r"\bBW[34]\w*\b"
```

Figure 7.3: Code snippet showing the pattern used to filter relevant coffee machines

which was then implemented in the code as seen in figure 7.4.

```
96 df_filtered = df_filtered[
97     df_filtered[ASSET_PRODUCT_COL].str.contains(PATTERN_BW, regex=True, na=False)
98 ]
```

Figure 7.4: Code snippet showing the filter used to filter relevant coffee machines

This resulted in 6223 unique work orders left.

Primary Asset Product is one of the two features included in the model training. This choice is based on the fact that many product IDs are specific to certain coffee machine models, as illustrated in Figure 7.5.

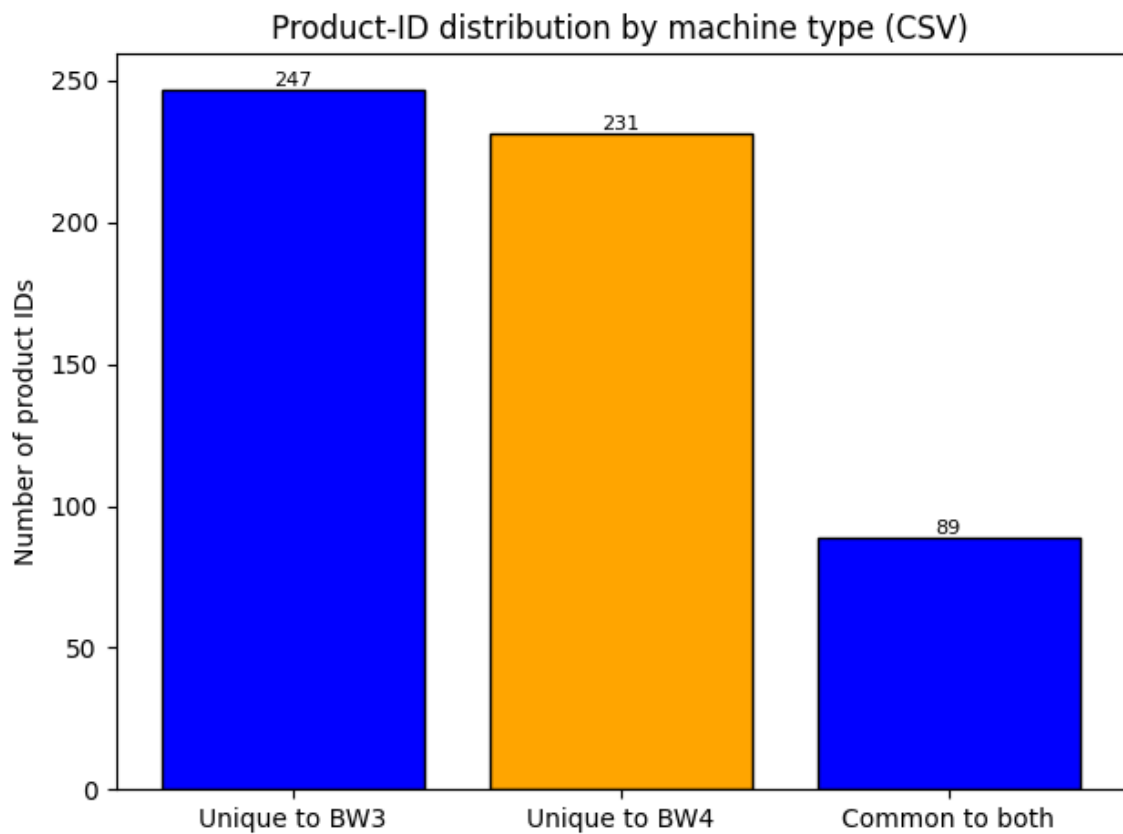


Figure 7.5: Unique product id for each Primær Asset Produkt

7.2.4 Feature – Instruction

The second feature used for model training is the Instruction field, which contains the problem description provided by the customer. An English-translated example would be:

“Milk is not working and water is leaking.”

This feature is important to enable the model to predict the correct spare part based on the described issue. Without this textual information, it would be extremely difficult for the model to make accurate predictions using only the Primary Asset Product.

As a result, 15 work orders were removed from the dataset due to missing instructions, leaving a total of 6,208 work orders for training and evaluation.

7.2.5 Feature – Product ID

The Product ID represents the spare part(s) used for a repair and is the target variable the model is trained to predict.

During preprocessing, a check was performed to ensure that no entries with missing product IDs were included in either the training or test sets. Since all work orders contained a valid product ID, no entries had to be removed in this step.

7.2.6 Feature – Quantity

The Quantity feature describes how many units of a particular spare part were used to complete a repair. To be able to predict a repair as a whole, the decision was made to include quantity so the model not only is able to predict the part, but also how many needed. As shown in Figure 7.6, most repairs required only a single unit of each part.

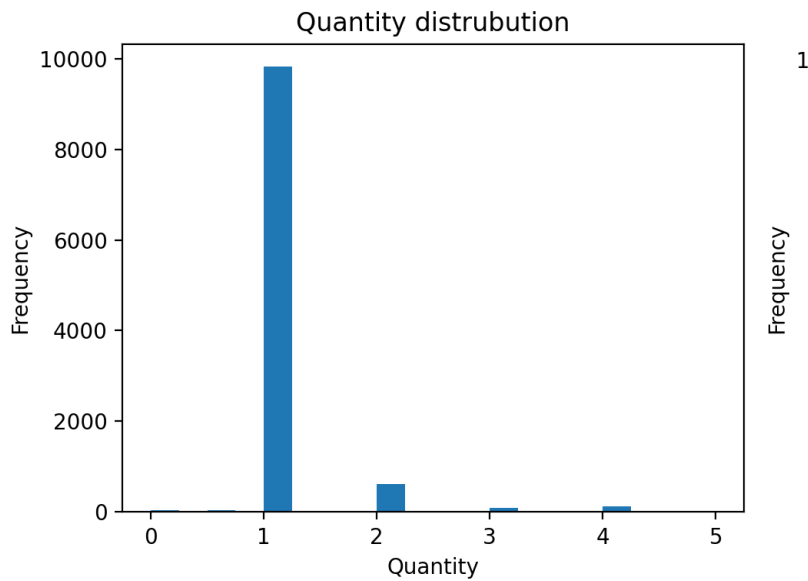


Figure 7.6: Distribution of spare part quantities used in repairs

A total of 29 work orders were found to have missing quantity values. Instead of removing these entries, it was decided to apply median imputation. Although one might argue for removing rows with missing values, imputing them ensures that the pipeline remains robust and generalizable, especially useful for future applications of the model where similar missing values might occur.

The choice of the median as the imputation method was based on the skewed distribution of the data, where most values are low and concentrated around 1. With 6,179 work orders containing valid quantity values, the median is a reliable and representative statistic. The resulting median value was 1, which was then rounded to an integer before being used to fill in the missing values.

7.2.7 Min samples per part

During the experimental phase, it became apparent that some spare parts—represented by specific Product IDs—occurred only a handful of times in the entire dataset, which spans approximately five years of repair records.

These rare parts can introduce noise into the data and pose significant challenges for the model. Since machine learning models rely on patterns learned from historical data, having too few examples makes it difficult for the model to generalize or make accurate predictions when encountering similar situations in the future.

Moreover, including very infrequent Product IDs increases the risk of overfitting, where the model memorizes specific examples rather than learning generalizable patterns. It also makes the evaluation metrics less meaningful, especially if certain classes are represented by only a few samples.

To address this issue, all work orders containing a Product ID that appeared fewer than five times in the entire dataset were removed. This threshold was selected to balance two goals: preserving as much data as possible while filtering out classes that are too infrequent for the model to learn from effectively.

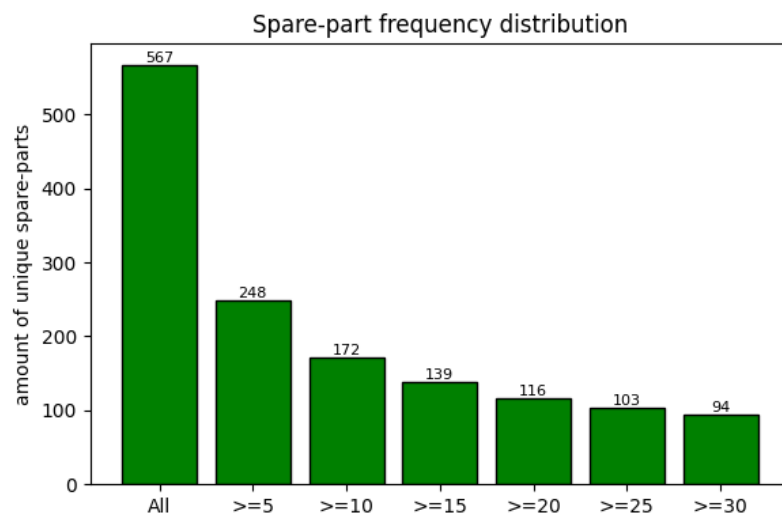


Figure 7.7: Distribution of 'rare' spare parts at different thresholds

As illustrated in Figure 7.7, a significant number of spare parts appear fewer than five times, supporting the decision to exclude these rare cases from the training set.

7.2.8 Summary of Features

Table 7.1 presents a summary of the preprocessing steps and their impact on the number of work orders.

Table 7.1: Overview of filtering steps and remaining work orders

Filtering Step	Work Orders Removed	Remaining Work Orders
Initial work orders (grouped)	–	34,227
Filter: Work Order Type == Nedbrud	16,195	18,032
Filter: Asset product matches BW[34c] *	11,809	6,223
Filter: Non-empty Instructions	15	6,208
Filter: Non-empty Product ID list	0	6,208
Imputation: Missing Quantity values (median)	0	6,208
Total remaining		6,208

After these preprocessing steps, the data was split into training and test sets using an 80/20 ratio, as shown in Table 7.2. This resulted in a clean and focused dataset, well prepared for training and testing the machine learning model.

Table 7.2: Train-test split of the filtered dataset

Dataset	Number of Work Orders	Number of Features
Total work orders after filtering	6,208	6
Train dataset	4,966	6
Test dataset	1,242	6

7.3 Modelling

After the data is cleaned and ready for

[here we describe how we trained our model]

for example: as discussed in the [reference to discussion of the experimental phase] the decision was made to continue with a linear support vector machine in combination with tf-idf.

tf-idf uses quite a high dimension space, as described in [ref to explain of tf-idf], which results in data segmentation that easily can be split using a linear classifier.

Tf-idf

the decision was made to use tfidf. there are a few parameters to edit. as seen earlier in table 6.1.

In figure 7.8 is visualized how the decision was made to choose:

max features = 20 000

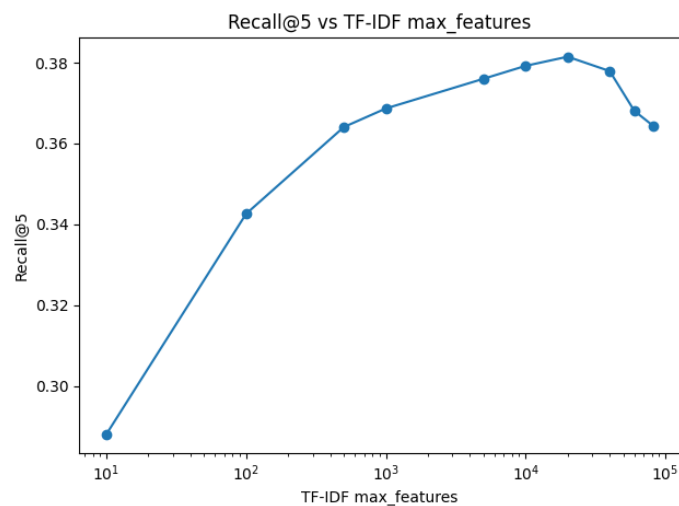


Figure 7.8: figure showing recall at different max features

7.3.1 evaluation

[chosen metrics]

Chapter 8

Results

[our results? how was the performance? etc etc] [Final Approach only]

8.1 Introduction

This section presents the results of the final machine learning model developed to predict the most relevant spare parts for Thermoplan coffee machines based on historical service data. The model's performance is evaluated using a range of standard metrics suitable for multi-label classification with Top-5 output. The results include recall, precision, F1-score, and label agreement, providing an overall overview of the model's ability to suggest relevant spare parts based on free-text fault descriptions.

8.2 Performance Matrix

To evaluate the performance of the final machine learning model, a range of standard multi-label classification metrics was computed on the test set. The evaluation focuses on the model's ability to return a Top 5 ranked list of spare parts for each input instruction.

The test results indicate that the model demonstrates a reasonable ability to identify relevant spare parts within a large label space. With a top 5 recall of 45 percent, the model succeeds in including at least one correct part in nearly half of the predictions. This suggests that the model is suitable as a decision support tool rather than a fully autonomous solution, as technician oversight is still required to select the appropriate part.


```
===== Test-set metrics =====  
precision@5: 0.145  
recall@5: 0.450  
f1@5: 0.219  
hamming: 0.979  
weighted: 0.134  
partial_cov: 0.450  
iou: 0.129  
quantity_acc: 0.951
```

Figure 8.1: Test results

The relatively low precision implies that many of the suggested parts are not actually needed. However, this is an expected trade off, when optimizing for recall rather than precision a strategy that aligns with the practical goal of avoiding part shortages rather than strictly limiting the number of suggestions.

The high hamming score (0,979) and quantity accuracy 95 percent indicate that the model generally predicts a reasonable number of parts and rarely makes completely irrelevant suggestions. Other metrics, such as the F1-score and Intersection-over-Union (IoU), highlight areas for improvement, particularly in terms of predicting the full set of correct labels.

8.3 Analysis

8.4 Error Analysis

8.5 Visualizations

Ideas: 3-d Vector Space w. clustering?

8.6 Evaluation

Acc. Weight Propa-score, 'vægtet sandsynlighed'. Recall (and 'partial coverage', same-same) Precision F1-score , top-5 IOU - Intersection-over-union Quantity acc. - recall og Precision ind for de forskellige klasser, de vi har en kæmpe overvægt af klasse-1 (hvor der kun skulle bruges 1 reservedel til en opgaven) - spørgsmålet er, er modellen god til sin prediction eller er den blot god til at "sige 1". Det store spørgsmål indenfor machinelearning - forstår modellen det eller gætter den bare

Hamming-score

8.6.1 Type of Faults

Strong Predictions

Weak Prediction

Chapter 9

Discussion

Introduktion: 1. diskussion af resultater vil give et grundlag for at kunne analysere hvorfor vores eda, metoder, osv. har været godt/eller dårligt - det kan jo bedømmes ud fra resultaterne.

9.1 Discussion of results

Det store spørgsmål: Er det dataen eller er det vores metoder.? Recall Precision IOU

9.1.1 Has the Model Leaned or Is It Just Guessing?

```
==== Test-set metrics ====
precision@5: 0.145
recall@5: 0.450
f1@5: 0.219
hamming: 0.979
weighted: 0.134
partial_cov: 0.450
iou: 0.129
quantity_acc: 0.951
```

Figure 9.1: Cases with up to 5 spare parts

```
==== Test-set metrics ====
precision@5: 0.148
recall@5: 0.464
f1@5: 0.224
hamming: 0.970
weighted: 0.140
partial_cov: 0.464
iou: 0.132
quantity_acc: 0.945
```

Figure 9.2: Cases with up to 10 spare parts

The evaluation results suggest that the model has learned some patterns from the data rather than merely guessing, considering all evaluation metrics, but seems to fail in some important evaluations. Quantitatively, the model's Recall@5 is in 5-spare-part cases at 45% and in 10-spare-part cases at 46.4%, meaning in approximately 45% of cases the correct spare part is present among the top five suggestions. This is traditionally not considered as a strong indicator that the model has learned anything. A recall@5 at 70% would have been much more satisfying. The model's quantity prediction (how often it predicts the correct number of each part needed) is high, reflecting that it can identify the right number of replacements. Since most cases typically require one unit, as most repairs require single parts (60%), it can be discussed whether this is a case of overfitting or a real understanding from the model on this parameter.

9.1.2 Strong vs. Weak Prediction Scenarios

One thing we do know for sure, is that the quality of the model's predictions varies depending on the clarity and specificity of the fault description. Certain scenarios consistently yield strong predictions, while others remain challenging.

Strong Prediction Scenarios

When the input description contains specific symptoms, refers to particular functions, and/or clear context about the machine, the model's predictions are very relevant. For example, descriptions like "kaffemaskine lækker vand ved skyl og steam", or "Larmer i kværn. Blinker i display..." are likely to trigger suggestions such as an "O-ring" for the steam valve, or grinder burrs/gear that commonly cause those loud symptoms. **Like-wise some error codes seems to be predicted with the same relevance. These strong code errors are predicted with higher precision due to the code error refers to a very specific solution, or a scenario where a small number of spare parts can be relevant.** Another strong factor that clearly strengthens predictions is if the needed part is a frequent part in the historical data. The model has in these cases many examples to learn from, so it confidently recommends those parts when the symptoms match. ** Clear mention of the machine type or model can also help, since certain models have known recurring issues. The model can leverage that context if it was part of the input**

Weak Prediction Scenarios

Predictions are less accurate when the description is vague, unusual, or involves multiple issues at once. Vague descriptions such as; "Kontakt for kode til at komme ind. Kan ikke

lave kaffe" or "Flere maskiner melder temperatur for høj. Vil ikke starte og kommer med fejlmelding" provide too little information for a precise match, leading the model to rely heavily on general frequency (suggesting the most commonly replaced parts as a catch-all). In these cases the correct part might be something uncommon that the model doesn't consider without a clear clue, and therefor might miss it in top-5. Similarly, if the actual fault lies with a rare part that hardly ever failed in training data, the model is likely to overlook it. For example, is a very rarely failing sensor is the true culprit, the model might not include that sensor in the top predictions simply because it had insufficient examples to learn that association. Another weak scenario is when the 'instructions' lists multiple simultaneous problems, the model might correctly suggest a part related to one of the issues but not cover all aspect. **Furthermore, multiple described errors/issues/problems in an 'instruction' can possibly confuse the model as it potentially can become confused about which spare part actually belongs to the correct symptoms as described in the 'instruction', this is at least our hypothesis.** In such complex cases, the top-5 recommendations might include only partial solutions or more generalized guesses. This highlights a limitation: the model doesn't truly reason through multiple fault causes; it treats the input as one combined text and returns the parts that collectively have the highest likelihood of being needed, which might not perfectly match a multi-fault situation.

Overall, the model is robust in well-defined scenarios (clear, common and single issues) but shows weakness in ambiguous or novel situations. This behavior is typically for machine learning models trained on historical data, They shine where the new case looks like what they've seen before, and falter when encountering something significantly different or under-specified.

9.1.3 Prediction Table

to be continued

9.1.4 Performance Against Requirements

Spare Part Recommendation The model fulfills the core functional requirement of providing a ranked list of up to five likely spare parts for a given fault description. The relatively low recall@5 (app. 45%) indicates that in the majority of cases it did not predict the actual spare part needed for the given 'instruction' at a top-5 suggestion.

Handling Ambiguity and Uncertainty: The system does return a ranked list of possible spare parts and indicate low or high evaluation. - forstå ikke helt denne

Danish Language Support: The maintenance logs and input descriptions are in Danish (often with informal phrasing and technician shorthand). The model handles Danish text as well as it can on our specific data. This is achieved through text pre-processing techniques like Danish stemming (reducing words to their root form) and removal of Danish stopwords, which the implementation includes. As a result, the model can interpret variations of Danish terms, for example: "mælkeskum" and "skummer ikke" similarly to "mælkeskum problem". The model's strong performance on real Danish service descriptions demonstrates that language support is adequate. Customer service can write the issue in their natural Danish wording (to a certain extent) and the system will still make sense of it.

Semantic Understanding & Natural Language Input: The model exhibits a basic semantic understanding of the issue, though it is largely based on learned correlations rather than true comprehension. It goes beyond simple keyword matching in cases with higher historical data, such as milk related problems. However, its understanding is not deep in a human sense; it cannot reason about problems it hasn't seen before or fully grasp context if it's implied rather than stated. It mainly relies on patterns observed in the historical data. (e.g. the word "lækker" often co-occur with replacing a gasket or valve). In practice, this level of semantic handling is sufficient for many routine issues, but it may misinterpret phrases that differ from the training examples. Overall, the requirement for semantic text comparison is partially met: The model can match issues to likely parts using textual similarity and co-occurrence patterns, but it isn't infallible in truly understanding novel phrasing.

Relevant Case Retrieval: The system does not.. - har vi helt glemt denne?

Critical Reflection and Limitations

Overfitting & Bias Toward Common Faults: The model naturally exhibits a bias towards frequent parts and common failure modes present in the historical data. This is a common outcome in machine learning classification when some classes (spare parts) dominate the dataset. The model has effectively learned "popularity" but not necessarily "reality". The model knows which parts are most frequently replaced and tends to favor them, especially when the input description is not very distinctive. This bias is double-edged: it helps in many cases (since common failures do account for a large portion of issues, the model's priors are useful), but it can lead to over-prediction of popular parts (overfit). The risk of overfitting the training data was minimized through careful model design and regularization. Techniques like limiting the model complexity and using regularization (L2 penalties or dropout in training) were applied, ensuring that the model did not simply memorize the training examples.

9.2 Discussion of Methods

[old: below here]

9.2.1 TF-IDF

kasper: we dont use predictive maintenace, so this needs to change?

Advantages TF-IDF can be useful in the initial analysis of service-related text, where the objective is to identify patterns in fault descriptions linked to specific spare part replacements. The method highlights technically relevant and specific terms while down-weighting generic expressions that hold limited analytical value. This leads to a more focused and interpretable representation of textual data, which is essential for developing models aimed at predicting component failures and supporting targeted maintenance planning[[goldengrisha2020](#)]. Additionally, TF-IDF is simple to implement and computationally efficient. It does not require large annotated datasets or complex pre-processing, making it a cost-effective entry point for working with unstructured service logs[[goldengrisha2020](#)]. Its vector-based representation is well suited for tasks such as document similarity, clustering of error reports, and feature extraction in traditional machine learning pipelines.

Disadvantages Despite its advantages, TF-IDF also presents several important limitations, especially in relation to the project’s ambition of building an intelligent, context-aware decision support system. Most notably, TF-IDF lacks the ability to capture semantics, as it treats each word as an independent unit and disregards word order, syntax, and sentence structure. This makes it difficult to interpret the meaning of compound expressions or distinguish between subtle but significant variations in fault descriptions[[goldengrisha2020](#)]. Similarly, technical multi-word expressions such as “brew unit blocked” may be broken into individual words, potentially resulting in the loss of their domain-specific meaning[[goldengrisha2020](#)].

TF-IDF also suffers from the so-called curse of dimensionality. The resulting feature vectors are as long as the size of the vocabulary, often leading to sparse and memory-intensive representations[[goldengrisha2020](#)]. While this may be manageable in small classification tasks, it can become problematic in large-scale applications such as real-time prediction or unsupervised clustering of service records. Lastly, TF-IDF does not support dynamic learning or generalization across linguistic variations, which limits its adaptability in more advanced use cases[[goldengrisha2020](#)].

9.2.2 Sentence Transformer

The use of sentence transformers in this project was motivated by the desire to explore a more semantically informed alternative to traditional vectorization techniques such as TF-IDF. Instead of relying on frequency based features, the goal was to represent each instruction as a dense vector that captures its meaning within a broader semantic context.

Despite the theoretical strengths of semantic embeddings, the model did not achieve better performance than simpler TF-IDF-based classifiers. The evaluation results showed a recall score of 0.358, slightly lower than the TF-IDF approach.

Several factors may explain the modest results. Firstly, the model was trained on general-purpose semantic similarity tasks and was not fine-tuned for technical Danish texts. This likely reduced its ability to distinguish between domain-specific fault descriptions. Secondly, the instruction texts were sometimes short, informal, and inconsistent, limiting the benefit of using context aware embeddings.

Although the performance was lower than TF-IDF it did come close, which provide insight into the potential and limitations of using pre-trained language models without domain adaptation. It highlighted that general language understanding does not always translate into strong performance on specialized classification tasks.

In conclusion, while the sentence transformer did not outperform TF-IDF in this specific case, it served as a useful intermediate step in the experimental phase. It illustrated an important methodological direction for future work, especially in cases where fine tuning or domain specific data becomes available.

9.2.3 LLM, Lanja

The use of LLMs such as BERT was explored only to a limited extent in this project, as the method lies outside the formal scope of the course curriculum. Nevertheless, the approach invites reflection on its strengths and limitations in relation to text classification in technical and unstructured domains.

A key advantage of LLM-based methods is their ability to capture semantic and contextual patterns directly from raw textual data. This reduces the need for manual feature engineering and makes the method particularly suitable for tasks where language usage and word order play a crucial role in comprehension. In cases involving unstructured work order descriptions, as in this project, transformer architectures have the potential to provide improved generalization and accuracy compared to traditional approaches.

At the same time, the use of LLMs introduces significant challenges. Models like BERT are resource-intensive in terms of both computation time and memory requirements. Furthermore, their application demands greater technical understanding of deep learning concepts, including optimization strategies, regularization techniques, and hyperparameter tuning. Within the constraints of a student project—limited by both time and computational resources—these requirements pose practical obstacles.

Two model configurations were tested: fine-tuning a pre-trained Danish BERT model, and a hybrid approach where BERT was used as a feature extractor in combination with a Random Forest classifier. The second approach—without any modification to BERT's internal weights—yielded competitive results at significantly lower complexity. It was easier to implement and required fewer resources, while still leveraging the linguistic representations encoded by BERT.

Since the use of LLMs was not a central focus of the project, the method was not explored in depth. However, it is likely that more extensive experimentation—for example, through additional training cycles, learning rate adjustment, or the use of more sophisticated classification layers—could have further improved performance. Among the approaches evaluated, the combination of BERT embeddings with a classical machine learning model demonstrated the greatest practical potential under the given circumstances.

Overall, the findings suggest that LLM-based technology, even in a simplified form, can serve as a powerful tool for text classification. Realizing its full potential, however, requires deeper technical expertise and access to adequate computational resources. In an educational context such as this—where the emphasis lies primarily on traditional machine learning techniques—the hybrid approach emerges as a balanced compromise between theoretical capability and practical feasibility.

9.2.4 Discussion of relevance

The task of bringing the correct spare part to a repair call is still handled by technicians who rely on personal experience and intuition, so outcomes vary from technician to technician. A data-driven prediction model that analyzes historical repair data can uncover patterns a human might miss and deliver more consistent, evidence-based decisions. Each repair would then use data from thousands of past cases, reducing variation in service quality and lessening dependence on experts.

The current model does not yet outperform technician intuition (recall is roughly 45), but it already detects meaningful links between often vague fault descriptions and the parts that are needed to fix the problem. With a larger and better structured dataset, its performance is expected to climb. Deploying the model now would still offer practical

value: technicians could review a ranked list of likely parts before arriving at the repair site and bring extra likely parts and thereby potentially decrease the number of follow-up visits. Just as importantly, the model learns symptoms of faults of many spare parts which can later be extended with error codes and real-time IoT data, enabling true predictive maintenance.

Advantages of a data-driven approach:

- Reduces dependence on individual technician expertise by combining five years of collective data.
- Suggests likely parts before a technician departs, lowering returns for the right spare part.
- Delivers consistent service quality across the department.
- Scales easily without a proportional rise in expert troubleshooters.
- Offers a foundation for a more data driven approach to maintenance.

Disadvantages of a data-driven approach:

- Training data is still sparse and noisy, especially for rare parts.
- Present performance does not yet meet the requirements set in this paper
- Requires ongoing retraining and quality checks.
- Implementation and maintenance cost.

9.3 Discussion of ethics

[Kasper?]

[old:]

Chapter 10

Conclusion

Here we conclude things and the chapter number is 10.

10.1 Introduction

10.2 Problem Analysis

10.3 Methods and Implementation

10.4 Final Model

10.5 Results

10.6 Requirement Specification

[here we take a look back at our requirement specification]

old text: (Tænker vi skal bruge noget der direkte diskuterer anvendeligheden, altså giver det overhovedet mening eller er det bare noget der kan bruges til at vise et evt potentiale)

10.7

Appendix A

Appendix

Here is the appendix.

A.1 Bentax Service Documentation



Om Bentax

Hos os handler det om god kaffe: kaffebønner, kaffemaskiner, kaffebrygning, kaffeudstyr og rigtig god service til vores kunder. Vores fokus er professionelle løsninger til virksomheder i hele Danmark, både kontorer og institutioner, restauranter, konferencecenter og cafeer. Og vi kan levere alt der skal til, fra kaffe i bedste kvalitet til professionelle maskiner, - og ikke mindst den gode service, der holder din kaffeløsning i gang, hver dag.

Vores kultur er stærk og drevet af engagerede, ihærdige og professionelle medarbejdere, der alle bidrager til fællesskabet i Bentax. Og alle støtter op om vores værdier: hæderlighed, ærlighed og høj faglighed.

Bentax



Forslag 1: Prediktiv vedligeholdelse og optimeret serviceplanlægning

For os er det vigtigt at vores kunders kaffemaskiner altid er kørende. Vi har en bred vifte af kunder også kunder som lever af at sælge kaffe, derfor er det vigtig at vores maskiner har en høj opetid. Det kan være dog være omkostningstungt at holde maskinerne kørende, da vi nogle gange skifte dyre reservedele for at være på den sikre side.

Vores drift controller arbejder med at kigge ind i data for at finde mulige optimeringsmuligheder i forhold til vores serviceprogrammer, som hvor vi bruger færre omkostninger, men samtidig holder maskinerne kørende længere.

En ændring i et serviceprogram, kan tage op til et år inden vi kan se effekten af, og foregår på maskinniveau. Derfor kunne det være interessant med en intelligent og mere målrettet løsning.

Projekt

Udvikling af løsning til at kunne forudsige, hvilke reservedele der skal skiftes, og hvornår det bedst kan betale sig at udføre service. Ved anvendelse af historisk data om opgavetype, fejltype og reservedelsforbrug samt udnyttelse af IoT- data om maskinens aktuelle tilstand, skal systemet hjælpe teknikeren med at træffe beslutning om vedligeholdelse. Målet er at reducere omkostninger og samtidig opnå længere opetid for maskinerne.

- Forudsige, hvornår og hvilke reservedele der sandsynligvis vil fejle, og dermed hjælpe teknikeren med at lave forebyggende vedligehold.
- Komme med forslag til optimal serviceplan baseret på både historisk data og realtids IoT-data fra maskinen der reducerer behov for stort eftersyn ved i stedet at foreslå mindre, men mere målrettede, vedligeholdelsestiltag.

Udvidelse

Skal det også kunne bruges til **automatisk genbestilling af reservedele**, i den forstand , at der automatisk bestilles de reservedele der sandsynligvis vil være nødvendige i den kommende periode, baseret på forudsigelserne om deres levetid.

Automatisk generering af forebyggende service eller forslag til besøg.



Forslag 2: Real-time ankomstforudsigelse, samt fokus på CO2.

Bentax står overfor en stigende udfordring med at opretholde kundeloyalitet, da kunderne ofte oplever frustration og usikkerhed på grund af manglende information om, hvornår en tekniker præcist ankommer. Dette kan ikke blot påvirke kundetilfredsheden negativt, men også medføre en øget belastning på vores kundeserviceteam, som modtager hyppige opkald fra kunder, der ønsker opdateringer eller en præcis tidsramme for besøget. Desuden skaber denne usikkerhed en hektisk og uforudsigelig arbejdsdag for vores teknikere, hvilket kan påvirke deres effektivitet og trivsel

Forudsigelse af ankomsttid

En mere præcis forudsigelse af, hvornår vi ankommer hos kunden baseret på historisk data, trafikinformation, kundens SLA og åbningstider samt CO2-optimeringer.

- Historisk data fra tidligere besøg, fejltype, opgavetype, tidsforbrug osv.
- Trafikinformationer eks. I København
- Rute-optimering skal tænkes ind i forhold til at reducere CO₂-udledningen under kørslen, det vil sige, at den skal komme med forslag til den mest miljøvenlige rute.
- SLA og åbningstider: Den skal også kunne tage højde for, om kunderne har nogle ønsket besøgstider og åbningstider, eks. Hvis en cafe har primetime mellem 12-15, så skal vi ikke besøge dem der ved forebyggende service, men ved nedbrud er det hurtigst muligt.

Real-time forudsigelse

Den skal løbende ændres forudsigelse, så vi kan holde kunden opdateret løbende, det kan være:

- Når kunden bestiller service
- Ved automatisk genereret forbyggende service
- Når der sker ændringer i tidsplanen
- Når teknikere er på vej

A.2 Technical Theory