FernUniversität in Hagen
Faculty of Mathematics and Computer Science

Artificial Intelligence Group

Predicting Disorders of Service Station Components using Machine Learning-based Classification

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Praktische Informatik

submitted by
Jana Kaszyda

Prof. Dr. Matthias Thimm
Artificial Intelligence Group

Advisor: Prof. Dr. Matthias Thimm
Artificial Intelligence Group

Statement

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Yes	No
I agree to have this thesis published in the library.	☒	☐
I agree to have this thesis published on the webpage of the artificial intelligence group.	☒	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐
The source code is available under a GNU General Public License (GPLv3).	☒	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐

Nantes, 30/11/23 .....

(Place, Date) (Signature)

Abstract

Many industry domains benefit from machine learning models, which help recognize user, machine, or customer data patterns. In this work, we investigate the possibilities of using machine learning-based classification to predict critical hard- and software issues of on-site equipment of service stations based on the equipment logs. The data is available through cooperation with MADIC and was collected over the last two years on more than 2,000 service stations in France. We apply different classification methods such as K-nearest neighbors, Random Forest, Fully Convolutional Networks, and Residual Networks and compare their performance on test data from the same period as the training data and on entirely unseen data collected afterward. Furthermore, we investigate the impact of different sample selection approaches, data encoding methods, data imbalance, and the upsampling method SMOTE on the model performance.

Contents

1 Introduction **1**

2 Project Context **2**
2.1 Maintenance Strategies . . . 2
2.1.1 Preventive Maintenance . . . 2
2.1.2 Predictive Maintenance . . . 2
2.2 MADIC group . . . 3
2.3 On-site Equipment . . . 3
2.3.1 Payment Machines . . . 4
2.3.2 Fuel Tanks . . . 4
2.3.3 Fuel Dispensers . . . 5
2.4 MagView . . . 5
2.5 Available Data . . . 6
2.6 Project Goals . . . 8
2.7 Project Schedule . . . 8

3 Theoretical Background **9**
3.1 Methods used in Related Work . . . 10
3.1.1 Sequential Pattern Mining . . . 10
3.1.2 Frequent Chronicle Mining . . . 13
3.1.3 Change Point Detection . . . 15
3.2 Supervised Learning and Classification . . . 17
3.3 Selected Algorithms . . . 20
3.3.1 K-nearest Neighbors . . . 20
3.3.2 Random Forest . . . 22
3.3.3 Fully Convolutional Network . . . 23
3.3.4 ResNet . . . 26

4 Methodology **28**
4.1 Technical Environment of the Development . . . 29
4.2 Experimental Setup . . . 29
4.3 Data Preparation . . . 30
4.3.1 Overview and Characteristics of the available Data . . . 30
4.3.2 Data Preprocessing . . . 31
4.3.3 Data Encoding . . . 33
4.3.4 Sample Selection . . . 35
4.3.5 Dealing with Imbalanced Data . . . 37
4.4 Parameter Tuning . . . 38

5 Evaluation **38**
5.1 Train-Test Split . . . 39
5.1.1 Training on All Data . . . 39

5.1.2 Training on Unique Samples 42
5.1.3 Training on Randomly Selected Samples 45
5.2 New Data 48
5.2.1 Models Trained on All Data 49
5.2.2 Models Trained on Unique Samples 51
5.3 The Effects of SMOTE 53

6 Discussion **55**

7 Conclusion **57**

1 Introduction

Intelligent solutions using machine learning approaches are increasingly present in our everyday lives. The probably most famous examples are voice assistants and smart home solutions that guide us, especially in our private lives and are accessible to a broad range of users. However, professional environments also increasingly integrate such concepts into their business processes. The industrial sectors could benefit from those solutions as they would lead to more stability, faster processes, higher precision, and more comfortable work conditions for employees. This project focuses on integrating machine learning, more precisely classification methods, into maintenance processes at service stations. As an essential part of human transportation, service stations carry a big responsibility and must always be operational. Therefore, the maintenance of such stations is a sensitive topic as it needs to be as fast as possible to keep down periods short and economic loss low.

This thesis examines how machine learning approaches are suitable for making predictions based on data created by on-site equipment of service stations. While predictions are usually made based on sensor data, the available data for this thesis comes in the form of discrete log lines with predefined error codes, which is why the principal problem is interpreted as a classification task. The main objective of this project is to compare different classification approaches, apply them to the given data set, and evaluate their performance to decide whether one of the created predictive models is suitable for integration into the maintenance process of MADIC.

After a short introduction, Section 2 describes the project context and the available data from actual on-site equipment thanks to a collaboration with the French company MADIC group. The chapter briefly explains common maintenance strategies, why predictive maintenance has powerful advantages, the different equipment producing the data, and this thesis project's goals and future perspective.

Section 3 shows a detailed overview of the theoretical background, including approaches used in related works. Furthermore, the section describes the selected algorithms of the project: Random Forest, K-nearest neighbors (KNN), and the deep learning approaches Fully Convolutional Neural Networks (FCN) and Residual Neural Networks (ResNet).

Section 4 explains the methodology of the project, including an overview of the technical environment of the development, the Python libraries used, and the experimental setup. Furthermore, it describes the different steps of data preprocessing. The codebase for this thesis can be found on GitHub[1]. The data and trained models are available in a Google Drive folder[2].

In Section 5, we evaluate each classification method, each within a train-test-split and with entirely new and unseen data. Further on, the evaluation results are discussed in Section 6 before the thesis finishes with a conclusion and outlook in Section 7.

[1] https://github.com/jkaszyda/FailurePredictionServiceStations

[2] https://drive.google.com/drive/folders/1pePHq78y5BpTlUaORQy_C7dtRnEZReXo?usp=sharing

2 Project Context

The following section explains the project context and background, including a small overview of standard industrial maintenance methods, a description of the available data, the equipment creating this data, and the project's primary goals.

2.1 Maintenance Strategies

As the number of automation processes and mechanization in industrial companies increases, the need for maintenance activities also increases. Technological components can fail over time and need regular surveillance to enhance their lifetime and to keep downtimes short. When it comes to maintenance, the classic approach is corrective maintenance, in which technological units are monitored and repaired in case of malfunctions and signs of wear. However, as stated by Basri et al. [EIB17], those processes are often long, from the failure detection and the intervention schedule to the final repair. Furthermore, it causes more system failures and high repair costs as problems can occur anytime, and spontaneous planning is expensive. Therefore, modern industry often applies more efficient maintenance strategies like preventive maintenance and predictive maintenance. Such strategies lower the effects of unplanned failure and can also enhance the lifetime of technological components.

2.1.1 Preventive Maintenance

According to Basri et al. [EIB17], preventive maintenance unites routine maintenance activities on machines and facilities. Its principal objectives are to ensure technical functionality, the risk reduction of unplanned downtime, and to schedule maintenance activities before problems occur. Maintenance is usually performed during production time.

Based on tests and experiences with the equipment, this recommendation often concerns age-related problems and has a time-based schedule.

Even if preventive maintenance should be integral to all maintenance strategies, especially in domains where human security is critical, Edwards et al. [EHH98] explained that there are more optimal strategies than this. Furthermore, preventive maintenance is based on eventualities with which problems may occur in a specific period, which makes it a very cost-intensive maintenance method. It can also mean that spare parts in perfect condition get replaced because they have reached their average lifetime. Therefore, it cannot be the only maintenance strategy within a company.

2.1.2 Predictive Maintenance

Predictive maintenance is a more efficient alternative to compensate for those typical uncertainties of preventive maintenance. This maintenance approach aims to

predict how long a currently operational machine or equipment will still work correctly and when it will degrade. Its goals are reducing downtimes, determining when exactly a problem will occur, avoiding unnecessary, expensive, and ecologically problematic maintenance activities, and keeping maintenance activities as low as possible.

Its goals are the following: As described by Sipos et al. [SFMW14], predictive maintenance is based mainly on data analysis of the concerned equipment. This analysis determines which factors and patterns are possible indicators for imminent failures and allows for planning maintenance actions at the right time and before a potential breakdown occurs. Data often comes in the form of sensor information of physical components and gives a clear picture of what happens inside them. Having available sensor data also means that the concerned equipment is connected, allowing real-time monitoring. Thanks to modern solutions, companies can supervise all their production processes in detail to initiate planned maintenance activities when the data shows a previously identified pattern leading to a disorder.

2.2 MADIC group

This thesis project takes place in cooperation with the 1971-founded company MADIC group, one of the leaders in France in the energy industry. Among other activities, MADIC primarily focuses on the conception, construction, and maintenance of service stations and car wash.

The headquarters are in Nantes, France, but the company has several locations in Europe and other parts of the world, each with specialization. In total, the company employs around 1300 employees at 36 sites.

The most significant part of the present components, such as fuel tanks, fuel dispensers, payment machines, and pumps in a typical service station of MADIC, are self-produced. Therefore, the company offers many products and services related to this domain.

2.3 On-site Equipment

Typically, service stations in France are self-service stations without on-site employees. The client chooses one of the available terminals at which they validate their payment by card or by cash before the fuel dispenser gets unlocked, and the client can refuel their car.

Within the scope of this thesis project, the connected equipment of the service stations is of interest as it creates the data on which we will test different machine learning algorithms. Currently, this equipment range consists of the three following equipment types:

- Payment machines
- Fuel dispensers

- Fuel tanks

2.3.1 Payment Machines

As most French service stations are self-service stations, specific payment machines are installed on each fuel dispenser to allow the client to validate a secure payment transaction. The latest model is the APL 3.5. The payment takes place by card, but some stations offer cash payments. In this case, customers pay at another machine that prints the payment confirmation as a QR code. This code will be scanned by the APL 3.5 to unlock the fuel dispenser.

A small screen allows the interaction between the client and the payment machines. The client validates a card payment on this screen and chooses the fuel type. Even though the GUI of the machine is simple and easy to understand, the user automatically receives assistance from a computer voice explaining what to do in each step.

An APL 3.5 has, among others, two modular and separately exchangeable components. These are the card reader and the terminal. The card reader reads the bank card information, and the terminal guides the user through the transaction validation process, including the payment validation and the selection of the fuel type.

Each payment machine follows specific laws and regulations as it contains sensitive data and is a popular target of vandalism and data theft. Only authorized engineers can realize manipulations and maintenance on these machines under specific safety measures. Figure 1 shows a typical payment machine MADIC produces.

Examples of event log messages of these machines are:

- Paper low
- No paper
- Printer error
- Machine inactive
- Reader intrusion

2.3.2 Fuel Tanks

Fuel tanks are an essential equipment type of a service station as they contain the fuel stock. They are usually installed underground and connected to the fuel distribution system. They resist violent impacts such as extreme weather and have a leakage detector.

Examples of event logs related to fuel tanks are:

- Tank empty
- Low level
- High level

Figure 1: MADIC group payment machine of type APL 3.5

2.3.3 Fuel Dispensers

Fuel dispensers interface fuel tanks and the client who fuels their car. They come in different sizes and can dispense only one or several fuel types. On self-service stations, the fuel dispenser also contains the payment machines.

Examples of event logs related to dispensers:

- Low flow rate
- Alert in pump
- Pumps disconnected

2.4 MagView

As a big part of the service stations in France are self-service stations, the tracking of disorders is more difficult as there is no one to notice and to report possible malfunctions on site. If issues are detected too late, it is a problem for people possibly left without the ability to fuel their car, and providers face economic loss.

To overcome this difficulty, MADIC has developed a surveillance system, "MagView", which gives a real-time overview of all client service stations. It comes as a web portal and allows the selection of each service station to have a more detailed view in which it shows all elements on site and their status.

The points on the map (as shown in Figure 2) mark different service stations in the area, and their color indicates the overall status of the station:

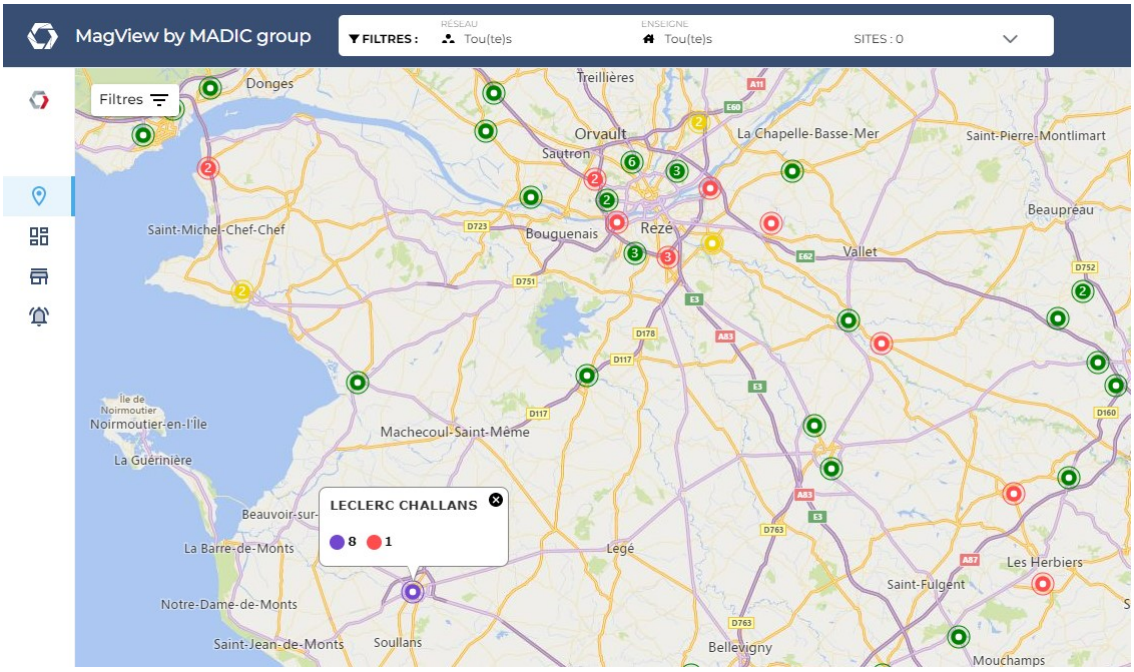

Figure 2: A typical map shown in MagView

- Green: no disorder
- Yellow: information
- Orange: minor problem
- Red: major problem
- Purple: critical problem

By selecting a service station, an event log of the on-site equipment becomes visible, and a statistic of each event during a period defined by the user as shown in Figure 3.

2.5 Available Data

The available data for this thesis project comes from alarm logs created by the connected on-site equipment. The on-site point of sales system (POS) called *ELYS* tracks each component of the service station and receives statistics and data from each of them. ELYS monitors, for example, sensor data and, according to previously defined rules, triggers the different alarm sent via the cloud service *ELYSONLINE* to the systems of MADIC.

The log lines contain, among others, the following information:

Réseau	Enseigne	Site	Identifiant	Gravité	Dysfonctionnement	Description	Date ouverture	Date cloture	Equipement	Détail Equipement
LECLERC	PETRO-OUEST	Leclerc Saumur	2375328	MAJEUR	31	Faible débit	21/10/2022 15:25:00		POMPE03	PISTOLET01
LECLERC	PETRO-OUEST	LECLERC ST HERBLAIN C2	2375212	MAJEUR	31	Faible débit	21/10/2022 15:19:48		POMPE14	PISTOLET02
LECLERC	PETRO-OUEST	NANTES LE PERRAY	2375086	MAJEUR	31	Faible débit	21/10/2022 15:14:21		POMPE03	PISTOLET04
LECLERC	PETRO-OUEST	LECLERC ANCENIS	2375097	MAJEUR	31	Faible débit	21/10/2022 15:11:14		POMPE07	PISTOLET02
LECLERC	PETRO-OUEST	THOUARS	2374995	MAJEUR	31	Faible débit	21/10/2022 15:10:09		POMPE04	PISTOLET01
CARREFOUR	CarrefourMarket	CARREFOUR CANNES	2374672	MAJEUR	29	Anomalie pistolet	21/10/2022 15:08:22		POMPE07	PISTOLET01
CARREFOUR	CarrefourMarket	CARREFOUR CANNES	2374671	MINEUR	38	Imprimante DAC erreur	21/10/2022 15:07:39		AUTOMATE08	

Figure 3: Extract of the MagView event log

- The kind of malfunction/problem represented by an integer ID
- Severity, e.g., "Major" or "Minor"
- Date and time of the event
- Concerned equipment
- Concerned service station and brand

The data is finally stored in an SQL server database, by default accessible from the IT perimeter of MADIC.

2.6 Project Goals

Due to surveillance systems like MagView and the high connectivity of service stations, maintenance providers such as MADIC already have a detailed overview of each station. However, there is room for improvement as not all components and equipment are connected. This thesis project is part of a more significant ongoing project in the company, called *The service station of the future*, whose intention is to increase the connectivity of each service station to have more information and possibilities of action from a distance.

MagView currently collects much data about connected devices and already facilitates the maintenance organization. However, MADIC uses this data only to detect problems when they already exist, which is always an organizational challenge, especially when many service stations need assistance simultaneously or have specific contracts that limit the maximum processing time. To shorten the downtime and the maintenance process, we will look closer at the possibilities of creating a failure forecasting mechanism that recognizes typical patterns leading to critical errors to avoid critical dysfunctions before they occur. Different machine learning and deep learning algorithms will be applied to achieve reliable failure forecasting.

The goals of the project are the following:

- Exploration of different machine learning-based classification methods and their evaluation for the specific use case
- Forecasting critical alarm types based on less serious alarm types that typically occur in advance

2.7 Project Schedule

The project takes place in different phases:

- Data Preparation
- Development of a predictive model for each selected classification algorithm
- Evaluation

Data Preparation An essential prerequisite of a reliable predictive model is data preparation. The given data comes in the form of event logs of different components, so it tends to be noisy. Therefore, logs need to be sorted and separated to account for only those events that concern the equipment in question.

Development of Predictive Models Different classification methods will be applied to the same data set to find a method that suits the specific use case of the thesis. As the main goal of the project is to forecast critical error types for a specific component or machine, the task is interpreted as a classification problem, which suggests the use of the following classification methods:

- K-nearest neighbors (KNN)
- Random Forest
- Fully Convolutional Neural Network (FCN)
- Residual Neural Network (ResNet)

Evaluation Evaluating the different models allows us to see which achieves the best results regarding reliability and accuracy.

The evaluation process is composed of two parts:

1. **Train-Test-Split**: Not all available data will be part of the training. 30 percent will serve as test data. This allows a first and immediate evaluation of the model. The company collected the initial data set for over two years.
2. **Test on new data**: We further evaluate the different models with entirely new data. The company collected this data for around nine months after the initial data collection. This data set is not part of the train-test-split, and the model performance on this test data will show if they can achieve good results on data from another period than the training data.

The evaluation takes place using classic metrics for classification models (Accuracy, Precision, Recall, F1-Score) as explained by Kelleher [JDK20].

3 Theoretical Background

The following section starts with an overview of standard machine learning and data mining methods on discrete data used in previous works. Furthermore, it explains the techniques chosen for this project.

3.1 Methods used in Related Work

Pattern recognition and creating predictive machine learning models based on event logs is a less familiar task. Nonetheless, several methods were already part of previous scientific work. Standard techniques are pattern mining, time series analysis [DWP+22], and natural language processing [DLZS17]. The three exemplary methods *Sequential Pattern Mining* [AS], *Frequent Chronicle Mining* [SST18], and *Change Point Detection* in time series [AC16] are explained in the following.

3.1.1 Sequential Pattern Mining

An essential approach in mining discrete data is *Sequential Pattern Mining*. Initially introduced by Agrawal et al.[AS] for analyzing customers' purchase histories, an adapted version could be suitable for analyzing event logs of industrial machines. Sequential pattern mining aims to find statistically relevant and recurrent patterns in data. It allows us to find relationships between different data items that would not be visible when manually analyzed as the amount of data needed to give reliable results is too large.

Another related mining method introduced by Agrawal et al. [AIS93] detects intra-transaction patterns, a common e-commerce technique. When selecting articles, online shops display a section of other articles that previous customers have often bought in combination with the currently selected article. Sequential pattern mining does not only consider which combination of events or transactions has occurred but also in which specific order. The order delivers essential additional information on the behavior of users or machines.

The Algorithm in Detail The first algorithm for sequential pattern mining was *AprioriAll*, introduced by Agrawal et al. [AS] to find relevant database patterns. The following paragraph explains the algorithm in detail.

To understand the procedure of the algorithm, *Item* and *Sequence* are two essential terms. An item represents a discrete piece of data. In the work of Agrawal et al., [AS], one item corresponds to one purchased article by a customer and is part of an item set corresponding to the list of articles a customer has purchased. A sequence corresponds to a list that includes one or more item sets. The given example represents an ordered list containing all the lists of articles by purchase. For example, if a customer buys first article a and article f first, in a second order article g, and in a third order articles c, e, and m, the sequence looks like the following: *{{a,f}, {g}, {c,e,m}}*. The user defines whether a found pattern is relevant, as they need to input a minimal support threshold in percent. For example, if the threshold is 30%, the candidate sequence must appear in the transactions of at least 30% of the customers to be considered a relevant pattern. Table 1 shows an example database.

The algorithm follows five different work steps. The shown examples are based on the examples of Agrawal et al. [AS].

Customer ID	Date	Items
2345	March 2 2022	A
2348	February 1 2022	B
2348	February 23 2022	A
2350	February 27 2022	B
2350	February 28 2022	D, F
2350	March 1 2022	A
2355	March 5 2022	G, C
2355	March 7 2022	B
2355	March 9 2022	D, E, F
2378	February 28 2022	B, H, F

Table 1: Table with sample data

1. **Sort Phase**: A sequence database as shown in table 2 is created. The database entries are sorted by client ID for a better overview of the transactions and their items.

2. **Large Item Set Phase**: In this phase, the algorithm searches item sets that fulfill the minimal support. These item sets are called large item sets. As each item in a large item set needs minimal support, this algorithm phase also identifies single items with minimal support.

3. **Transformation Phase**: To make It easier to compare item sets and sequences, each large item set is mapped. Agrawal et al. have chosen to map them to simple integers, but other values, such as the article number, are also possible. Afterward, the algorithm transforms the sorted database in the following way: All transactions that correspond to a large item set get replaced by their matching mapped value. If an item or an item set does not fit to any large item set, it gets removed from the database. Customers whose transactions do not contain large item sets are also removed but still counted regarding the support calculation. Table 3 shows the transformed database after this phase.

4. **Maximal Sequence Phase**: This phase applies the actual sequential pattern mining algorithm, which initially was AprioriAll. However, other algorithms with the same input type are possible as well. The goal of this phase is to identify maximal sequences. These sequences fulfill the minimal support and are not part of any other longer sequence with minimal support. AprioriAll is a level-wise algorithm that starts with the sequences of length 1, continues with sequences of length 2, and so forth until no other sequences with minimal support are left. All maximal sequences found in this phase form the output of the algorithm. Table 4 shows the output of the algorithm on the data set of table 1

ID	Sequences
2345	{A}
2348	{B},{A}
2350	{B}, {D,F}, {A}
2355	{G,C}, {B}, {D,E,F}
2378	{B,H,F}

Table 2: A sequence database after phase 1.

Customer ID	Transformed Sequence
2345	{(A)}
2348	{(B)} {(A)}
2350	{(B)} {D,F} {A}
2355	{B} {(D), (F), (D,F)} {(B)}}
2378	{B,F}

Table 3: The transformed database after phase 3.

Found Sequential Patterns
{(B) (A)}
{(B) (D,F)}

Table 4: The result of the algorithm on the data set shown in 1 and with a minimal support of 25%.

Evolution and Variants of the Algorithm Since the first proposition of the AprioriAll algorithm, many other algorithms for sequential pattern mining such as *GSP* (introduced by Srikant et al. [RS96] as an improvement of AprioriAll), Spade [Zak01], Spam [AFGY02] and CM-Spam [PFV14] where proposed. Fournier et al. [PFV17] gives an informative overview of the most important algorithms of the domain. When running an algorithm for sequential pattern mining, it finds all sequences corresponding to the minimum support the user gives. That means that only one single result is possible when the algorithm runs on the same data set with the same minimal support given. Therefore, all algorithms will find the same sequence patterns and only differ regarding computational resources, runtime, and how they identify relevant patterns. The four significant points of distinction are:

- Search algorithm - whether Breadth-first search or Depth-first search
- Database representation
- Selection of the next pattern to explore
- Count method of the pattern support

Shortcomings One major shortcoming of sequential pattern mining, in general, is that it only considers the specific order in which the events occur. The approach does not consider the time that has passed between the occurrence of the events in a frequent sequence.

Application on Service Station Error Logs Sequential Pattern Mining has numerous fields of application. Besides the analysis of the shopping behavior of customers, it is, among others, also a successful method for predictive maintenance use cases, as shown by Kahraman et al. [KKKK21] who used it to predict failures of trucks, and in the field of health care like in the work of Garg et al. [GMMM08], who applied the method on patient data. As sequential pattern mining operates on categorical data, the above-described method can easily be applied to our use case of alarm logs of service stations as well because each log contains a categorical alarm ID that represents an alarm type. An item would correspond to an alarm type, and a sequence would represent a sequence of occurring alarms. Similar to the analysis of customer transactions, sequential pattern mining can give an insight into which disorder types occur together regularly in the same service station and find patterns among them.

3.1.2 Frequent Chronicle Mining

Another data-driven mining approach is *Frequent Chronicle Mining*. It is similar to sequential pattern mining but overcomes one of its critical problems: it considers a temporal aspect between the events. Dousson et al. [CD99] first established the term chronicle and described it as a pair of events in the order in which they have occurred and a set of temporal constraints displayable as a graph.

SID	Sequence
1	(A,1), (B,7), (C,9)
2	(A,2),(B,6),(C,7)
3	(A,1),(D,5)
4	(A,2),(D,3)

Table 5: Sample set of four sequences with time constraints.

Table 5 shows an example data set, including four sample sequences. Each sequence consists of the event (e.g., A) and the number of time units (e.g., hours, days) after which the event has occurred in this specific sequence. When looking at sequence 1, the notation means the following:

- Event A occurs one time unit after the beginning of the sequence.
- Event B occurs seven time units after event A.

- Event C occurs nine time units after event B.

The temporal constraints must be defined to transform the given sequences into a chronicle with time constraints, as shown in Figure 4.

Temporal Constraint Sellami et al. [SST18] described a temporal constraint between two events as a quadruple (e_1, e_2, t^-, t^+), where e_1 represents the first event, e_2 the second event, t^- the minimal number of time units and t^+ the maximal number of time units which pass between these two events.

Given the sample chronicle in 4, the following temporal constraints are included:

- $(e_1, e_2, t^-, t^+) = (A, B, 4, 6)$ - Event B occurs four to six time units after A.
- $(e_1, e_2, t^-, t^+) = (B, C, 1, 2)$ - Event C occurs one to two time units after B.
- $(e_1, e_2, t^-, t^+) = (A, D, 1, 4)$ - Event D occurs one to four time units after A.

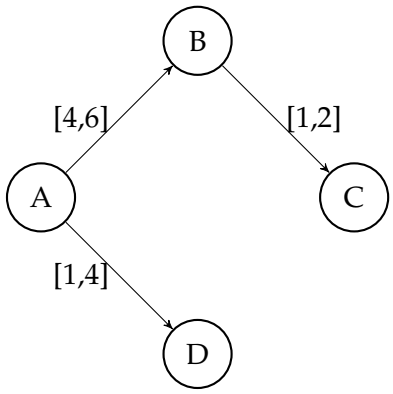

Figure 4: A chronicle based on table 5

Sellami et al. [SST18] have introduced the approach used in predictive maintenance to predict the time of a failure of industrial machines, given a set of event-time-pairs. Their approach consists of five steps:

1. **Preprocessing of the data set**: This includes transforming the database entries into sequences of events, feature selection, and discretization.
2. **Sequential failure pattern mining**: Finding relevant and frequent patterns in the given data set as described in section 3.1.1. This phase only considers sequence patterns leading to a failure or breakdown of the machines. The algorithm *CloSpan* introduced by Yan et al. [YHA03] was applied.

3. **Time constraints extraction**: Identification of temporal constraints for the occurrence of each event in the failure sequence patterns.

4. **Failure Chronicle Generation**: Generation of chronicles from the found sequential failure patterns and their belonging temporal constraints. A failure chronicle is a chronicle whose last event is an event that indicates a failure or breakdown.

5. **Failure detection**: Detection of failure patterns in new, incoming data using previously found failure chronicles. A sequence is recognized as a failure-leading sequence if it covers a chronicle containing the same events and time constraints as a given and known failure chronicle.

Overall, the introduced algorithm shows results between 83% and 90% for the actual positive rate, the precision, and the f-measure. These results make the proposed approach a potential candidate for further application in the predictive maintenance domain on the basis of discrete event logs.

Sellami et al. [SMS+19] enhanced this approach and expanded the failure prediction by how much time is left until a machine breaks down given the incoming events. Cao et al. [CSZM+20] proposed another approach for predictive maintenance using frequent chronicle mining, where it was combined with semantics to make the generated knowledge more understandable for human stakeholders.

3.1.3 Change Point Detection

Sequential pattern mining and chronicle pattern mining are two approaches to analyzing discrete event data. *Change point detection* is an entirely different approach that works on data sets containing time series. According to Aminikhanghahi et al. [AC16], change point detection identifies points of a sudden change in a machine's or a process' behavior. This can be, for example, a physical phenomenon like the abrupt modification of the temperature or an increasing vibration of a machine.

Until today, researchers have introduced various change point detection methods. Van den Burg et al. [BW20] gives an interesting overview of existing algorithms and their performance. Accordingly, the existing methods can be distinguished based on the following criteria:

- **Online/Offline**: Online methods identify change points in real time. These methods include a tolerance to avoid false alarms. Offline models identify change points after having received the entire amount of data.

- **Univariate/Multivariate**: Different methods can deal with univariate time series, others with multivariate time series.

According to Van den Burg, the change point detection algorithms with the best performance are the following:

- Binary Segmentation (BINSEG), introduced by Scott and Knott [SK74]
- At Most One Change (AMOC), introduced by Hinkley et al. [Hin70]
- Pruned Exact Linear Time (PELT), introduced by Killick et al. [KFE12]
- Bayesian Online Change Point Detection (BOCPD), introduced by Adams and MacKay [AM07]
- BOCPD with Model Selection (BOCPDMS), introduced by Knoblauch and Damoulas [KD18]

The probably most established change point detection algorithm is the binary segmentation introduced by Scott and Knott [SK74]. The basic idea of this algorithm is to calculate a cost function of a sequence of events or measures. The cost of a sequence is lower if it contains a change point. At the beginning of the algorithm, it looks for a change point in the entire time series. The algorithm stops here if no change point exists and no further analysis is necessary. If a change point exists, the time series gets separated into two sequences: before and after the change point. When looking for a change point, the algorithm tries to find a position at which the current sequence gets separated and where the cost of both sequences in a sum is smaller than the cost of the original sequence. When a sequence gets divided, the algorithm applies the same procedure to the subsequences and separates them again if it finds another change point. This procedure continues until no other sequence contains change points.

When change points represent a modification in the behavior of a machine, they can also mark the change from a normal working behavior to an abnormal behavior. Therefore, this approach can be of interest when predicting malfunctions and disorders. When recording sensor data of a machine, such as temperature, vibration, and volume, upcoming breakdowns might be predictable based on abnormal changes in this sensor data. Early identification of such behavioral patterns could help optimize the schedule of maintenance activities and reduce the downtime of the machine in question.

Research by Guillaume et al. [GVW20] used change point detection to forecast automated teller machines (ATM) malfunctions. The available data set of this research are discrete event logs that contain transaction events such as withdrawals, but also warnings and errors of the machine. Each record comes with a time stamp as well as an event code. In total, 284 unique event codes exist, and the data was collected over more than two years from around 150 different ATMs in France. The research aimed to develop a method that predicts the probability of upcoming disorders of the machine while analyzing sequences of events. A predictive padding was defined to avoid false alerts. A predictive padding is the time that indicates how long beforehand the future error should be signaled to prevent the schedule of possibly unnecessary maintenance actions due to a false alarm. The time series

were separated into life cycles to process the data, representing the period from the last maintenance until the following raised error from which the machine cannot recover. As the data set consists of an event log and change point detection usually takes a time series as input, a transformation of the event log into a time series is necessary.

To find a suitable change point detection method for this individual use case, five different methods have been tested and compared on the same data set:

- Low-cost Unipotent Semantic Segmentation (FLUSS)
- Kernel Change Point Detection (KCPD)
- Linearly Penalized Segmentation (Pelt)
- Binary Segmentation (Binseg)
- Bottom-Up Segmentation (BottomUp)

At the end of the research, none of these methods achieved satisfying results on the data set. Guillaume et al. give several explanations for this outcome. The condition that a prediction is considered a true positive is rather strict, as the error must be predicted before it appears but necessarily during the padding period. Furthermore, the collected data has shown that changes in the machine behavior often only happen right before it raises the error event, leaving no time for actions to avoid the error itself.

3.2 Supervised Learning and Classification

The project aims to predict critical alarm types based on a sequence of occurred alarms with lower severity. As the number of alarm types is limited, we interpret the project task as a classification task, where critical alarm types represent possible classes. Furthermore, each alarm type has a unique ID, which is why the input data is categorical. Therefore, it is unnecessary to interpret time series (which is especially suitable for continuous data such as measurements or sensor data) nor to analyze the text of the different log lines. This opens the possibility to investigate the efficiency of classification methods that deal with categorical data.

Classification Classification is a learning task that takes labeled data samples as input and tries to find patterns that allow one to make predictions on future data. The goal is to map the input samples to different categories (classes) based on their features. Based on the number of different available classes, a classification can be:

- **Binary**: Only two distinct classes are possible. Two use case examples of binary classification models are fraud detection (fraud/not fraud) and spam detection of e-mails (spam/not spam).

- **Multiclass**: More than two classes are possible. The possible use cases of a multiclass classification are very diverse. Models are used, for example, to categorize animals into different species (e.g., bird, cat, dog), for the recognition of handwritten text (detection of different letters, numbers, and symbols), or to recognize different objects in images (e.g., house, car, flower).

Supervised Learning Classification belongs to the supervised learning methods. According to Liu et al. [Liu11], supervised learning is a learning paradigm that maps input labels to an output label based on the features of the input data. Its main task is to find a rule that correctly maps all input labels to their corresponding output labels. The opposite approach is the unsupervised learning paradigm, which handles non-labeled input data.

Figure 5 shows the typical steps when developing a supervised learning model as described by Ouadah et al. [OZGS22]:

The project follows similar steps, which we will apply as follows:

1. **Data collection and exploration**: The available data is collected, and its characteristics are explored.

2. **Data Preprocessing**: In this step, we clean the data and reduce noise. Afterward, we separate the data into samples with their corresponding label.

3. **Algorithm Selection**: The learning algorithm for the individual learning task is selected. For this project, we have chosen *K-nearest neighbors*, *Random Forest*, *Fully Convolutional Neural Network*, and *ResNet*.

4. **Model Training**: We train the model with the training data. Training data is around 70% of the entire data set.

5. **Evaluation**: We evaluate the model using the remaining 30% of the data set, which serves as test data. In addition to the train-test split, we evaluate the models with entirely new and unseen data collected after creating the initial training data set.

6. **Parameter Tuning**: Each algorithm or learning method has parameters to define before training. Every parameter can influence the model performance significantly and, therefore, needs to be defined carefully. There is no general recommendation for setting each parameter as it depends on the individual learning task and the available data.

The development process is not linear but contains iterations. Each of the described steps can be repeated if the model performs poorly after training. Sometimes, it needs more data, parameter tuning, or a completely different algorithm to achieve better results on the learning task.

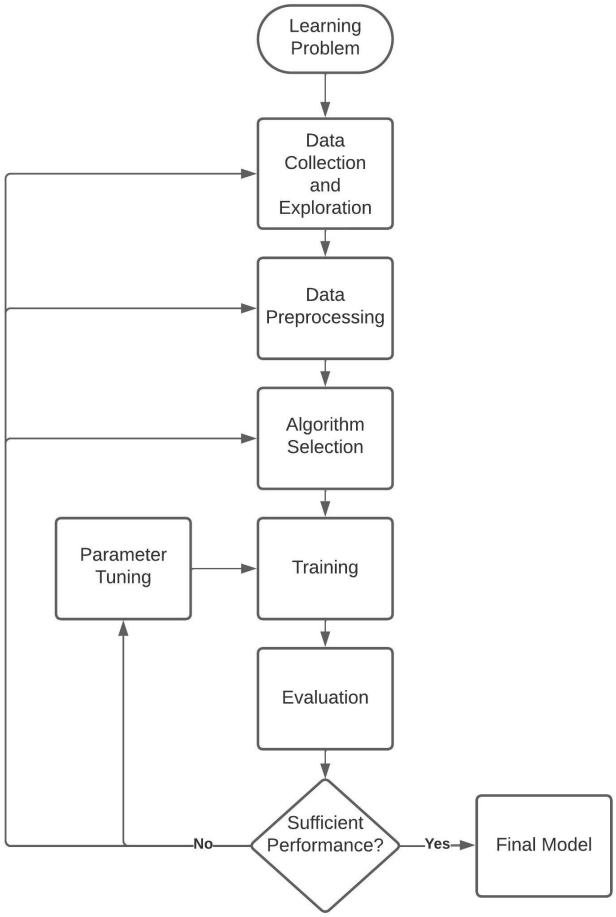

Figure 5: Development Process of Supervised Learning Models

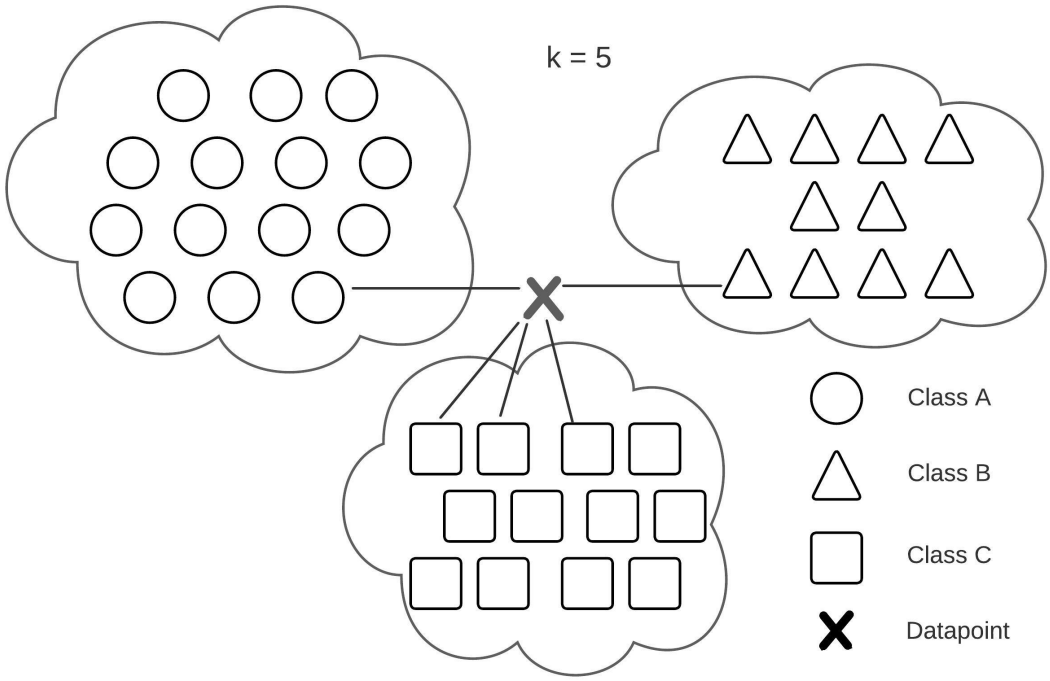

Figure 6: KNN working principle

3.3 Selected Algorithms

The following section describes the selected classification algorithms in detail.

3.3.1 K-nearest Neighbors

K-nearest neighbors (KNN) is an algorithm that makes predictions of a data sample based on the distance to other similar data points known to the model. As stated by Kelleher et al. [JDK20], the algorithm is based on the assumption that similar data points are close. The algorithm evolved from different pattern recognition methods and is suitable for classification and regression tasks and is considered a lazy learning technique as the training samples are only registered during the training process and need to be loaded into the memory when the model is used to make predictions. Figure 6 shows a simplified visualization of the working principle.

The prediction process of KNN consists of three steps:

1. Identification of the k nearest neighbors based on a distance metric
2. Determination to which classes the k nearest neighbors belong to

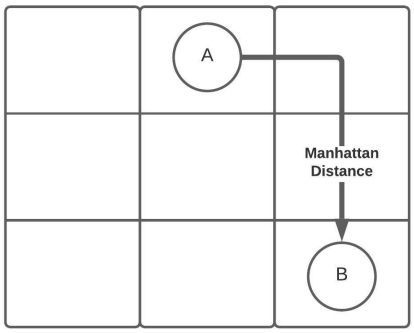

Figure 7: Manhattan Distance

3. Majority voting between these classes to define the predicted class of the unknown data point

Parameters The algorithm has a few parameters to define before the training process. The parameter k is the number of closest neighbors taken into account for a prediction and is a hyperparameter as it can significantly affect the performance of the classification model. The selection of k is very complex. A too-small k can create noise as it only considers a few data points. A too-big k, on the other hand, can decrease the prediction quality, as it also considers data points with longer distances from the new data point. The definition of k always depends on the number of training samples and the individual use case and needs experimentation with different values. Furthermore, it is possible to define whether the calculated distances to the k neighbors should be weighted. Weighted distances give greater importance to those nearest neighbors closer to the data point than others. The last parameter is the distance metric that defines how to calculate the distance between the new and the other data points to select the nearest neighbors. The following distance metrics are the most common:

- **Manhattan Distance**: The Manhattan distance is also known as city block distance and calculates the shortest distance between two data points as shown in Figure 7.

$$\text{Manhattan} = \sum_{i=1}^{n} |x_i - y_i|$$

- **Euclidean Distance**: The Euclidean distance is the straight-line distance between two data points as shown in Figure 8.

$$\text{Euclidean} = \sqrt{\sum_{i=1}^{2} (x_i - y_i)^2}$$

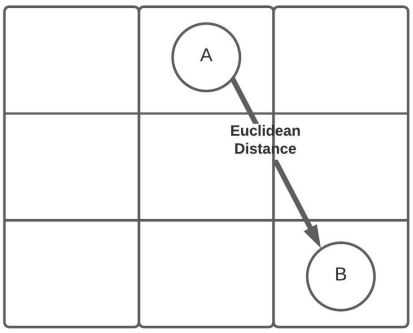

Figure 8: Euclidean Distance

3.3.2 Random Forest

Random Forest is an ensemble learning technique and an extension of the classic decision tree algorithm. A decision tree model creates a tree-like structure that splits data into data subsets step by step. Each decision tree has one root node and several internal and leaf nodes. The decision of how to divide the data is made based on the most significant feature of the current step and is represented by an internal (or decision) node. Leaf nodes represent the possible classes. Figure 9 shows a sample decision tree. Bramer [Bra] explained that decision trees tend to overfit, especially when they become deeper. Random Forest can overcome this problem because the prediction is not based on only one single decision tree but on several. Each tree is constructed based on a different selection of features and gives its prediction, which prevents single features from becoming too dominant in the prediction process. Figure 10 shows a visualization of the working principle of a random forest. The prediction process contains the following steps:

1. The data sample is passed individually through all available decision trees.
2. A majority voting between the predictions of each decision tree is done.

Parameters The number of decision trees a model will create is a parameter to choose carefully. The more estimators the algorithm creates, the better the performance, but an increasing number of estimators also increases the need for computational resources. Another important parameter is the criterion, which indicates how to select the next feature, on which the current data subset gets split into further subsets. The most common criteria are:

- **Entropy**: The feature with the highest information gain is selected

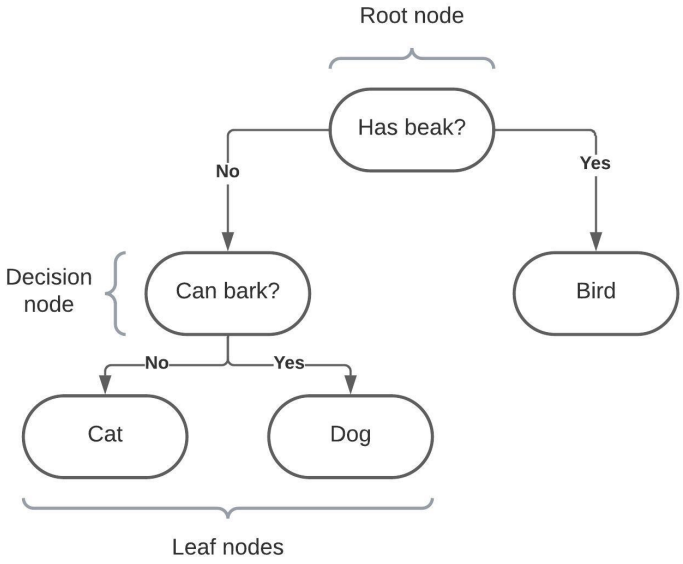

Figure 9: A simple decision tree

- **Gini Impurity**: The feature that creates the purest subset of data labels is selected. This criterion is common for binary classification rather than multiclass classification.

Other parameters of random forest are the maximum depth, which limits the depth of each decision tree to a maximum value, how many samples are at least necessary to split an internal node, how many samples are at least needed to be in a leaf node, and how many features need to be taken into consideration when making a split.

3.3.3 Fully Convolutional Network

A *Fully Convolutional Network (FCN)* is a deep learning approach and a variant of convolutional neural networks. Long et al. [LSD15] initially proposed this powerful architecture, especially for semantic segmentation for image classification. However, as there are convolutional layers for 1-dimensional input existing, it can also be used in a modified version for non-image input. Convolutional neural networks differ from other neural network architectures in having particular convolutional layers that perform, as the name suggests, an operation called *Convolution*. Convolutional layers contain kernels that serve as a kind of filter and which can recognize patterns. A filter is shaped like a grid or a matrix and has a smaller dimension than the input data. Each pattern has its filter. Each kernel loops through the image data block by block, while each block has the exact dimensions as the kernel itself. For

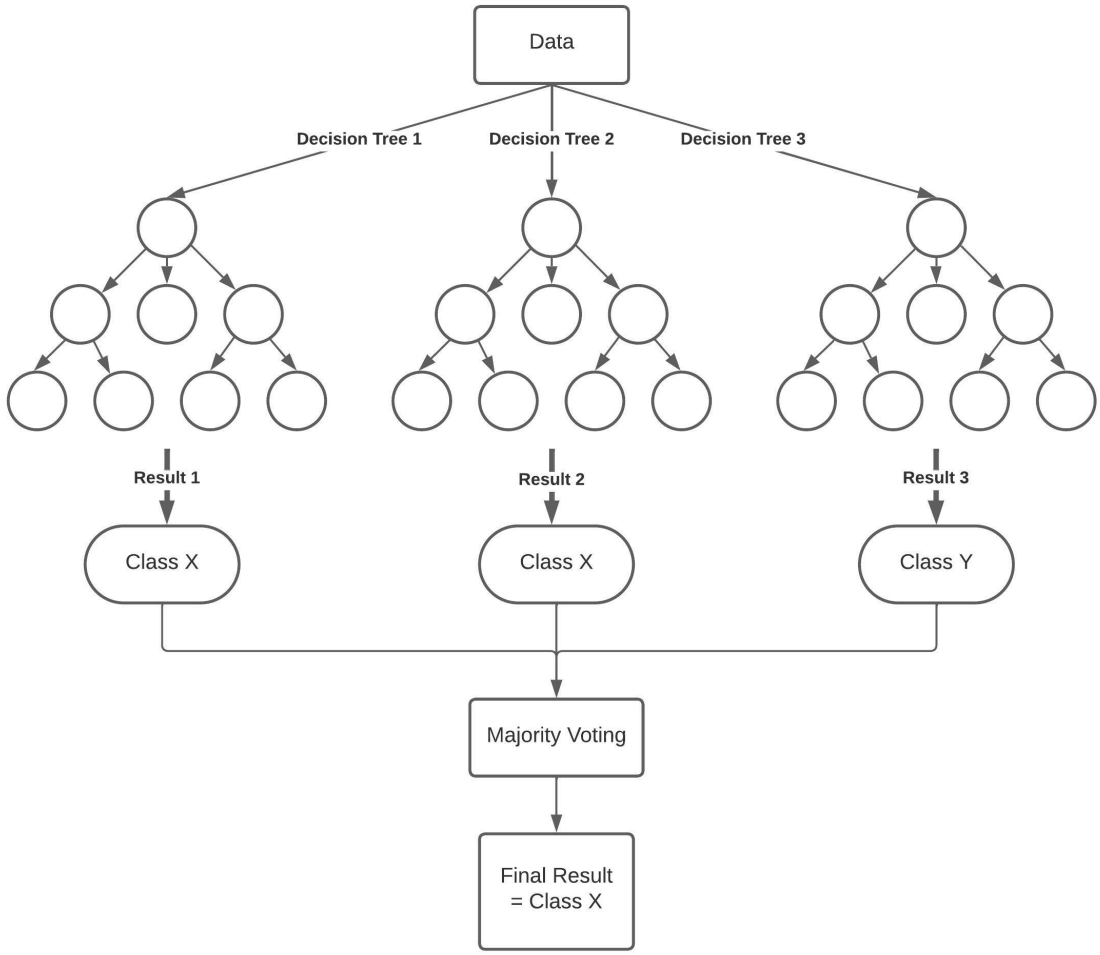

Figure 10: A simple random forest model consisting of three estimators, each giving its prediction, followed by a majority voting to make the final prediction.

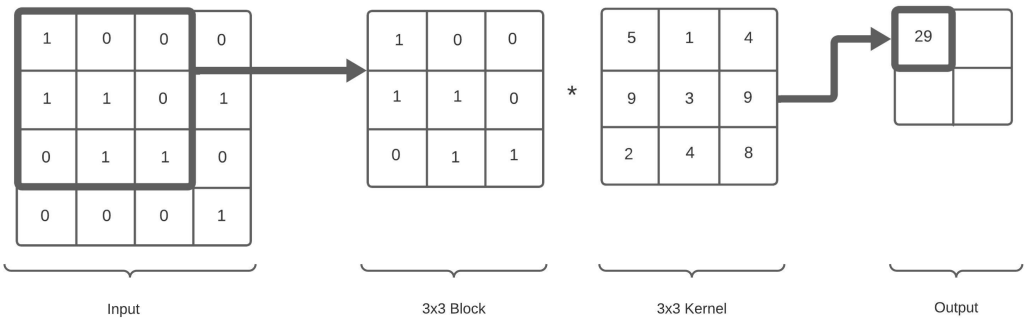

Figure 11: Visualization of a convolution example. The algorithm loops through the data in blocks of 3x3, and the output dimensions represent the number of possible 3x3 blocks in the input data's horizontal and vertical direction.

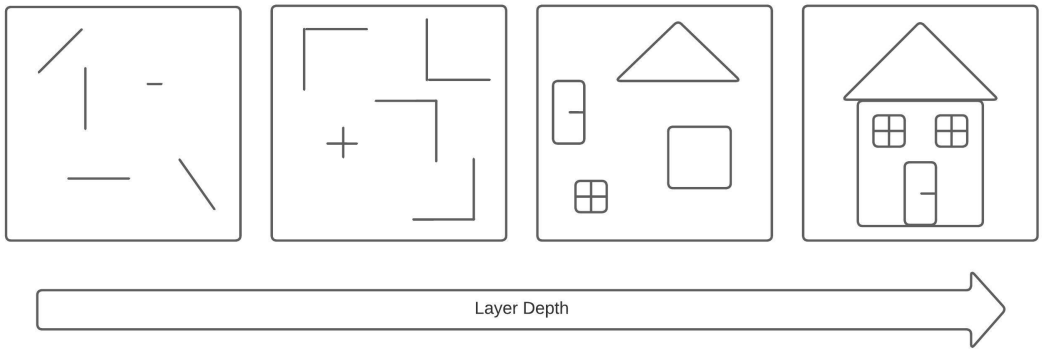

Figure 12: The deeper the convolutional layer, the more sophisticated the patterns recognized by the kernels.

example, if the filter is of dimension 3x3, the convolution will pass through the input data block by block of size 3x3. Figure 11 shows a visualization of a sample convolution.

In the example of an image classification, the first convolutional layers recognize more basic shapes like vertical or horizontal lines. But the deeper the layers, the more sophisticated the patterns the kernels can detect. When analyzing an image of a house, the first layers will likely recognize different lines and angles. Later, they can recognize smaller objects like windows, doors, and roofs until they can recognize the entire house as a complex object like in Figure 12.

Contrary to classic convolutional neural network architectures, fully convolutional networks contain especially convolutional layers with a different number of filters on each layer. An important characteristic is that they do not have fully con-

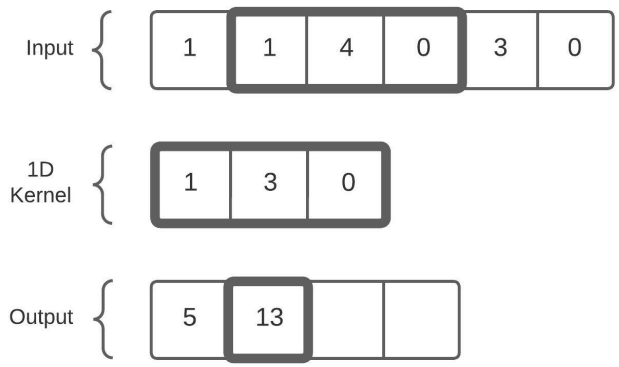

Figure 13: Similar to 2-dimensional data, the convolution can also be performed on 1-dimensional data. A 1D kernel with a length of 3 steps through the input data block by block.

nected layers, except in some architectures designed for classification tasks where dense layers follow the convolutional part. Many FCNs have an Encoder-Decoder-Architecture:

- **Encoder**: The encoder downsamples the input to recognize features and spatial relations between the objects in the image.
- **Decoder**: The decoder upsamples the data to make predictions with the same dimensions as the original input data.

Even if the FCN architecture was designed for image recognition, the architecture is also suitable for 1-dimensional data. Specific 1D kernels perform the convolution, as shown in Figure 13.

3.3.4 ResNet

A *Residual Neural Network (ResNet)* is a specific architecture for deep neural networks and was initially proposed by He et al. [HZRS16]. The architecture was originally invented for image classification and tried to overcome the vanishing gradient problem, a recurrent problem of deep neural networks.

Aggarwal et al. [Agg18] described the vanishing gradient problem as a problem that affects the stability of weight updates during the training process. In classic neural network architectures, the output of a layer is also the input of its underlying layer. Therefore, the data passes through the network layer by layer, and the weights of the layer connections are updated using backpropagation. A vanishing gradient manifests itself when the gradient of the loss function during the training process becomes extremely small. Therefore, the training process is very slow, which can, among others, slow down the update of layer weights, extend the training process

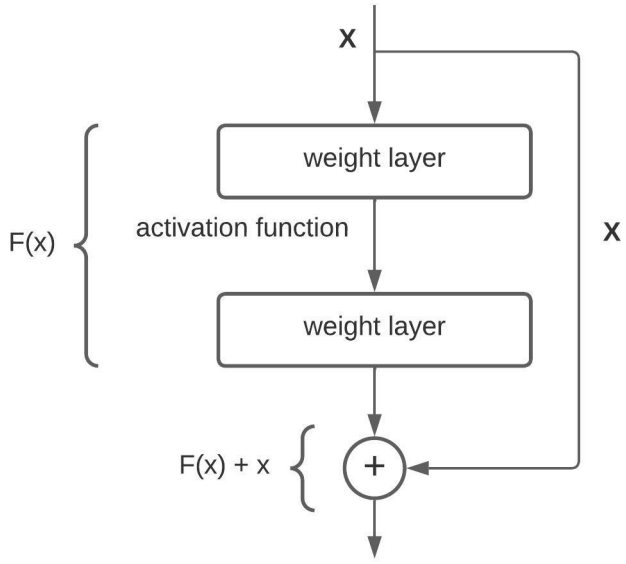

Figure 14: Sample of a residual block with skip connections as shown by He et al. [HZRS16].

as many epochs are needed, and prevent the neural network from learning essential patterns in the data.

He et al. proposed the ResNet architecture as a possible solution to reduce the vanishing gradient problem. It has a crucial difference to other neural networks as it allows *Skip connections*. Contrary to classic neural network architectures in which each layer is connected only to its next layer, ResNets use specific connections to bypass the following layers in the network. Residual blocks common to the architecture make this possible.

A residual block in a ResNet contains two or more convolutional layers, each with an activation function. A skip connection transfers the input data directly to the output of the block without performing any convolutions on the data. Figure 14 illustrates the structure of a sample residual block as shown by He et al. [HZRS16].

The network calculates the output of a residual block as follows:

$$\mathrm{y} = F(x) + x$$

F(x) represents the output of the convolutional layers of the current residual block, and x is the input of the block. The network architecture is created by stacking residual blocks onto each other, as visible in Figure 15. The significant advantage is that adding residual blocks to a neural network architecture does not harm its performance. A block can be skipped if it is not helpful or would negatively affect the model performance. On the contrary, if the residual block turns out to be beneficial,

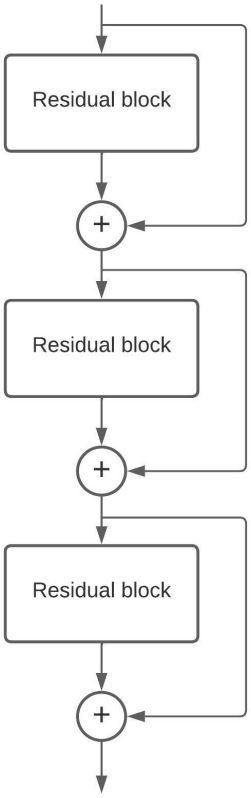

Figure 15: Extract of a ResNet architecture created by stacking several residual blocks.

the model performance will increase. With that, the learning process focuses on crucial features and reduces the flow of less critical layer connections. Furthermore, the number of layers does not need to be tuned like a hyperparameter, as more blocks can only increase the model performance or leave it unaffected. Skip connections can also lead to better generalization and faster convergence.

Similarly to FCNs, ResNets are initially designed for 2-dimensional input, but the architecture can also be adapted to process 1-dimensional input.

4 Methodology

The following section describes the methodology of the investigation, including the technical environment of the model development, data preprocessing, data encoding methods, and hyperparameter tuning.

4.1 Technical Environment of the Development

The data preprocessing and the different machine learning models have been developed under *Python* version 3.9.2 in the IDE *DataSpell* version 2023.1 using *Jupyter Notebook*. We used the following libraries:

- **Data preprocessing, data encoding and sample selection:** *Pandas* version 1.4.2
- **Development of the Random Forest and KNN models:** *Scikit Learn* version 1.0.2
- **Model export:** *Joblib* version 1.1.0
- **Upsampling:** *Imblearn* version 0.11.0
- **Development of FCN and ResNet:** *Keras* version 2.12.0, *Tensorflow* version 2.12.0
- **Data access:** *Microsoft SQL server*

4.2 Experimental Setup

This thesis project compares different classification approaches to determine whether they are suitable for predictions about the given data set. We have tested each algorithm on the same data set for a reliable comparison. The experiment contained the following steps:

1. **Data preparation:** In the first step, we cleaned the data and removed incomplete sequences. The encoding of the data samples brought them into a shape that the classification algorithms can process. We applied two encoding approaches to compare their impact on the model performance.
2. **Sample selection:** Three different sample selection approaches were tested.
3. **Training:** We have trained one model by data encoding approach and classification method.
4. **Evaluation on test data:** The models' evaluation using the train-test split test data gave a first overview of their performance.
5. **Evaluation on new data:** We tested those models with the best performance in the train-test split on new data.
6. **Upsampling:** As the data set is imbalanced, we did an upsampling of the minority classes to enhance the prediction performance of those classes and analyze the impact on the overall model performance.

4.3 Data Preparation

The following section describes the available data set in more detail and closely examines its characteristics and the handling of its imbalanced nature.

4.3.1 Overview and Characteristics of the available Data

The overall data set is an alarm log of over 30 million lines of more than 2700 service stations. MADIC collected this data for around two years. Each log line represents an alarm type raised at a specific time. In total, 109 different alarm types exist, meaning each log line can contain only one out of the 109 given alarm types. Furthermore, each alarm type has a severity level, which can be *Anomaly*, *Minor*, *Major*, or *Critical*. Table 6 shows an extract of the possible types. The alarms are raised thanks to an on-site surveillance system that monitors different indicators, providing information about the current condition of each equipment or machine. Typical indicators are, for example, machine logs, the flow rate of the fuel dispensers, and different sensor data like temperature, volume, and vibration. The error events are raised and logged according to specific rules implemented for each error type within the surveillance system.

Example: If a specific fuel dispenser usually has a flow rate of around 70 liters per minute, but its actual flow rate is 90 liters per minute, the surveillance system raises an alarm that states that the flow rate of this specific fuel dispenser is higher than usual.

As different service stations do not have the same size and facilities, the installed hardware on-site varies enormously from service station to service station. Therefore, each rule is defined individually according to the machine type and adapts to the specific technical characteristics of the equipment. The fact that future machine learning models must operate on error logs is both an advantage and a disadvantage. The advantage is that no sensor data needs to be analyzed. Therefore, the models do not need to differ between machine types and their various characteristics because the on-site surveillance system translates the result of its analysis into discrete events that apply to all stations, no matter which machines are available there. For instance, different types of fuel dispensers exist. Therefore, their regular average flow rate can differ. Nevertheless, the event in case of a derogation of this average flow rate is the same, which means that the predictive models work on a higher level. A disadvantage is that the log only contains alarms and no other event types, such as information, warnings, or transaction details, which could be precursors for future breakdowns. The available data on what happens in a machine during its life cycle is quite limited.

The log which contains the alarms is transferred from each service station to a cloud service until it reaches the database. Each log line includes, among others,

Alarm ID	Alarm Description	Severity
5	The payment machine does not read payment cards anymore.	Major
37	The printer cannot deliver payment receipts anymore due to an error or a lack of paper.	Minor
40	The fuel stock of a tank is lower than the authorized minimum.	Major
32	A pistol is in use for longer than normal. This can indicate a problem with a contact.	Major
34	The distributed fuel volume of a pump is higher than the maximum authorized volume.	Critical
56	The card reader is no longer accessible.	Critical

Table 6: Extract of possible alarm types.

the alarm ID, the date and time of its occurrence and transmission, the concerned service station, the equipment in question, and its equipment ID. After treating and closing an alarm, its database record moves to a history table. The content of this table serves as the base to create the data set for this thesis project. Table 7 shows a small extract of the alarm history table.

The severity level of each log line is not directly part of the history table. However, it is available due to the mapping with another table that contains the ID, description, and severity level for each disorder type.

4.3.2 Data Preprocessing

Before the data encoding can take place, the data set needs to be cleaned up and prepared. The log will be grouped by service station and ordered by the date and time of the alarm occurrence to achieve a chronological order of the log lines by service station. As the project concentrates on predicting critical alarms using a sequence of occurring alarms with a lower severity (anomaly, minor, or major), the logs are separated into these sequences whenever a critical alarm type occurs. Given the example of Table 8, the input X for this example would correspond to the sequences

- X = [62,2,22,48] with label y = 33
- X = [43,5,37] with label y = 39.

Date and Time	Alarm Type ID	Site
2018-12-20 06:24:39.517	67	106000002078567
2019-01-01 03:03:22.860	48	0106000000001372
2019-01-01 03:06:04.787	48	0106000002058181
2019-01-02 03:03:36.283	48	0106000002042329
2019-01-02 03:06:29.353	37	0106000002042329
2019-01-03 03:03:20.803	48	0106000002042329
2019-01-03 03:05:26.030	56	106000002078567
2019-01-04 03:03:26.377	9	0106000002058181
2019-01-04 03:05:24.087	72	0106000002009726
2019-01-05 00:38:59.323	48	0106000002009726
2019-01-05 03:03:33.340	48	0106000002059720
2019-01-05 03:05:25.240	48	0106000002058092
2019-01-06 03:03:20.690	37	0106000002058042
2019-01-06 03:05:36.940	48	0106000002058042
2019-01-07 03:03:44.583	48	0106000002058181
2019-01-07 03:05:48.473	65	0106000002078052
2019-01-08 03:03:34.477	56	0106000002078052
2019-01-08 03:05:52.987	95	0106000002009726
2019-01-08 15:01:44.050	95	0106000002042329

Table 7: Short and simplified extract of the alarm history.

In total, 109 different alarm types are possible, of which 32 are critical alarm types and 77 alarm types with a lower severity. After investigating the critical alarm types, five do not have any occurrence in the data. We removed 15 more as they correspond to manual actions or events that technically cannot correlate with possible disorders of the on-site equipment. Examples of such sorted-out alarm types are:

- An intrusion of a payment machine has been detected.
- Someone has manually restarted the ELYS application.
- The payment servers are not available.

Events such as intrusions happen suddenly, especially at night, and cannot be predicted based on on-site disorders of other equipment. Manual actions such as an application restart are non-predictable as well. The reasons for a restart are so diverse and only sometimes linked to the equipment that we have excluded these

Date and Time	Disorder ID	Severity
2020-10-20 06:35:39.517	62	Major
2020-10-21 03:03:22.860	2	Anomaly
2020-10-21 03:06:04.787	22	Major
2020-10-21 06:22:36.283	48	Minor
2020-10-21 06:45:29.353	33	Critical
2020-10-25 06:35:39.517	43	Major
2020-10-26 03:03:22.860	5	Major
2020-10-27 06:22:36.283	37	Minor
2020-10-27 06:45:29.353	39	Critical

Table 8: Example of a chronologically ordered alarm history.

alarm types from the project scope. Furthermore, alarms such as the unavailability of the payment servers are external events whose causes do not lie in the material on site and, therefore, are unpredictable with prior on-site alarms. The 77 alarm types with lower severity contain manual actions and events caused by reasons outside the on-site system, but we did not remove them as their impact can correlate with other disorders in the field. 68 out of 77 possible types occurred in the data.

Occasionally, some critical alarm types occurred in the data without any other alarm of lower severity beforehand. These examples show that a possible correlation between several critical alarm types can exist or that some critical alarms do not have any harbingers or correlations with other alarm types. After investigating which critical alarm types occur alone, it became clear that no alert type always occurred alone. Therefore, we have removed these log lines, but it is important to investigate them further in future projects. Table 9 shows the final list of the critical alarm types representing the model classes.

Not all sequences between two critical alarms have the same length. The shortest sequence length is one, while other sequences can have up to 2500 elements. Technically, it is not likely that all the elements of such a long sequence are related to the critical alarm at the end. Therefore, we only consider the last 20 incidents before the critical incident occures.

4.3.3 Data Encoding

Given the result of the data preprocessing, the starting point of the data encoding are sequences of length 20 as the input and the corresponding critical alarm type ID as the label. We investigated two encoding approaches and compared them regarding their impact on the performance of each created machine-learning model:

Alarm ID	Description
7	The limit of payment machines which are out of order has been exceeded.
15	The service station is disconnected.
16	An incident prevents the station system from starting.
21	A measuring unit is out of service.
33	A fuel dispenser dispenses fuel for more money than allowed.
34	The distribution volume of a pump is higher than allowed. All payment transactions should be verified.
39	High water level in fuel tank.
56	The reader of the payment machine does not respond.
68	A transaction is blocked. A customer validated a payment transaction, but the fueling process was never registered as finished.
95	Several payment machines have the same terminal ID.
1000	The console of the station does not react anymore
1001	The payment machine is inactive

Table 9: List of possible classes and their meaning.

- **Sequences of disorder IDs**: This approach extracts the disorder IDs (integer) from the log lines and transforms them into a sequence.
- **Count-based approach**: This approach counts the number of occurrences of each disorder type in the given sequence.

Sequence Encoding When dealing with categorical data, encoding its features can be challenging. However, in the given data set, each alarm type has its unique integer ID, making the treatment easier. This encoding approach treats the samples as sequences of integer IDs.

Example Given the data example of Table 8, this approach would transform the log into the two following sequences:

- X = [62,2,22,48] with label y = 33
- X = [43,5,37] with label y = 39

Not all sequences have the same length. As we only consider the last 20 items of a sequence, the maximum length is 20. Nevertheless, some sequences are shorter than

that, and most classifiers only work on sequences of the same length. Therefore, those with a length of less than 20 get fulfilled with 0 values.

The significant advantage of this encoding approach is its simplicity, as it does not need much transformation work because all alarm types already have their unique integer ID. A disadvantage is the difficulty in treating the sequences with different lengths.

Count-based Approach The count-based approach is the simplest of the three selected encoding approaches. In total, 77 different alarm types of lower severity exist, and the idea is to go through each given sequence and count which alarm types occurred and how often during this sequence. Therefore, the input X has a dimension of 77, one for each possible alarm type, and each position contains an integer value representing the number of occurrences of the corresponding alarm type in the given sequence.

Example: To demonstrate the encoding approach, we assume a simpler data set with ten different alarm types with the possible IDs *1,2,3,4,5,6,7,8,9,10*. The following three examples show each the base sequence *s* and the count-based encoded result.

- s = [2,2,3,1,5,1,2] becomes s' = [2,3,1,0,1,0,0,0,0,0], indicating that the alarm type 1 has occurred once, the alarm type 2 three times, and the alarm types 3 and 5 both once.
- s = [5,10,10] becomes s' = [0,0,0,0,1,0,0,0,0,2], indicating that alarm type 5 has occurred once and the alarm type 10 has occurred twice.
- s = [3,3,3,8,6,6,9,8,7,5] becomes s' = [0,0,3,0,1,2,1,2,1,0], indicating that the alarm type 3 has occurred three times, the alarm types 6 and 8 each twice and the alarm types 5, 7 and 9 each once.

This count-based approach has the disadvantage that it does not show in which order the different alarms have occurred, which means that some information that may be important gets lost. Furthermore, it has a bigger dimension than the sequence data encoding. On the other hand, the length of each encoded sample is automatically the same, making it easier to deal with the base sequences of different lengths, as it is unnecessary to add NaN values to reach the same length for every data sample.

4.3.4 Sample Selection

When training classification models, sample selection is a crucial step towards a reliable model with good performance. Even if more data is fundamentally beneficial,

training models with the entire available data is not always necessary. Training processes can be time-consuming and need lots of computational resources, so carefully selecting training samples is essential. Table 10 shows how many samples for each class are available in the data set.

Alarm ID	Number of occurrences
7	49.959
15	517
16	276
21	118
33	14.168
34	2
39	32
56	7.068
68	342.485
95	274
1000	2.726
1001	177.830

Table 10: Number of occurrences for each critical alarm type

Since the representation of classes 34 and 39 is shallow, both classes have been removed from the project scope because there is not enough available data. As this thesis project is exploratory, different sample selection strategies are applied to compare their impact on the model performance. These are the following:

- **Training on all data:** The models are trained on the entire data. This gives a first overview of how the models perform on the original data and whether the fact that some classes are represented more often in the data than others causes problems.
- **Training only on unique samples:** Unique data set samples are selected for training. This decreases the computational load while training without removing samples, which might be important.
- **Random selection of samples:** To further reduce the need for computational resources while training, the classification models get trained with a random selection of samples. To ensure that all possible classes are part of the training data, we made the random sample selection not on the entire data set but by class. We selected ten thousand samples from each class and added the data of those classes less represented than this to the training set without limitation. As a side-effect, this sample selection approach reduces the imbalance of the training data.

4.3.5 Dealing with Imbalanced Data

An essential characteristic of a data set is the distribution of the available samples to the possible classes. In many cases, it is natural that some classes are represented more often in the available data set than others. Depending on the individual data set, those differences can cause performance issues, especially in predicting those classes for which much fewer data samples are available. Table 10 shows that the data set is imbalanced. The error types with the ID 7, 68, and 1001 are by far the most represented classes, while the representation of error types with the ID 34 and 39 is vanishingly small.

To overcome problems caused by imbalanced training data sets, it can be beneficial for each class to have the same representation in the training data. Several common techniques are available:

- Undersampling of the majority classes: Not all available data samples of the majority classes are part of the training data.
- Oversampling of the minority classes: More data samples are added to the minority classes. These are whether duplicates or synthetic samples.
- Regroup minority classes: Regroup minority classes into a more general group.

At first, the classification models get trained without any measures to reduce data imbalance. Afterward, the majority classes are undersampled indirectly with the random sample selection approach. Finally, those models with the best performance get trained again after oversampling the minority classes using SMOTE.

SMOTE As an oversampling method, *SMOTE* has been chosen. SMOTE stands for the *Synthetic Minority Oversampling Technique* and is a standard oversampling method when dealing with imbalanced data sets. Chawla, Bowyer, Hall, and Kegelmeyer [CBHK02] initially proposed SMOTE to overcome specific problems of the classic random oversampling method, which duplicated random samples of the minority classes. Although random oversampling led to better performance scores of classification models, it can also promote overfitting, leading to poor generalization performance. As stated by Fernandez et al. [FGHC18], the proposition of SMOTE considerably influenced the treatment of imbalanced data and became the base of many different variants and further oversampling algorithms. The main characteristic of this technique is that it does not duplicate data samples of the minority classes but creates new synthetic data points. It randomly chooses existing data points and defines their k nearest neighbors to interpolate a new data point between the original and selected sample. This creates new data samples to prevent the model from being trained on repeating data and increases the diversity of the training set.

4.4 Parameter Tuning

We did a hyperparameter tuning to find the parameters leading to each model's best performance. We did a grid search to tune the KNN and the random forest algorithm. A grid search is a common technique for hyperparameter tuning and tries to explore all possible combinations of the different parameters in the given search spaces and measures the model's performance with each parameter combination. At the end of the search, the grid search determines the parameter combination with which the model has achieved the best score. Instead of accuracy, we have used the F1-Score to measure the model performance as it is the harmonic mean of precision and recall. It, therefore, gives a better idea about the overall performance. Table 11 and Table 12 show the parameters that have worked best for the KNN and the random forest models.

	All Samples		Unique Samples		10,000 Samples	
Parameter	Sequence	Count-based	Sequence	Count-based	Sequence	Count-based
Metric	manhattan	euclidean	manhattan	euclidean	manhattan	manhattan
K	11	11	15	15	13	15

Table 11: The parameter values which lead to the best performance results of the KNN model.

	All Samples		Unique Samples		10,000 Samples	
Parameter	Sequence	Count-based	Sequence	Count-based	Sequence	Count-based
Criterion	log_loss	entropy	log_loss	entropy	entropy	log_loss
Max_features	7	7	7	9	7	7
Min_samples_leaf	3	2	3	2	3	2
Min_samples_split	3	2	3	2	4	2
Estimators	50	40	50	50	60	40

Table 12: The parameter values which lead to the best performance results of the random forest model.

To tune the FCN and the ResNet models, we have chosen to experiment with different parameters, such as batch size and kernel size, and compare the performance results. Furthermore, we have trained all models using early stopping to find the optimal number of training epochs before the model starts to overfit. Table 13 and Table 14 show the parameter values that have worked best.

5 Evaluation

The following section presents the performance results of each algorithm and each data encoding and sample selection approach. The models are evaluated using a train-test-split of the original data and a test of new unseen data.

	All Samples		Unique Samples		10,000 Samples	
Parameter	Sequence	Count-based	Sequence	Count-based	Sequence	Count-based
Batch size	32	64	32	64	32	64
Kernel size	3	3	3	9	3	9

Table 13: The parameter values which lead to the best performance results of the FCN model.

	All Samples		Unique Samples		10,000 Samples	
Parameter	Sequence	Count-based	Sequence	Count-based	Sequence	Count-based
Batch size	32	32	32	32	32	32
Kernel size	3	9	3	9	3	9

Table 14: The parameter values which lead to the best performance results of the ResNet model.

5.1 Train-Test Split

In the following section, the performance results of each classification algorithm after a train-test split are presented. The class support of the training and test set is shown for each sample selection method.

5.1.1 Training on All Data

Table 15 shows the support for each class, both in the training and test data set.

Class	Training	Test
7	49,959	16,479
15	517	186
16	276	84
21	118	40
33	14,168	4,535
56	7,068	2,301
68	342,485	113,157
95	274	90
1000	2,726	877
1001	177,830	58,740

Table 15: The support by class in training and test data

KNN The KNN model achieves an overall accuracy of 0.72 when trained on all the data with sequence encoding and without a specific sample selection. The macro average of the precision is 0.52, the recall is 0.36, and the f1-score is 0.40. The weighted

average of the precision is 0.70, the recall is 0.72, and the f1-score is 0.70. The model achieves an accuracy of 0.71 when using the count-based encoding. The macro average of the precision is 0.52, the recall is 0.37, and the f1-score is 0.42. The weighted average of the precision is 0.70, the recall is 0.71, and the f1-score is 0.70. See Table 16 for the results by class for both encoding methods.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.91	0.90	0.91	7	0.92	0.94	0.93
15	0.64	0.34	0.44	15	0.52	0.20	0.29
16	0.63	0.48	0.54	16	0.51	0.44	0.47
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.32	0.07	0.12	33	0.52	0.07	0.12
56	0.58	0.30	0.39	56	0.65	0.30	0.41
68	0.73	0.85	0.79	68	0.73	0.85	0.79
95	0.12	0.01	0.02	95	0.00	0.00	0.00
1000	0.68	0.32	0.44	1000	0.77	0.33	0.46
1001	0.60	0.46	0.52	1001	0.60	0.47	0.53

Table 16: Performance results of the KNN model trained on all data with count-based and sequence data encoding.

Random Forest After being trained on 70% of the entire data set and using the count-based encoding approach, the model achieves an accuracy of 0.72. The weighted average of the precision is 0.72, the recall is 0.72, and the f1-score is 0.67. The macro average of the precision is 0.64, the recall is 0.41, and the f1-score is 0.46. The model achieves an accuracy of 0.75 when using the sequence encoding approach. The weighted average of the precision is 0.74, the recall is 0.75, and the f1-score is 0.73. The macro average of the precision is 0.62, the recall is 0.41, and the f1-score is 0.46. Table 17 shows the results by class.

FCN The FCN achieves an accuracy of 0.74 when trained on all data using the sequences data encoding. The macro average of the precision is 0.56, the recall is 0.38, and the f1-score is 0.42. The weighted average of the precision is 0.73, the recall is 0.74, and the f1-score is 0.72. When trained using the count-based encoded data, the model has an accuracy of 0.66. The macro average of the precision is 0.43, the recall is 0.29, and the f1-score is 0.32. The weighted average of the precision is 0.63, the recall is 0.66, and the f1-score is 0.61. Table 18 shows the results by class.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.94	0.98	0.96	7	0.93	0.98	0.95
15	0.85	0.40	0.55	15	0.77	0.32	0.45
16	0.90	0.68	0.78	16	0.83	0.74	0.78
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.72	0.07	0.13	33	0.66	0.08	0.14
56	0.69	0.28	0.40	56	0.68	0.30	0.42
68	0.69	0.95	0.80	68	0.76	0.87	0.81
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.94	0.38	0.54	1000	0.91	0.36	0.52
1001	0.71	0.26	0.38	1001	0.65	0.52	0.58

Table 17: Performance results of the random forest model trained on all data with count-based and sequence data encoding.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.80	0.60	0.69	7	0.91	0.97	0.94
15	0.07	0.01	0.01	15	0.25	0.02	0.03
16	0.83	0.68	0.75	16	0.94	0.66	0.78
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.56	0.07	0.12	33	0.64	0.07	0.12
56	0.00	0.00	0.00	56	0.67	0.32	0.43
68	0.67	0.93	0.78	68	0.74	0.89	0.81
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.87	0.36	0.51	1000	0.80	0.40	0.54
1001	0.54	0.23	0.32	1001	0.65	0.46	0.54

Table 18: Performance results of the FCN model trained on all data with count-based and sequence data encoding.

ResNet The ResNet model achieves an overall accuracy of 0.73 when trained on all data using the count-based data encoding. The macro average of the precision is 0.60, the recall is 0.44, and the f1-score is 0.47. The weighted average of the precision is 0.72, the recall is 0.73, and the f1-score is 0.70. Using the sequence data encoding, the model reaches an accuracy of 0.75 with a macro average precision of 0.57, a recall of 0.38, and a f1-score of 0.43. The weighted average of the precision is 0.73,

the recall is 0.75, and the f1-score is 0.73. See Table 19 for details by class.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.94	0.98	0.96	7	0.91	0.97	0.94
15	0.70	0.56	0.62	15	0.31	0.09	0.13
16	0.74	0.75	0.75	16	0.86	0.65	0.74
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.70	0.07	0.12	33	0.64	0.07	0.12
56	0.65	0.30	0.41	56	0.65	0.32	0.43
68	0.72	0.90	0.80	68	0.76	0.87	0.81
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.92	0.39	0.55	1000	0.90	0.34	0.49
1001	0.65	0.41	0.50	1001	0.65	0.53	0.58

Table 19: Performance results of the ResNet model trained on all data with count-based and sequence data encoding.

5.1.2 Training on Unique Samples

In the following, the performance results of the models are presented when training only on unique data samples. Table 20 shows the sample support by class for the training and test set.

Class	Training	Test
7	13,231	4,351
15	303	96
16	202	71
21	86	29
33	8,456	2,746
56	3,906	1,290
68	192,301	63,539
95	181	50
1000	1,359	427
1001	85,482	28,218

Table 20: The support by class in training and test data

KNN The KNN model achieves an overall accuracy of 0.66 when trained on unique data samples only and with count-based encoding. The macro average of the preci-

sion is 0.40, the recall is 0.33, and the f1-score is 0.35. The weighted average of the precision is 0.62, the recall is 0.66, and the f1-score is 0.62. When using the sequence encoding, the model has an accuracy of 0.68. The macro average of the precision is 0.37, the recall is 0.26, and the f1-score is 0.28. The weighted average of the precision is 0.63, the recall is 0.68, and the f1-score is 0.63. See Table 21 for the results by class for both encoding methods.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.89	0.94	0.91	7	0.83	0.85	0.84
15	0.46	0.22	0.30	15	0.00	0.00	0.00
16	0.61	0.56	0.58	16	0.60	0.30	0.40
21	0.10	0.03	0.05	21	0.00	0.00	0.00
33	0.17	0.06	0.09	33	0.41	0.04	0.07
56	0.08	0.02	0.03	56	0.06	0.00	0.00
68	0.70	0.88	0.78	68	0.70	0.91	0.79
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.55	0.35	0.43	1000	0.54	0.28	0.37
1001	0.46	0.24	0.32	1001	0.51	0.24	0.33

Table 21: Performance results of the KNN model trained on unique samples using count-based and sequence data encoding.

Random Forest When trained only on unique samples of the data set, the random forest algorithm achieves an accuracy of 0.70 with the sequence data encoding. The macro average of the precision is 0.48, the recall is 0.32, and the f1-score is 0.34. The weighted average of the precision is 0.67, the recall is 0.70, and the f1-score is 0.67. The model also achieves an accuracy of 0.70 when using the count-based data encoding. The macro average of the precision is slightly better with 0.58, the recall is 0.33, and the f1-score is 0.36. The weighted average of the precision is 0.67, the recall is 0.70, and the f1-score is 0.63. Table 22 shows the results by class for both encoding approaches.

FCN Trained on unique samples using the count-based data encoding, the FCN model achieves an accuracy of 0.69. The macro average of the precision is 0.51, the recall is 0.35, and the f1-score is 0.37. The weighted average of the precision is 0.67, the recall is 0.69, and the f1-score is 0.62. With the sequence data encoding, the model has an accuracy of 0.70. The macro average of the precision is 0.49, the recall is 0.32, and the f1-score is 0.34. The weighted average of the precision is 0.67, the recall is 0.70, and the f1-score is 0.65.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.88	0.99	0.93	7	0.86	0.96	0.91
15	0.81	0.26	0.39	15	0.46	0.06	0.11
16	0.85	0.58	0.69	16	0.81	0.59	0.68
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.70	0.04	0.08	33	0.54	0.04	0.07
56	0.50	0.00	0.01	56	0.16	0.00	0.01
68	0.69	0.96	0.80	68	0.72	0.90	0.80
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.75	0.36	0.48	1000	0.69	0.27	0.38
1001	0.60	0.17	0.26	1001	0.56	0.34	0.42

Table 22: Performance results of the random forest model trained on unique samples using count-based and sequence data encoding.

Table 23 shows the results by class.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.89	0.96	0.92	7	0.84	0.96	0.90
15	0.64	0.38	0.47	15	0.21	0.07	0.11
16	0.77	0.56	0.65	16	0.80	0.56	0.66
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.60	0.04	0.08	33	0.62	0.05	0.08
56	0.34	0.01	0.03	56	0.40	0.01	0.03
68	0.69	0.96	0.81	68	0.71	0.93	0.81
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.60	0.42	0.49	1000	0.72	0.34	0.47
1001	0.61	0.15	0.23	1001	0.58	0.27	0.36

Table 23: Performance results of the FCN model trained on unique samples using count-based and sequence data encoding.

ResNet After training on unique samples using the count-based data encoding, the ResNet has an accuracy of 0.70. It achieves a macro average of 0.55 for the precision, 0.35 for the recall, and 0.38 for the f1-score. The weighted average of the

precision is 0.67, the recall is 0.70, and the f1-score is 0.63. Using the sequence data encoding, the model achieves an accuracy of 0.72 with a macro average of 0.46 for the precision, 0.32 for the recall, and 0.34 for the f1-score. The weighted average of the precision is 0.69, the recall is 0.72, and the f1-score is 0.69. Table 24 shows the results by class for both encoding approaches.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.89	0.98	0.93	7	0.85	0.96	0.90
15	0.68	0.47	0.56	15	0.22	0.02	0.04
16	0.84	0.52	0.64	16	0.68	0.55	0.61
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.61	0.05	0.09	33	0.54	0.03	0.07
56	0.43	0.00	0.01	56	0.31	0.05	0.08
68	0.70	0.95	0.81	68	0.74	0.89	0.81
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.72	0.39	0.50	1000	0.68	0.29	0.40
1001	0.59	0.18	0.28	1001	0.58	0.40	0.47

Table 24: Performance results of the ResNet model trained on unique samples using count-based and sequence data encoding.

5.1.3 Training on Randomly Selected Samples

The following section shows the results when the models get trained on a data set that consists of 10,000 randomly selected samples by class or less if fewer samples of the respective class are available. In total, 50,982 samples are available. Table 25 shows the support by class.

KNN Training the KNN model with 10,000 randomly selected samples and using the sequence encoding results in a total accuracy of 0.52. The macro average of the precision is 0.43, the recall is 0.39, and the f1-score is 0.40. The weighted average of the precision is 0.53, the recall is 0.52, and the f1-score is 0.52. Combined with the count-based data encoding, the KNN model has an accuracy of 0.53. The macro average of the precision is 0.52, the recall is 0.46, and the f1-score is 0.47. The weighted average of the precision is 0.54, the recall is 0.53, and the f1-score is 0.53. Table 26 shows the results by class for both encoding methods.

Random Forest After training the random forest model on 10,000 randomly selected samples and using the count-based encoding approach, it achieves an accu-

Class	Training	Test
7	10,000	3,376
15	517	173
16	276	83
21	118	37
33	10,000	3,303
56	7,068	2,284
68	10,000	3,279
95	274	76
1000	2,726	910
1001	10,000	3,303

Table 25: The support by class in training and test data

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.88	0.92	0.90	7	0.84	0.90	0.87
15	0.71	0.58	0.64	15	0.50	0.32	0.39
16	0.66	0.76	0.71	16	0.41	0.55	0.47
21	0.38	0.08	0.13	21	0.00	0.00	0.00
33	0.34	0.40	0.37	33	0.36	0.45	0.40
56	0.58	0.40	0.47	56	0.61	0.39	0.48
68	0.41	0.44	0.42	68	0.41	0.45	0.43
95	0.05	0.01	0.02	95	0.00	0.00	0.00
1000	0.82	0.52	0.64	1000	0.72	0.45	0.55
1001	0.41	0.45	0.43	1001	0.42	0.40	0.41

Table 26: Performance results of the KNN model trained on 10,000 selected samples with count-based and sequence data encoding.

racy of 0.56. The macro average of the precision is 0.55, the recall is 0.46, and the f1-score is 0.48. The weighted average of the precision is 0.60, the recall is 0.56, and the f1-score is 0.55. When training using the sequence encoding approach, the model has an accuracy of 0.57. The macro average of the precision is 0.53, the recall is 0.45, and the f1-score is 0.47. The weighted average of the precision is 0.59, the recall is 0.57, and the f1-score is 0.56. Table 27 shows the results by class for both encoding approaches.

FCN The FCN achieves an accuracy of 0.55 when trained on 10,000 selected samples using the sequences data encoding. The macro average of the precision is 0.48,

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.45	0.97	0.90	7	0.85	0.93	0.89
15	0.81	0.61	0.70	15	0.78	0.39	0.52
16	0.82	0.75	0.78	16	0.77	0.78	0.78
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.43	0.22	0.30	33	0.41	0.36	0.38
56	0.88	0.36	0.51	56	0.77	0.40	0.53
68	0.41	0.63	0.50	68	0.44	0.60	0.50
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.87	0.52	0.65	1000	0.83	0.50	0.62
1001	0.41	0.57	0.48	1001	0.46	0.54	0.50

Table 27: Performance results of the random forest model trained on 10,000 randomly selected samples using the count-based and sequence data encoding.

the recall is 0.43, and the f1-score is 0.44. The weighted average of the precision is 0.58, the recall is 0.55, and the f1-score is 0.55. When trained using the count-based encoded data, the model has an accuracy of 0.56. The macro average of the precision is 0.53, the recall is 0.47, and the f1-score is 0.48. The weighted average of the precision is 0.59, the recall is 0.56, and the f1-score is 0.55. Table 28 shows the results by class.

ResNet The ResNet achieves an accuracy of 0.55 when trained on 10,000 selected samples using the sequences data encoding. The macro average of the precision is 0.46, the recall is 0.44, and the f1-score is 0.44. The weighted average of the precision is 0.55, the recall is 0.55, and the f1-score is 0.55. When trained using the count-based encoded data, the model has an accuracy of 0.56. The macro average of the precision is 0.54, the recall is 0.47, and the f1-score is 0.48. The weighted average of the precision is 0.60, the recall is 0.56, and the f1-score is 0.56. Table 29 shows the results by class.

The results show that those models trained on the entire data set perform better in terms of accuracy and weighted average precision, recall, and f1-score. The random forest and the ResNet model, both with sequence encoding, achieve the highest results with an accuracy of 0.75. The models trained on 10,000 randomly selected samples achieved the lowest performance with an accuracy between 0.52 and 0.55. Those models trained on unique data samples have mixed performance results. Random forest, the FCN, and the ResNet models perform better than the KNN models.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.89	0.91	0.90	7	0.88	0.86	0.87
15	0.79	0.70	0.74	15	0.44	0.32	0.37
16	0.67	0.78	0.72	16	0.74	0.73	0.74
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.41	0.29	0.34	33	0.39	0.36	0.37
56	0.83	0.38	0.52	56	0.82	0.38	0.52
68	0.43	0.55	0.48	68	0.43	0.57	0.49
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.84	0.50	0.62	1000	0.64	0.56	0.60
1001	0.40	0.62	0.49	1001	0.43	0.53	0.48

Table 28: Performance results of the FCN model trained on 10,000 samples with count-based and sequence data encoding.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.90	0.91	0.91	7	0.84	0.92	0.88
15	0.74	0.69	0.71	15	0.38	0.32	0.35
16	0.58	0.77	0.66	16	0.59	0.75	0.66
21	0.25	0.03	0.05	21	0.00	0.00	0.00
33	0.42	0.27	0.33	33	0.39	0.38	0.38
56	0.86	0.37	0.52	56	0.64	0.45	0.53
68	0.42	0.68	0.52	68	0.46	0.47	0.47
95	0.00	0.00	0.00	95	0.11	0.09	0.10
1000	0.78	0.49	0.60	1000	0.71	0.49	0.58
1001	0.44	0.54	0.48	1001	0.43	0.54	0.48

Table 29: Performance results of the ResNet trained on 10,000 samples with count-based and sequence data encoding.

5.2 New Data

In the following we show the results of the models that achieved an accuracy of more than 0.70 when testing them on new data. As the performance of those models trained on 10,000 randomly selected data items was lower than this, they are not considered anymore as it is not likely that they achieve satisfying results on unseen

data. The same applies to the FCN model with count-based encoding trained on all data and unique samples, as well as to the KNN model trained on unique data with both encoding approaches. Table 30 shows the support for each class in the new test set.

Class	Test
7	3,287
15	28
16	1
21	7
33	3
56	1,658
68	44,424
95	44
1000	96
1001	30,917

Table 30: The support by class the test data set.

5.2.1 Models Trained on All Data

The following section shows the performance results for the models trained on all data.

KNN When testing new data, the KNN model trained on all data achieves an accuracy of 0.68 with count-based encoding. The macro average of the precision is 0.35, the recall is 0.27, and the f1-score is 0.29. The weighted average of the precision is 0.63, the recall is 0.68, and the f1-score is 0.66. With the sequence encoding, the model achieves an accuracy of 0.70 on the test data. The macro average of the precision is 0.32, the recall is 0.24, and the f1-score is 0.25. The weighted average of the precision is 0.70, the recall is 0.70, and the f1-score is 0.68. Table 31 shows the results by class.

Random Forest When testing new data, the random forest model trained on all data achieves an accuracy of 0.68 with count-based encoding. The macro average of the precision is 0.49, the recall is 0.31, and the f1-score is 0.32. The weighted average of the precision is 0.70, the recall is 0.68, and the f1-score is 0.64. With the sequence encoding, the model achieves an accuracy of 0.74 on the test data. The macro average of the precision is 0.40, the recall is 0.27, and the f1-score is 0.30. The weighted average of the precision is 0.75, the recall is 0.74, and the f1-score is 0.73. Table 32 shows the results by class.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.95	0.93	0.94	7	0.89	0.80	0.84
15	0.50	0.18	0.26	15	0.00	0.00	0.00
16	0.00	0.00	0.00	16	0.00	0.00	0.00
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.00	0.00	0.00	33	0.00	0.00	0.00
56	0.48	0.20	0.28	56	0.62	0.21	0.31
68	0.68	0.84	0.75	68	0.69	0.89	0.78
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.24	0.09	0.14	1000	0.33	0.03	0.06
1001	0.65	0.45	0.53	1001	0.71	0.46	0.56

Table 31: Performance results of the KNN model trained on all data with count-based and sequence data encoding when tested with new unseen data.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.94	0.97	0.95	7	0.94	0.92	0.93
15	0.95	0.68	0.79	15	0.38	0.11	0.17
16	0.00	0.00	0.00	16	0.00	0.00	0.00
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.00	0.00	0.00	33	0.00	0.00	0.00
56	0.64	0.17	0.27	56	0.67	0.21	0.31
68	0.65	0.93	0.77	68	0.73	0.89	0.80
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	1.00	0.01	0.02	1000	0.50	0.06	0.11
1001	0.74	0.31	0.44	1001	0.76	0.54	0.63

Table 32: Performance results of the random forest model trained on all data with count-based and sequence data encoding when tested with new unseen data.

FCN With the sequence encoding, the FCN model achieves an accuracy of 0.73 on the test data. The macro average of the precision is 0.35, the recall is 0.27, and the f1-score is 0.28. The weighted average of the precision is 0.73, the recall is 0.73, and the f1-score is 0.71. Table 33 shows the results by class.

Sequences			
Class	Precision	Recall	F1-score
7	0.90	0.90	0.90
15	0.21	0.14	0.17
16	0.00	0.00	0.00
21	0.00	0.00	0.00
33	0.00	0.00	0.00
56	0.64	0.21	0.32
68	0.72	0.89	0.79
95	0.00	0.00	0.00
1000	0.33	0.03	0.06
1001	0.74	0.51	0.61

Table 33: Performance results of the FCN model trained on all data with sequence data encoding and tested with new unseen data.

ResNet When testing new data, the ResNet model trained on all data achieves an accuracy of 0.70 with count-based encoding. The macro average of the precision is 0.41, the recall is 0.33, and the f1-score is 0.34. The weighted average of the precision is 0.71, the recall is 0.70, and the f1-score is 0.69. With the sequence encoding, the model achieves an accuracy of 0.75 on the test data. The macro average of the precision is 0.33, the recall is 0.26, and the f1-score is 0.27. The weighted average of the precision is 0.74, the recall is 0.75, and the f1-score is 0.74. Table 34 shows the results by class.

5.2.2 Models Trained on Unique Samples

The following section presents the test results of the models trained on unique samples.

Random Forest When testing new data, the random forest model trained on unique samples achieves an accuracy of 0.66 with count-based encoding. The macro average of the precision is 0.39, the recall is 0.29, and the f1-score is 0.28. The weighted average of the precision is 0.66, the recall is 0.66, and the f1-score is 0.60. With the sequence encoding, the model achieves an accuracy of 0.63 on the test data. The macro average of the precision is 0.25, the recall is 0.23, and the f1-score is 0.20. The weighted average of the precision is 0.63, the recall is 0.63, and the f1-score is 0.62. Table 35 shows the results by class.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.96	0.96	0.96	7	0.90	0.90	0.90
15	0.61	0.71	0.66	15	0.00	0.00	0.00
16	0.00	0.00	0.00	16	0.00	0.00	0.00
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.00	0.00	0.00	33	0.00	0.00	0.00
56	0.64	0.22	0.32	56	0.62	0.22	0.33
68	0.69	0.88	0.77	68	0.75	0.86	0.80
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.54	0.07	0.13	1000	0.33	0.03	0.06
1001	0.71	0.45	0.55	1001	0.73	0.60	0.66

Table 34: Performance results of the ResNet model trained on all data with count-based and sequence data encoding when tested with new unseen data.

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.69	0.98	0.81	7	0.54	0.91	0.68
15	0.95	0.68	0.79	15	0.00	0.00	0.00
16	0.00	0.00	0.00	16	0.00	0.00	0.00
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.00	0.00	0.00	33	0.00	0.00	0.00
56	0.50	0.00	0.00	56	0.74	0.05	0.10
68	0.65	0.92	0.76	68	0.68	0.78	0.73
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.40	0.02	0.04	1000	0.01	0.09	0.01
1001	0.68	0.28	0.39	1001	0.58	0.42	0.49

Table 35: Performance results of the random forest model trained on unique samples with count-based and sequence data encoding when tested with new unseen data.

FCN With the sequence encoding, the FCN model achieves an accuracy of 0.65 on the test data. The macro average of the precision is 0.30, the recall is 0.22, and the f1-score is 0.19. The weighted average of the precision is 0.68, the recall is 0.65, and the f1-score is 0.61. Table 36 shows the results by class.

Sequences			
Class	Precision	Recall	F1-score
7	0.40	0.94	0.56
15	0.33	0.04	0.06
16	0.00	0.00	0.00
21	0.00	0.00	0.00
33	0.00	0.00	0.00
56	0.60	0.03	0.06
68	0.67	0.90	0.77
95	0.00	0.00	0.00
1000	0.23	0.03	0.06
1001	0.72	0.28	0.40

Table 36: Performance results of the FCN model trained on unique samples with sequence data encoding and tested with new unseen data.

ResNet When testing new data, the ResNet model trained on unique data samples achieves an accuracy of 0.63 with count-based encoding. The macro average of the precision is 0.36, the recall is 0.29, and the f1-score is 0.27. The weighted average of the precision is 0.62, the recall is 0.63, and the f1-score is 0.61. With the sequence encoding, the model achieves an accuracy of 0.66 on the test data. The macro average of the precision is 0.22, the recall is 0.23, and the f1-score is 0.20. The weighted average of the precision is 0.70, the recall is 0.66, and the f1-score is 0.67. Table 37 shows the results by class.

5.3 The Effects of SMOTE

On all tested models, SMOTE improved the model performance of the train-test split but decreased the performance when testing with new and unseen data. The following section shows the effect of SMOTE on some of our trained models.

The random forest model trained on all data with SMOTE achieves an accuracy of 0.68 on the test data, compared to 0.74 when trained without SMOTE. Table 38 shows the results by class.

The FCN trained on all data with SMOTE achieves an accuracy of 0.60 on the test data, compared to 0.73 when trained without SMOTE. Table 39 shows the results by class.

The ResNet trained on all data with SMOTE achieves an accuracy of 0.62 on the test data, compared to 0.75 when trained without SMOTE. Table 40 shows the results by class.

SMOTE did not affect the weighted and macro average values of the precision,

Count-based				Sequences			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.55	0.97	0.70	7	0.41	0.88	0.56
15	0.71	0.61	0.65	15	0.00	0.00	0.00
16	0.00	0.00	0.00	16	0.00	0.00	0.00
21	0.00	0.00	0.00	21	0.00	0.00	0.00
33	0.00	0.00	0.00	33	0.00	0.00	0.00
56	0.54	0.01	0.02	56	0.05	0.14	0.08
68	0.67	0.80	0.73	68	0.74	0.77	0.75
95	0.00	0.00	0.00	95	0.00	0.00	0.00
1000	0.53	0.10	0.17	1000	0.25	0.03	0.05
1001	0.57	0.37	0.45	1001	0.70	0.51	0.59

Table 37: Performance results of the ResNet model trained on unique data with count-based and sequence data encoding when tested with new unseen data.

Train-Test Split				New Data			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.95	0.92	0.93	7	0.93	0.82	0.87
15	0.91	0.96	0.93	15	0.06	0.61	0.11
16	0.99	0.98	0.98	16	0.00	0.00	0.00
21	0.88	0.97	0.93	21	0.01	0.14	0.01
33	0.76	0.69	0.72	33	0.00	0.00	0.00
56	0.95	0.84	0.89	56	0.28	0.27	0.28
68	0.66	0.75	0.70	68	0.75	0.81	0.78
95	0.79	0.94	0.86	95	0.00	0.30	0.01
1000	0.89	0.82	0.85	1000	0.07	0.28	0.11
1001	0.65	0.54	0.59	1001	0.72	0.49	0.58

Table 38: Performance results of the random forest model trained on all data with sequence data encoding when tested with new unseen data using SMOTE.

recall, and f1-scores. The macro averages remain lower than the weighted averages.

Train-Test Split				New Data			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.88	0.88	0.88	7	0.80	0.78	0.79
15	0.89	0.93	0.91	15	0.03	0.29	0.05
16	0.97	0.97	0.97	16	0.00	0.00	0.00
21	0.88	0.95	0.91	21	0.00	0.00	0.00
33	0.58	0.41	0.48	33	0.00	0.00	0.00
56	0.75	0.70	0.72	56	0.14	0.35	0.20
68	0.57	0.72	0.64	68	0.73	0.72	0.72
95	0.76	0.93	0.84	95	0.00	0.34	0.01
1000	0.82	0.78	0.80	1000	0.05	0.36	0.09
1001	0.51	0.40	0.45	1001	0.61	0.43	0.50

Table 39: Performance results of the FCN trained on all data with sequence data encoding when tested with new unseen data using SMOTE.

Train-Test Split				New Data			
Class	Precision	Recall	F1-score	Class	Precision	Recall	F1-score
7	0.94	0.91	0.92	7	0.84	0.81	0.83
15	0.90	0.96	0.93	15	0.05	0.61	0.10
16	0.98	0.98	0.98	16	0.00	0.00	0.00
21	0.87	0.97	0.92	21	0.01	0.14	0.01
33	0.72	0.70	0.71	33	0.00	0.00	0.00
56	0.93	0.84	0.88	56	0.20	0.27	0.23
68	0.66	0.67	0.67	68	0.75	0.70	0.72
95	0.78	0.94	0.85	95	0.00	0.30	0.01
1000	0.86	0.82	0.84	1000	0.02	0.24	0.03
1001	0.62	0.52	0.57	1001	0.64	0.49	0.56

Table 40: Performance results of the ResNet model trained on all data with sequence data encoding when tested with new unseen data using SMOTE.

6 Discussion

When evaluating each model's performance to decide which achieves the best results, we focus on the performance with new and unseen test data because this represents the purpose of integrating machine learning models into maintenance processes. They must deal primarily with new incoming data, which was not part

of the training process, and make reliable predictions.

The best performing models During our tests, the models with the best performance on the test data are the FCN with an accuracy of 0.73, the ResNet with an accuracy of 0.74, and the random forest model with an accuracy of 0.75, all trained on the entire data set and with sequence encoding. The results have shown that the models trained on the whole data set achieve better results than those trained on unique samples and the models trained on 10,000 randomly selected samples. Furthermore, the sequence encoding approach worked better than the count-based approach.

Overall performance The accuracy between 0.73 and 0.75 shows that the prediction works correctly for 73 to 75 % of the samples. However, the relatively low macro average of the precision, recall, and f1-score shows that the prediction works better for classes with many data samples than for those with few samples, which is a typical problem of imbalanced data sets. The higher values of the weighted average of the precision, recall, and f1-score confirm this. The prediction works best for the following alarm types:

- 7 - The limit of payment machines that are out of service has been exceeded.
- 68 - A transaction is blocked. A customer validated a payment transaction, but the fueling process was never registered as finished.
- 1001 - A Payment machine is inactive.

Alarms such as unwanted restarts of the payment machine, refused transactions, non-working fuel pumps, and the control unit's automatic deactivation of a payment machine lead to these critical alarm types. The alarm types for which the prediction does not work well are the following:

- 15 - The service station is disconnected.
- 16 - An incident prevents the service station from starting.
- 21 - A measuring unit is out of service.
- 33 - A fuel dispenser dispensed fuel for more money than authorized.
- 95 - Several payment machines have the same terminal ID.

The poor performance for these alarm types might not be representative, as it concerns the minority classes for which the test data did not contain many samples. For example, the prediction of alarm type 16 was better in the train-test split than in the test with new unseen data.

Possible reasons for the performance One primary reason for the mixed performance is the imbalanced data set and the fact that we have many critical alarm types for which we do not have enough data. This is visible in the classification reports, which show that the prediction generally works better for classes with higher support in the data than for those with few data samples. It would be possible to regroup these minority classes into a more general class. However, in the context of maintenance processes in which maintenance activities should be scheduled automatically based on the prediction of a machine learning model, this measure would not be helpful. The models trained on 10,000 randomly selected samples did not perform well because the data set contains several hundreds of thousands of samples for some classes. Only selecting a low range of samples means a significant loss of information. On the other hand, the upsampling of the minority classes with SMOTE did not increase but even decreased the performance because the available data of the minority class was so low that the models were trained with a significant amount of synthetic samples. Those samples do not necessarily represent reality. Furthermore, the fact that SMOTE increased the performance in the train-test split but decreased the performance in the test with unseen data suggests an overfitting of those models trained with SMOTE. The data itself could be more diverse as it is already challenging to base machine learning models on log data. However, the fact that the logs contain only alarms is hardening the conditions of making predictions, and adding other log data of each machine could give a better insight into what happens before the occurrence of critical problems. Furthermore, the service stations from which we collected the data are very different in size and equipment. Not every service station produces all possible alarm types. Even if most service stations have the same technological standards, deviations are possible, as behind every service station stands a client who makes economic decisions. Lastly, another reason for a too-low performance with the test data is that many technological evolutions occur throughout the year, both on-site and in the systems of MADIC. New alarm types are created, and others disappear. This is a challenge because a machine learning model constantly needs training on new data to ensure its reliability.

7 Conclusion

In this project, we analyzed whether machine learning-based classification can be integrated into maintenance processes to predict critical equipment failures of service stations. We explained and applied different classification methods to the same data set using different data encoding and sample selection methods. The results show that the prediction works best for those classes with many data samples and that the data imbalance leads to its typical challenges.

In conclusion, this thesis project fully achieved its objective of applying and comparing different machine learning algorithms on alarm log data to make predictions of critical alarm types. Furthermore, it partly accomplished the objective of developing a classification model that makes reliable predictions, as the prediction only

works for some classes. The project gave an interesting insight into the supervised learning process and is an excellent example of connecting science to real-life use scenarios. Even if the results are not strong enough yet to be applied to actual maintenance processes, they form a base for future research and development in the field of predictive maintenance. The low diversity of the data and the lack of data samples for certain classes are principal challenges to overcome to reduce the effects of data imbalance and to increase the models' reliability. Considering the thesis results as the beginning of further projects, future work could explore the characteristics of the given data more profoundly and enrich it with non-error-related information. The increasing connectivity of the service stations, as planned by MADIC, will add further informational content and variety to the base data as well. Furthermore, a completely different approach to identifying failure-leading patterns in log data is conceivable, such as Sequential Pattern Mining and Chronicle Pattern Mining as described in Section 3.1.

References

[AC16] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367, September 2016.

[AFGY02] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential PAttern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, July 2002.

[Agg18] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer International Publishing, 2018.

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, June 1993.

[AM07] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection, 2007.

[AS] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*. IEEE Comput. Soc. Press.

[Bra] M. Bramer. Avoiding overfitting of decision trees. In *Principles of Data Mining*, pages 119–134. Springer London.

[BW20] Gerrit J. J. van den Burg and Christopher K. I. Williams. An evaluation of change point detection algorithms, 2020.

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[CD99] T. V. Duong Christophe Dousson. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 620–626, 1999.

[CSZM+20] Qiushi Cao, Ahmed Samet, Cecilia Zanni-Merk, François de Bertrand de Beuvron, and Christoph Reich. Combining chronicle mining and semantics for predictive maintenance in manufacturing processes. *Semantic Web*, 11(6):927–948, October 2020.

[DLZS17] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2017.

[DWP+22] Zahra Zamanzadeh Darban, Geoffrey I. Webb, Shirui Pan, Charu C. Aggarwal, and Mahsa Salehi. Deep learning for time series anomaly detection: A survey, 2022.

[EHH98] David J. Edwards, Gary D. Holt, and F.C. Harris. Predictive maintenance techniques and their relevance to construction plant. *Journal of Quality in Maintenance Engineering*, 4(1):25–37, March 1998.

[EIB17] Hasnida Ab-Samat Shahrul Kamaruddin Ernnie Illyani Basri, Izatul Hamimi Abdul Razak. Preventive maintenance (pm) planning: a review. *Journal of Quality in Maintenance Engineering*, 77(2):114–143, 2017.

[FGHC18] Alberto Fernandez, Salvador Garcia, Francisco Herrera, and Nitesh V. Chawla. SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61:863–905, April 2018.

[GMMM08] L. Garg, S. McClean, B. Meenan, and P. Millard. Non-homogeneous markov models for sequential pattern mining of healthcare data. *IMA Journal of Management Mathematics*, 20(4):327–344, November 2008.

[GVW20] Antoine Guillaume, Christel Vrain, and Elloumi Wael. Predictive maintenance on event logs: Application on an atm fleet, 2020.

[Hin70] David V. Hinkley. Inference about the change-point in a sequence of random variables. *Biometrika*, 57(1):1–17, April 1970.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[JDK20] Aoife D'Arcy John D Kelleher, Brian Mac Namee. *Fundamentals of Machine Learning For Predictive Data Analytics*, volume 2. MIT Press, October 2020.

[KD18] Jeremias Knoblauch and Theodoros Damoulas. Spatio-temporal bayesian on-line changepoint detection with model selection, 2018.

[KFE12] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, October 2012.

[KKKK21] Abdulgani Kahraman, Mehmed Kantardzic, M. Mustafa Kahraman, and Muhammed Kotan. Sequential pattern mining method for predictive maintenance of large mining trucks. In *Communications in Computer and Information Science*, pages 126–136. Springer International Publishing, 2021.

[Liu11] Bing Liu. *Web Data Mining - Exploring Hyperlinks, Contents and Usage Data*. Springer, July 2011.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[OZGS22] Abdelfettah Ouadah, Leila Zemmouchi-Ghomari, and Nedjma Salhi. Selecting an appropriate supervised machine learning algorithm for predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 119(7-8):4277–4301, January 2022.

[PFV14] Manuel Campos Rincy Thomas Philippe Fournier-Viger, Antonio Gomariz. Fast vertical mining of sequential patterns using co-occurrence information. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2014.

[PFV17] Rage Uday Kiran Yun Sing Koh Rincy Thomas Philippe Fournier-Viger, Jerry Chun-Wei Lin. A survey of sequential pattern mining. *Ubiquitous International*, pages 54–77, February 2017.

[RS96] R Agrawal R. Srikant. Mining sequential patterns: Generalizations and performance improvements. pages 1–17, 1996.

[SFMW14] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, August 2014.

[SK74] A. J. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30(3):507, September 1974.

[SMS+19] Chayma Sellami, Carlos Miranda, Ahmed Samet, Mohamed Anis Bach Tobji, and François de Beuvron. On mining frequent chronicles for machine failure prediction. *Journal of Intelligent Manufacturing*, 31(4):1019–1035, September 2019.

[SST18] Chayma Sellami, Ahmed Samet, and Mohamed Anis Bach Tobji. Frequent chronicle mining: Application on predictive maintenance. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, December 2018.

[YHA03] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining: Closed sequential patterns in large datasets. In *Proceedings of the 2003 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, May 2003.

[Zak01] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42, June 2001.