Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Prediction of failures of IoT devices by using machine learning

Amil Imeri

Course of Study:	Informatik
Examiner:	PD Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Daniel Del Gaudio, M.Sc. Dr. rer. nat. Pascal Hirmer
Commenced:	December 8, 2020
Completed:	July 6, 2021

Abstract

In the era of the *Internet of Things (IoT)*, everyday objects are equipped with sensors and actuators. They are also referred to as IoT devices that collect data and perform actions by communicating with each other. The resource-constrained IoT devices may be put under significant stress or external influences that may lead to their failures. In IoT environments where safety is a critical requirement, the failures need to be prevented before they can cause any harm. By implementing *Predictive Maintenance (PdM)* with *Machine Learning (ML)*, different methods can be applied to predict the failures and provide enough time for maintenance. IoT devices collect data that can be used to train the ML algorithms to recognize the failure patterns of individual devices. This thesis presents the *Failure Prediction Platform (FPP)*, a platform that enables simple integration and management of ML models trained for failure prediction of IoT devices. Furthermore, different ML algorithms are compared regarding their failure prediction performance and their training and prediction speed. Based on the evaluation results, the tree-based algorithms showed the best performance in predicting failures, while the linear classifiers had the worst results. Finally, a prototype of the FPP is implemented and presented by using *Long Short-Term Memory (LSTM)* as an online learning approach to make time series predictions.

Kurzfassung

Im Zeitalter des *Internets der Dinge (IoT)* werden Alltagsgegenstände mit Sensoren und Aktuatoren ausgestattet. Sie werden auch als IoT-Geräte bezeichnet, die Daten sammeln und Aktionen ausführen, indem sie miteinander kommunizieren. Die ressourcenbeschränkten IoT-Geräte können einer erheblichen Belastung oder äußeren Einflüssen ausgesetzt werden, die zu ihrem Ausfall führen können. In IoT-Umgebungen in der Sicherheit eine wichtige Anforderung ist, müssen die Ausfälle verhindert werden bevor sie irgendeinen Schaden anrichten können. Durch die Implementierung von *Predictive Maintenance (PdM)* mit *Maschinelles Lernen (ML)* können verschiedene Methoden angewendet werden, um die Ausfälle vorherzusagen und genügend Zeit für die Wartung bereitzustellen. Die IoT-Geräte sammeln Daten, mit denen die ML-Algorithmen trainiert werden können, um die Ausfallmuster der einzelnen Geräte zu erkennen. In dieser Arbeit wird die *Failure Prediction Platform (FPP)* vorgestellt, die eine einfache Integration und Verwaltung von ML-Modellen, trainiert für die Vorhersage von Ausfällen von IoT-Geräten, ermöglicht. Zusätzlich werden verschiedene ML-Algorithmen hinsichtlich ihrer Leistung bei Ausfallvorhersagen und ihrer Geschwindigkeit beim Trainieren und Vorhersagen verglichen. Basierend auf den Bewertungsergebnissen, zeigten die baumbasierten Algorithmen die beste und lineare Klassifikatoren die schlimmste Leistung bei der Vorhersage von Ausfällen. Schließlich wird ein Prototyp der FPP implementiert und bei der Verwendung von *Long Short-Term Memory (LSTM)* vorgestellt. LSTM wurde als ML-Modell integriert, das online lernt und Zeitreihenvorhersagen macht.

Contents

List of Figures

List of Tables

Acronyms

AE Autoencoder. 26

AI Artificial Intelligence. 19

ANN Artificial Neural Network. 13

API Application Programming Interface. 29, 31, 58, 59

ARIMA Auto-Regressive Integrated Moving Average. 24

BN Bayesian Network. 26

CNN Convolutional Neural Network. 23

CPU Central Processing Unit. 23, 29, 33, 34, 35, 47

DBN Deep Belief Network. 26

DES Double Exponential Smoothing. 25

DL Deep Learning. 20

DNN Deep Neural Network. 20

DT Decision Tree. 25

FNN Feedforward Neural Network. 26

FPP Failure Prediction Platform. 14

GAN Generative Adversarial Network. 25

GBM Gradient Boosting Machine. 26

HDD Hard Disk Drive. 23, 25

HTTP Hypertext Transfer Protocol. 59

IC Integrated Circuit. 24

IIoT Industrial IoT. 17

IoT Internet of Things. 13

IT Information Technology. 14, 23, 25, 26

kNN k-Nearest Neighbor. 23

LGR Logistic Regression. 26

LR Linear Regression. 24

LSTM Long Short-Term Memory. 23

MBP Multi-Purpose Binding and Provisioning Platform. 14

MCU Microcontroller Unit. 24

ML Machine Learning. 13

MLP Multi-Layer Perceptron. 25

MQTT Message Queuing Telemetry Transport. 19

NB Naive Bayes. 42

NN Neural Network. 13

PCA Principal Component Analysis. 39

PdM Predictive Maintenance. 13

PM Preventive Maintenance. 21

PNN Probabilistic Neural Network. 24

QoS Quality of Service. 24

R Regression. 26

RAM Random Access Memory. 47

REST Representational State Transfer. 59

RF Random Forest. 13

RM Reactive Maintenance. 21

RNN Recurrent Neural Network. 23

RUL Remaining Useful Life. 22

SGD Stochastic Gradient Descent. 43

SSD Solid-State Drive. 25

SVM Support Vector Machine. 13

UI User Interface. 18, 55

WLAN Wireless Local Area Network. 24

XGB Extreme Gradient Boosting. 25

1 Introduction

With the emergence of the *Internet of Things (IoT)* [VF13], the current environments become smarter and maintain devices that can communicate with each other in an autonomous manner. They typically contain sensors that collect data from the environment, or use actuators to perform actions in the environment. In a *Smart Home* [RT17], an HVAC (heating, ventilation, air conditioning) system can be installed to control the room temperature, humidity and general air quality. It provides comfort for the occupants by continuously monitoring the room condition and performing necessary actions. The system typically uses temperature and humidity sensors to get the necessary information, and then automatically activate the heater, ventilator or air conditioner based on the predefined user conditions. In a *Smart Factory* [CWS+18], IoT devices can be used to monitor the production processes and machines. The acquired data provides insights into the production progress and can also be used to detect any anomalies in the machines or processes. Furthermore, IoT devices can be deployed in extreme environments with big limitations and challenges [GO21]. In case of precision agriculture, the devices can be placed underground to monitor soil moisture. Water quality monitoring can be conducted by deploying devices at different depths under the water. The extreme conditions require to deal with specific challenges, such as power consumption, environment pollution, communication obstacles, etc.

The external influences and the computation stress, the IoT devices may need to endure, can significantly contribute to their failures. Furthermore, the constrained resources can only foster different types of breakdowns. For example, reaching the memory limits can lead to failures due to memory exhaustion [RGPG19], while obstacles in wireless communication paths can lead to communication failures [SYW+19]. Often, the corresponding IoT systems have safety-critical applications and require solutions to prevent those types of failures. *Predictive Maintenance (PdM)* [Mob02] is a maintenance strategy that allows to schedule maintenance plans for different types of devices based on previous failure predictions. The strategy typically applies *Machine Learning (ML)* [MRT18] methods that use the data collected from the devices to learn and make predictions about potential failures. Various ML algorithms, such as *Support Vector Machines (SVMs)*, *Random Forests (RFs)* or *Neural Networks (NNs)*, also called *Artificial Neural Networks (ANNs)*, are widely used as a mean to enable PdM [ÇAZ+20; CSV+19]. IoT environments typically contain many devices with distinctive behavior. Even devices of the same type may be exposed to different external influences resulting in different failure patterns. Individual ML algorithms can be trained to recognize these specific failure patterns. Furthermore, if precision is a critical requirement, multiple algorithms can be trained on the same data aggregating their

results. This altogether can result in plenty of models, which leads to the requirement of a model management platform for failure prediction. There are usually IoT platforms that manage IoT environments by monitoring the existing IoT devices and collecting the corresponding data. Such a platform, described in this thesis, is the *Multi-Purpose Binding and Provisioning Platform (MBP)* [FHS+20; HBS+16]. Apart from the many functionalities provided by the MBP, it can also be used as a data source to feed the ML algorithms.

The goal of the thesis is to evaluate different ML algorithms on their failure prediction performance. The selected algorithms should be able to predict failures of IoT devices by learning on historical IoT data. A suitable metric and dataset should be used for the evaluation. Because of the many limitations of the real-world IoT data, the choice was made to generate synthetic datasets to test the algorithms. Additional focus of the thesis is to present a reusable solution for the failure prediction of IoT devices based on machine learning. This is realized by presenting a concept and a prototypical implementation of a platform called *Failure Prediction Platform (FPP)*. The platform represents a hosting environment for different ML models than can be integrated as uniform modules.

Structure

The remainder of the thesis is organized as follows:

Chapter 2 - Background: The fundamental terms and concepts are explained, including IoT, ML and PdM. Furthermore, the IoT platform MBP is described in the IoT section.

Chapter 3 - Related Work: A research on different failure prediction methods is conducted. The first section describes the methods used to predict failures of IoT devices. Additionally, ML-based methods for general IT and industrial devices are described.

Chapter 4 - Concept: The concept of the FPP is presented. The chapter describes the general architecture and the operation of the platform.

Chapter 5 - Algorithms: This chapter focuses on the ML algorithms. First, the data generation process for the synthetic datasets is described, followed by a brief explanation of the data preparation process. Furthermore, the selected algorithms are briefly described and the selection process is reasoned. Then, the implementation of the ML algorithms is clarified. Before the presentation of the results, the choice for the evaluation metrics is explained. The evaluation results are presented comparing the algorithms on several criteria. Finally, a special attention is given to the prediction strategies that provide sufficient time for maintenance.

Chapter 6 - Implementation: A prototypical implementation of the FPP is presented. The chapter includes the general implementation of the platform and a brief explanation of model integration by using an ML algorithm as example. Furthermore, an illustration of the training and prediction processes is given.

Chapter 7 - Conclusion and Outlook: The last chapter summarizes the results of the thesis and provides an outlook for future works.

2 Background

This chapter presents the background information that promotes the understanding of the following chapters. In the first place, a general and brief description of the Internet of Things and the Multi-Purpose Binding and Provisioning Platform is provided. In the following section, the most important terms and classifications of machine learning are presented. Finally, different maintenance strategies are compared, while the predictive maintenance is explained in greater detail.

2.1 Internet of Things

The Internet of Things describes a network of everyday objects that interact with each other to reach common goals [VF13]. The objects typically refer to IoT devices equipped with sensors and actuators, that can sense and act in the environment they inhabit. IoT has many applications in everyday life, forming IoT environments, such as *Smart Homes*, *Smart Factories* or *Smart Cities*. In a Smart Home [RT17], the appliances, devices and objects are connected and automated through IoT to improve the general quality of life. The Smart Factory [CWS+18] facilitates intelligent manufacturing and production optimization by employing *Industrial IoT (IIoT)*. Finally, a Smart City [SLF11] arises by embedding IoT equipment in city infrastructure, such as power grids, water systems, railways, buildings or roads, and connecting them over the Internet.

Because of their heterogeneity, the IoT devices produce data in different forms, volumes and intervals. Thus, the data needs to be managed. IoT platforms are implemented to manage the complex IoT environments by connecting the devices with the users and providing various processing capabilities [MMST16]. They vary in implementation, data management, but also in tasks they are trying to perform. However, they should be able to deal with the heterogeneity of the IoT devices, big data, privacy and security, and other IoT-related concerns. Guth et al. [GBF+16] describe a reference architecture for IoT platforms, displayed in Figure 2.1. It consists of several components. The devices use drivers to access sensors and actuators. They are connected to the integration middleware directly, or over a gateway in case of technical limitations. The IoT integration middleware is responsible for processing the received sensor data and sending commands to the actuators, while interacting with the application.

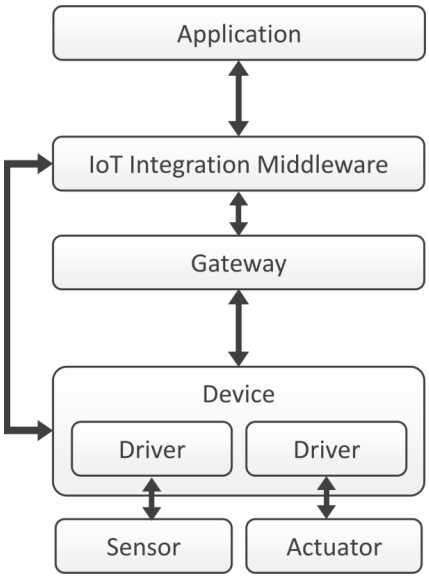


Figure 2.1: IoT platform reference architecture [GBF+16]

An example of an IoT platform is the The Multi-Purpose Binding and Provisioning Platform, that provides a range of capabilities in managing IoT environments [FHS+20; HBS+16]. A graphical tool provided by the platform can be used to model the IoT environments. Extract and control operators, implemented as scripts, can be automatically deployed on IoT devices to extract sensor values and control the actuators. The MBP provides a dashboard to monitor the IoT environment by showing status and statistical information, and also visualizing live and historical sensor values. Furthermore, rules for the environments can be created in form of events, conditions and actions. The users interact with the platform over the UI shown in Figure 2.2.

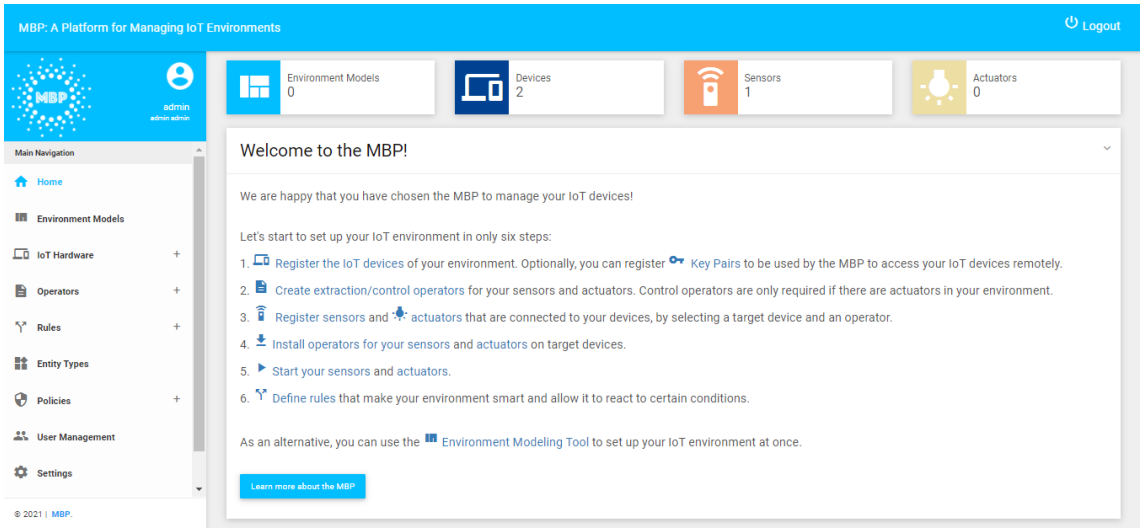


Figure 2.2: The MBP UI

MBP uses a Mosquitto[1] broker to connect the IoT devices. The sensor and monitoring data is sent from the devices to the MBP through the MQTT[2] protocol. The data sent over the broker is saved in a MongoDB[3] database. Maintaining the history enables the users to observe the behavior of the IoT environments through time. Moreover, the MBP with its straightforward implementation and history preservation serves as an exemplary IoT platform to be used for ML algorithm training and failure prediction of IoT devices. The MBP is available as an open-source platform on GitHub[4].

2.2 Machine Learning

Machine learning can be defined as the ability of a machine, algorithm or program to improve performance or make predictions by learning on historical data [MRT18]. The quality of ML algorithms often depends on the quality and size of the data used for learning. An ML algorithm learns on historical data and becomes an ML model used for prediction. In *online learning*, the learning and prediction phases occur simultaneously in a continuous process with a dynamic dataset, while in *offline learning*, there is a clear separation between the phases and the dataset is usually static [MRT18; SB14]. Apart from the differentiation between online and offline learning, there is also a distinction between *lazy* and *eager learning*. Lazy learners delay the processing of data until a prediction request arrives, in contrast to eager learners, which previously process the data creating a sort of rules used for predictions [Aha13].

Machine learning is considered a branch of *Artificial Intelligence (AI)* that uses computers to complement human intelligence rather than to simulate intelligent behavior, as it is the case in traditional AI [SB14]. There are many different classifications of ML algorithms [AKYC20; MRT18; NG20; SB14; SBB+20], often resulting in three main categories:

- *Supervised learning:* The training data is labeled, so the output of the algorithm is known. The algorithm maps the input to a pre-defined label. It is most commonly used for classification and regression tasks.
- *Unsupervised learning:* The training data is not labeled, so the output of the algorithm is not known. The algorithm tries to find patterns or common points in the input data. It is most commonly used for clustering tasks.
- *Reinforcement learning:* No training data in traditional sense, training and testing phases occur at the same time. The algorithm learns through the interaction with the environment by receiving feedback in form of rewards.

[1] https://mosquitto.org/
[2] https://mqtt.org/
[3] https://www.mongodb.com/
[4] https://github.com/IPVS-AS/MBP

The categories are illustrated in Figure 2.3. The figure also contains some of the most used algorithms in practice for each category. Additionally to the presented categories, there is a subcategory of machine learning called *Deep Learning (DL)* which is based on NNs [Nie15]. In contrast to the traditional NNs, *Deep Neural Networks (DNNs)* contain multiple layers enabling them to solve complex problems.

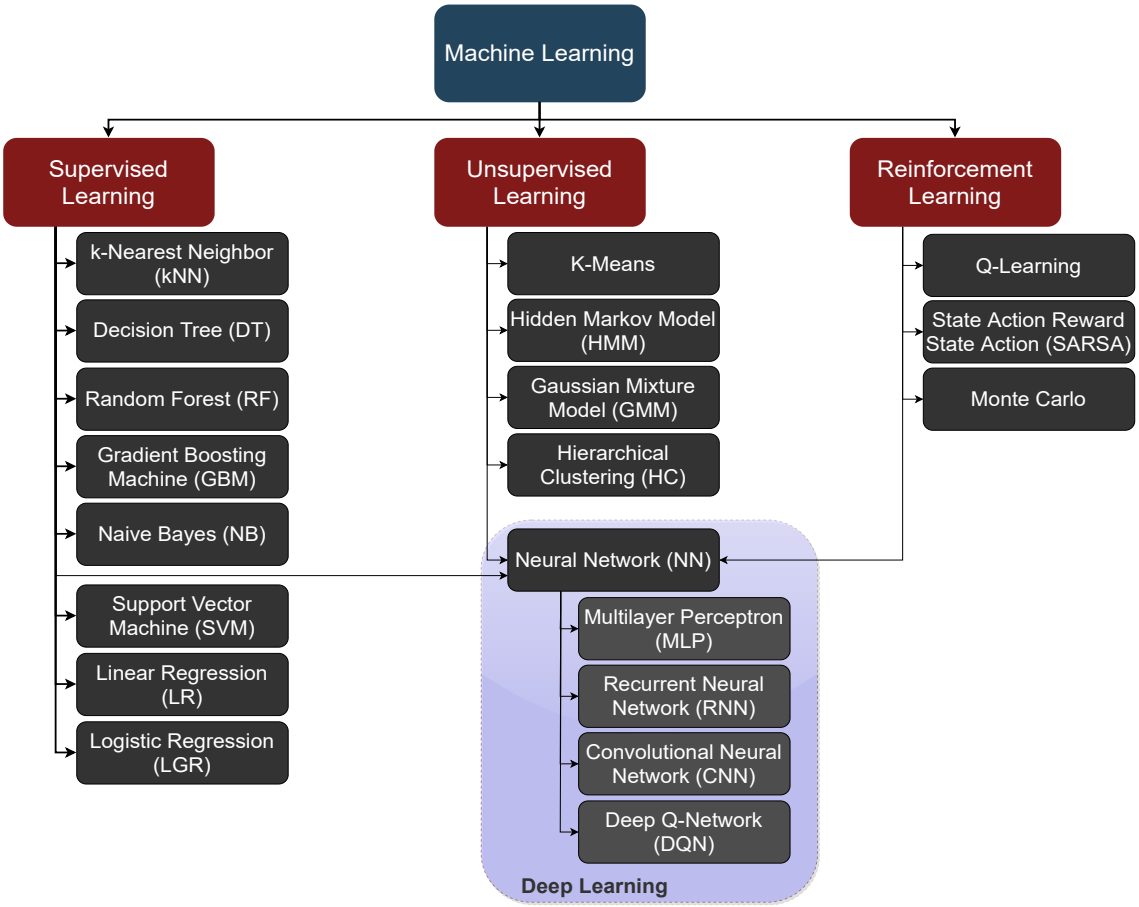


Figure 2.3: Classification of ML algorithms

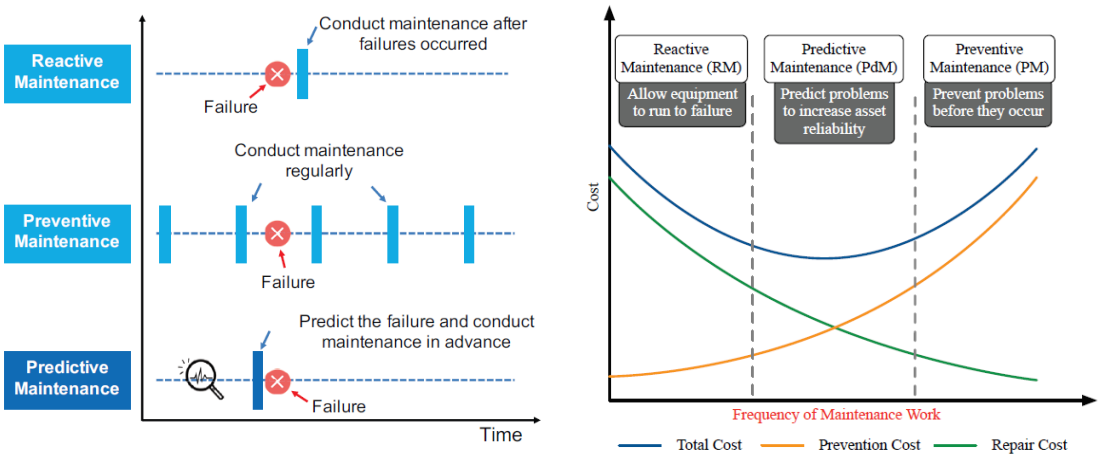
The ML algorithms can be applied in different areas to solve different problems. In recent years, machine learning is often used to predict the condition of devices, machines, equipment or processes to schedule maintenance and save failure costs. In the following section different maintenance strategies are described and the role of machine learning in maintenance is presented.

2.3 Predictive Maintenance

Maintenance ensures that a particular device remains in an optimal working condition. Different costs are associated with the maintenance depending on the used strategy. There are diverse classifications of maintenance strategies, usually resulting in three different strategies [Mob02; RZL+19]:

- *Reactive Maintenance (RM):* Maintenance is performed only after the device has failed. It is also known as corrective maintenance or run-to-failure. There are no maintenance costs at the time the device is working, but there are potentially high costs for repair or replacement. This typically results in high downtime of the device after a failure occurs.
- *Preventive Maintenance (PM):* Maintenance is performed in regular intervals even if the device is in an optimal working condition. The interval is selected based on different criteria. It is also called planned, scheduled or time-based maintenance. It typically results in a number of costly and unnecessary maintenance actions, but it may prevent failures and the costs associated with them.
- *Predictive Maintenance (PdM):* Maintenance is performed at specific occasions based on failure predictions. The predictions are typically based on the information gathered by monitoring the condition of the device. PdM maximizes the time between maintenance schedules and minimizes the costs. However, the costs for condition monitoring and failure prediction should not be neglected.

Figure 2.4a gives an overview of the different maintenance strategies by showing the time the maintenance is performed. Figure 2.4b displays the associated costs for each maintenance strategy. RM has high repair costs and low preventive costs, PM the other way around. PdM is a balance between these two strategies, resulting in lower overall costs.



(a) Maintenance plans

(b) Maintenance costs

Figure 2.4: Comparison of RM, PM and PdM [RZL+19]

As a common approach, different ML algorithms are used to predict failures for PdM [RZL+19]. To be able to perform the maintenance and prevent the failure, the predictions should provide enough time. The time between the failure prediction and the occurrence of the failure is referred to as *lead time* [RGPG19; ZKT+17]. The longer the lead time, i.e., the further the prediction in the future, the more time for maintenance is available. There are different failure prediction approaches that can provide sufficient time for maintenance. Classification algorithms can be used to predict whether a device will fail within a given time window [RGPG19]. Regression algorithms can be used to predict the *Remaining Useful Life (RUL)* of a specific device [MTS+17]. Anomaly detection indirectly provides lead time for maintenance [ZKT+17]. Furthermore, time series prediction can be used to predict sequences that provide information about failures in the future [HHX+20].

3 Related Work

Diverse ML techniques are applied to predict failures of a variety of devices. The following sections describe the techniques used in current research to predict failures of IoT, IT and industrial devices. They represent techniques that can be integrated in the extensible platform presented in this thesis, either in original or customized form.

3.1 Failure Prediction of IoT Devices

Many papers describe techniques for failure prediction focusing on specific IoT devices or specific causes. Some of the techniques presented in this section apply machine learning algorithms while others use simple statistical methods.

Rafiuzzaman et al. [RGPG19] describe a failure prediction technique where memory exhaustion is the cause for the failures of IoT devices. General-purpose applications running on IoT devices are typically not optimized for their constraints and often result in failures due to memory exhaustion. The presented technique uses *k-Nearest Neighbor (kNN)* algorithm to make predictions with different lead times. As part of the solution, Rafiuzzaman et al. created a concept called *MARK (Monitor and Analyze Resource Keys)* where the data is collected from the devices, processed, and then provided to the kNN-based prediction model. Because of the heterogeneity of the IoT devices, a similar pre-processing component will be used in the platform that will be introduced in this thesis.

Soualhia et al. [SFK19] present a framework for fault detection and prediction in edge cloud environments by using machine learning and statistical techniques. The focus of the paper is to evaluate the framework on HP[1] servers by detecting and predicting recurrent and accumulative faults (CPU, HDD, memory and network faults). They also performed the evaluation on Raspberry Pis[2] with CPU fault prediction and memory stress prediction by using *Convolutional Neural Network (CNN)*, *Long Short-Term Memory (LSTM)*, a type of *Recurrent Neural Networks (RNNs)*, and their combinations. Based on the results, the framework is suitable to detect and predict different faults in reasonable time. The framework also contains a pre-processing component, which the authors indicate as a crucial component that, amongst others, reduces the costs and improves the performance.

[1] https://www.hp.com/
[2] https://www.raspberrypi.org/

Wireless communication of IoT devices can also be a cause for failures. Suga et al. [SYW+19] describe a method to calculate the *QoS (Quality of Service)* outage probability for wireless communication. The authors indicate that ML-based predictions of the QoS values can not be accurate and sufficient if the communication failures are rare in the training process, which is typically the case. The described method uses QoS values predicted with *Probabilistic Neural Network (PNN)*, error distribution of different rarity classes, and the required QoS as a threshold to estimate the outage probability. Suga et al. achieve a high prediction accuracy after evaluating the method in an automobile factory scenario, using different wireless station nodes and a simulated WLAN throughput.

Ara et al. [AMB20] use *ARIMA (Auto-Regressive Integrated Moving Average)* to predict failures in wireless sensor nodes. ARIMA uses recognized patterns in the collected time series data to predict future faults. The authors evaluate the model by polluting a sensory dataset, simulating faults and creating specific patterns. Furthermore, the data goes through different pre-processing steps to achieve optimal results. The method is shown to be more useful for short-term predictions rather than for long-term ones.

Lian et al. [LCH19] describe an ageing monitoring and lifetime prediction system for IoT devices. The ageing of *Integrated Circuit (IC)* technology in IoT devices may be specific for each device or it may depend on the environment. The system presented by Lian et al. collects information from each device continuously, and then stores and processes the data in the cloud creating an IC-specific ageing model. Based on the ageing model, the lifetime of the particular device can be predicted. Another benefit of the cloud, stated by the authors, is the central collection of ageing information for all ICs of the same type, enabling simple identification of abnormalities.

A failure of a device can be predicted by estimating the battery lifetime of that device. There has been a lot of work on battery lifetime prediction in recent years, mainly targeting power optimization in smartphones. Li et al. [LLM18] use *Linear Regression (LR)* and tree-based ML models to predict the battery lifetime of smartphones. The models are trained and tested on a real-world dataset consisting of system, sensor and usage data. The proposed method can also be applied for other smart devices that produce similar data. The authors came to the result that tree-based models perform better than LR and that the battery discharge history is the most important feature for the prediction. LR is also used by Zhao et al. [ZGFC11] to estimate the battery discharge rate based on system data. Besides the ML techniques, there are many contributions that use statistical methods for battery lifetime prediction. Fafoutis et al. [FEV+18] calculate the battery lifetime of IoT devices using previous measurements of current consumption. The observations are conducted using environmental and wearable sensors with CC2650[3] MCU. Pandey et al. [PVK19] developed a framework for power optimization and battery lifetime prediction focusing on smartphones. The main data source used for prediction is the battery discharge pattern of individual user. A similar approach for battery lifetime prediction of mobile devices by using usage patterns is described by Kang et al. [KSH11].

[3] https://www.ti.com/product/CC2650

3.2 Failure Prediction of IT Devices

Current research is mainly focused on predictive maintenance and failure prediction of general IT devices rather than resource-constrained IoT devices. This section presents different works that use machine learning to predict failures of computers, servers, networks, hardware, software and other IT-related equipment. As the IoT devices are resource-constrained IT devices, the described methods can typically be adapted and used for IoT devices as well.

Storage disks, whether they are HDDs, SSDs or any other types, can be the cause for failures of the devices they inhabit. Different ML algorithms are used to predict and prevent these failures [AXDS19; CPV+17; Hua17; LLP+20; PHG13; PYAS20]. There is no consistent conclusion on which of the algorithms is best suited for this purpose, they are often affected by the type and model of the disk, usage environment, size and quality of data, and other factors. As a consequence, there are different results for the algorithm with the best performance: RF [AXDS19; PYAS20], *Decision Tree (DT)* [CPV+17], *Extreme Gradient Boosting (XGB)* [Hua17], CNN-LSTM [LLP+20], etc. Although the main research is focused on HDDs and SSDs, Ma et al. [MWL+20] present a failure prediction scheme for more generic NAND flash-based storage devices by using RF, SVM and *Multi-Layer Perceptron (MLP)*. NAND flash memory can also be found in IoT devices.

Memory can also be the cause for failures of IT devices. Du and Li [DL18] describe an online learning method based on a kernel function to predict *DRAM (Dynamic Random Access Memory)* failures. The method is based on historical error information to recognize patterns. DRAM failures can also be predicted by using RF with logs and sensor data, as presented by Giurgiu et al. [GSWB17]. In another paper, the authors describe the same technique for *DIMM (Dual In-line Memory Module)* failures [GWB17]. Sun et al. [SCH+19] propose a CNN model for failure predictions of disks and memories using time series data. In this case, CNN is shown to provide better results than RF and LSTM.

Network failures make the required device unreachable and can result in data loss. Wang et al. [WZW+17] use SVM and *Double Exponential Smoothing (DES)* to predict equipment failures in optical networks. Switch failures can be predicted by using RF to detect patterns in current and historical data [ZLM+18]. The same ML algorithm is used by Duenas et al. [DNP+18] to predict future network events based on a repository of previously collected network management events. DL techniques also have an application in network failure predictions, including CNN [JDC+18], LSTM [KC17; ZW19; ZWZ+19], and *Generative Adversarial Network (GAN)* [ZZY+20].

A more device-independent approach to predict failures is to aim the attention on the software, although the techniques provided by many authors depend on specific system metrics. Software aging can be one of the causes for failures, and machine learning can be used for prediction [AS08; ATBG10]. Pellegrini et al. [PSA15] developed a framework where multiple ML models can be built with the goal to predict the failure time of applications. Pitakrat et al. [PHG13] consider failures in the overall software

system architecture rather than in individual components by applying *Bayesian Networks (BNs)*. Furthermore, Campos et al. [CVC18] provide a performance comparison of different failure prediction algorithms by using the same dataset, and indicate the importance of data pre-processing for the performance. Moreover, the authors also study ensemble methods for software failure prediction [CCV19]. There are similar algorithm comparisons for software reliability prediction [JM18; KS12]. Apart from software in generic terms, ML-based prediction methods can also be used for database failures [KÖUG17; ZCW+18].

In addition to the research on individual software or hardware components, there are diverse papers focusing on servers or systems as a whole. Jauk et al. [JYS19] conducted a survey on prediction techniques for failures with different causes in high performance computing systems. Irrera et al. [IVD15] describe a failure prediction framework with self-adapting ability based on SVM. Sonoda et al. [SWM12] propose a method for predictions with sufficient lead time based on *Bayesian learning*. Predicting the workload of servers could be useful to prevent failures caused by overloading. Cao et al. [CYM+18] concluded that RF provides best results to predict server loads using monitoring data.

3.3 Failure Prediction of Industrial Devices

In addition to the resource-constrained IoT and general IT devices, industrial equipment, machines and processes have received much attention in recent years [KWH13; WSCL13]. IoT is typically used to monitor the equipment by attaching multiple sensors for data acquisition. The data is then collected in a central place where ML algorithms are applied to make failure predictions. This section summarizes the different ML techniques that are used in this research field.

Neural networks are often used for failure prediction. ANN can be used to predict motor failures using vibration data [SVSA19]. CNN can be used to predict power curves of solar panels to observe potential abnormal behavior [HJ18]. Predicting current engine condition with LSTM enables to generate repair alerts before the failure occurs [AG17]. Combinations of neural networks can be used as well, such as *Deep Belief Networks (DBNs)* for learning and *Feedforward Neural Networks (FNNs)* for predicting the RUL of rotating equipment [DH18]. There is a similar approach with *Autoencoder (AE)* and *Logistic Regression (LGR)* for aircraft engines [MSZL18]. Apart from NNs and DL techniques, other ML techniques also have a frequent application. DT can be applied to predict equipment failures in anode production [KVIT18]. RF can be used to predict different states of cutting machines [PRF+18]. *Gradient Boosting Machine (GBM)*, an ML technique for regression and classification problems, can be used for fault diagnostics and prognostics of vending machines [XHL18]. In aerospace industry, life consumption of jet engine turbine blades can be predicted with different *Regression (R)* techniques [Kar20]. Predictive maintenance of nuclear power plants can be achieved using SVM [GUL+20]. Table 3.1 summarizes the ML applications in predictive maintenance collected during literature surveys by Çınar et al. [ÇAZ+20], Carvalho et al. [CSV+19] and Ran et

al. [RZL+19]. In contrast to the original survey lists, which are much larger, the table only shows the different ML techniques that are used, represented with an application example. The table additionally reveals the equipment/system, the data and the device used for data acquisition for each example. The list of the ML techniques is not final. Furthermore, there are many contributions implementing and evaluating different combinations of the techniques to improve the overall results. Apart from that, there are different statistical methods and other approaches which will not be mentioned here.

ML Technique	**Equipment/ System**	**Data Acquisition Device**	**Data Description**	**Reference**
ANN	Motor	Accelerometer	Vibration	[SVSA19]
CNN	Photovoltaic panels	Wireless sensor	Electric power signals	[HJ18]
LSTM	Engine	Operational and sensors	NASA engine degradation simulation	[AG17]
DBN-FNN	Rotating equipment	Accelerometers	Vibration	[DH18]
AE-LGR	Aircraft engine	Multiple sensors	Engine degradation simulation	[MSZL18]
DT	Anode production of industrial equipment	Process sensor	Process sensor data	[KVIT18]
RF	Cutting machine	PLCs and communication protocols sensors	PLCs and communication data	[PRF+18]
GBM	Vending machine	-	Machine logs	[XHL18]
R	Jet engine blades	Flight sensor	Temperature, stress, strain	[Kar20]
SVM	Nuclear power plant	Temperature sensor, pressure sensor	Temperature, power, current, speed	[GUL+20]
kNN	Turbofan engine	Sensor run-to-failure measurements	NASA turbofan-engine dataset	[MTS+17]
k-Means	Oil-immersed power transformer	-	Dissolved gas concentrations data	[EACF17]
BN	Thermal treatment equipment	-	Semiconductor manufacturer	[ASZS15]

Table 3.1: ML for PdM (based on [ÇAZ+20; CSV+19; RZL+19])

4 Concept

As one of the main contributions of the thesis, this chapter presents a generic Failure Prediction Platform that can host multiple ML models. The first part of this chapter discusses the crucial components and their interactions, and the second part describes the general workflow and processes.

4.1 Architecture

The architecture of FPP and its environment are shown in Figure 4.1. The FPP assumes the existence of an *IoT platform* as illustrated in the top part of Figure 4.1. It specifically assumes the existence of two data sources, one for live and the other for historic and static data, resembling a *lambda-based architecture* [KMM+15]. The IoT platform receives the data from the *IoT environment*, which can be derived from sensors and actuators (*environment data*), or directly from the devices such as CPU and memory utilization (*monitoring data*). Additionally, the IoT platform stores general information about the devices (*device data*). The device data is obtained when the device is registered to the IoT environment, using for example a *device registration* component as described by Del Gaudio et al. [DRH20]. The process can be automated to some degree but it still requires a manual intervention. IoT platforms typically use a *message broker* that receives data from all the different devices in the IoT environment. It can be used as a source for the live data. Furthermore, there can be a *database* which permanently stores the received data from the environment and the static data of the devices. Usually, those two basic components are part of IoT platforms, as seen in the MBP. The MBP implements a Mosquitto broker and a MongoDB database. However, it is only important to provide those two data sources regardless of the types of components used in background. The IoT platform can explicitly provide an *API* component with access to its data, or a direct connection to the broker and database can be established.

The bottom part of Figure 4.1 shows the composition of FPP. The FPP should be able to build a connection to various kinds of IoT platforms and data sources. The *adapter* component decouples FPP from specific implementations and retrieves the data that will be used by the models. A message broker is used to coordinate and save the data permanently after the individual processing steps. After the data is retrieved by the adapter, it is sent to the broker and saved on a specific topic unchanged. In the next step, the data can be processed in the *data processing* component before it is used by the models. It can

be standardized using a pre-defined data template, enriched with missing information, transformed to a specific format, or adapted in any other way. Standardizing the data using a template leads to data consistency, which further eases the implementation of the models. The processed data is sent and saved again on a different topic, where it can be reused by the models. This component is required when dealing with heterogeneous data and conducts the processing needed by all the models. Assuming the data arrives as entries with timestamps, it can be used to extract meaningful information from the timestamp, such as hour of the day, weekday, holiday, etc. Another potential use case, in case of separated device topics, is to filter the data by devices and send it to the correct topic.

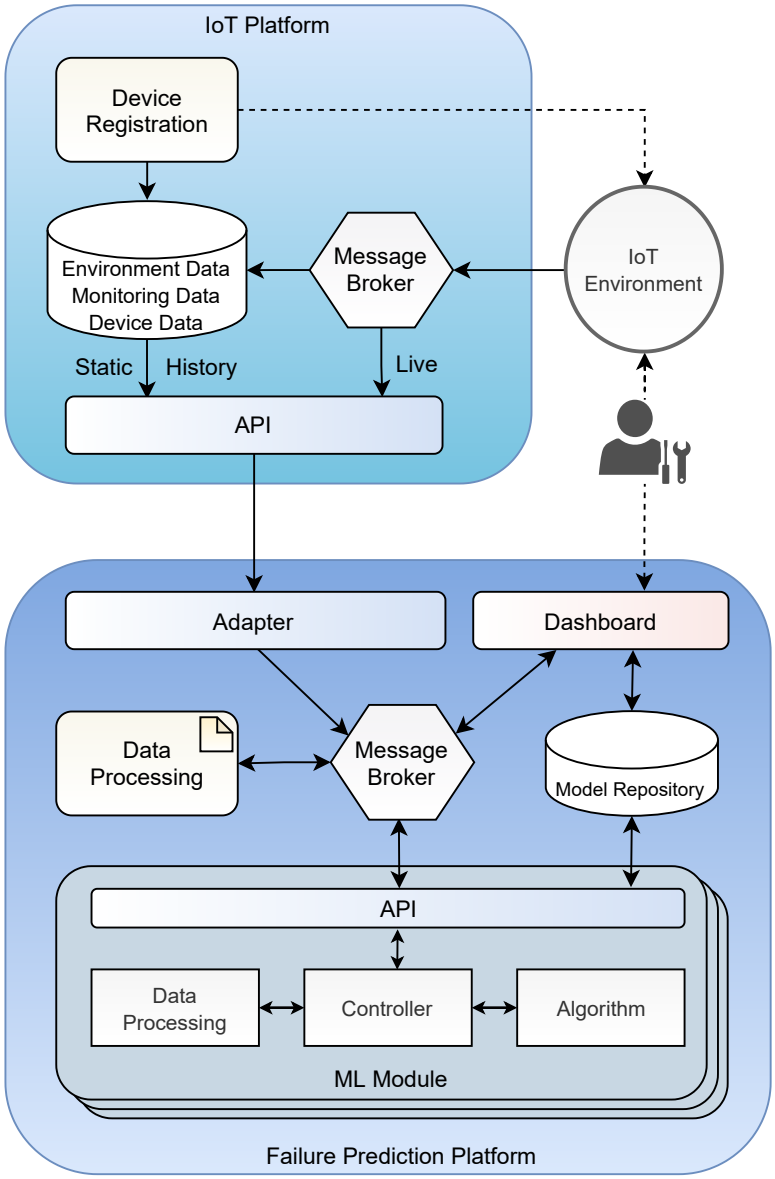


Figure 4.1: The FPP architecture. Solid lines stand for control and data flow, dotted lines for manual tasks.

Each ML model is integrated as a *module* consisting of four main components: API, data processing component, controller and the actual ML algorithm. The *API* has the same composition for all modules and consists of a message producer and two message consumers. There is a differentiation between a data consumer that consumes the previously processed data, and a commands consumer that consumes the commands sent from the dashboard. The message producer sends the results of the model and status or progress information to the message broker. The messaging interface decouples the ML modules from the platform. Although there is a generic data processing component in the previous step, different types of ML algorithms require the data to be in different formats. For example, tree-based algorithms can work very well with unchanged data in most of the cases, while LSTM usually needs the data to be normalized or standardized and put in a sequential format, as seen in the further course of the thesis. Therefore, an additional *data processing* component is integrated within each module that should prepare the data for the specific algorithm. The *algorithm* itself is the core of the module. It goes through a training process resulting in a model used for predictions. It can be implemented in various ways and with different libraries, but it should provide a training and prediction interface. Finally, all the components are orchestrated by the *controller* that has multiple purposes. It processes the commands received from the outside and performs the corresponding actions. It receives the data, forwards it to the processing component if necessary, and then to the algorithm. It can implement automated hyperparameter tuning to find the optimal parameters, i.e., the most optimal configuration of the algorithm based on the provided data. The controller also saves and loads data to the database or file system. It can implement a scheduler to periodically create backups, retrain the algorithm if online learning is not possible, or retune the parameters. It can estimate the current processing status and also send the predictions of the model with the message producer.

The *model repository* describes a general-purpose storage shared by all the models. The storage can be a database, a file system, or a combination. The main purpose is to store the description and the configuration of the algorithms, the hyperparemeters after tuning, or the complete ML models, i.e., internal parameters after learning. It can also store any other model-relevant information. It is accessed over the API components of the modules and over the dashboard.

After an ML model is ready and starts making predictions, the corresponding module sends the results to its predefined topic. A *dashboard* collects all the predictions from different models, processes them and visualizes the results. Besides the prediction topics, each module has a topic assigned for the commands sent from the dashboard. Furthermore, there can be additional topics used by the modules to send status and progress information. The dashboard should provide the possibility to interact with the modules and execute individual steps manually. The steps may include: importing history, starting automated hyperparameter tuning, training individual algorithms, starting and stopping individual model predictions, saving and loading models, etc. Furthermore, the dashboard accesses the model repository to retrieve and show the relevant model information in the UI. The dashboard can also be used to insert information during the model registration process. Finally, a maintainer monitors the dashboard and performs repairs or replacements in the

IoT environment. A notification system can also be implemented to inform the maintainer if some device is in the process of failing or has reached its critical thresholds based on the predictions.

4.2 Workflow

At the time FPP is connected to the IoT platform, the IoT platform could already have existed and collected some amount of data. The data can be used to initially train the algorithms. The FPP allows to get the old data from the database, go through the general processing step and store it on a specific topic. Afterwards, the training process of the algorithms can be started individually. The initial training is significant, because it allows a smooth start of the model predictions with meaningful results right at the beginning. But it can also be omitted, particularly if no data has been collected up to this point. This will result in training on the fly, either as online learning or retraining in specific intervals. Another crucial benefit of the old data is that it can be used to configure the scalers needed by some of the models. Scalers typically need to compute the internal parameters, such as min, max or mean, before they can be applied on incoming live data. Doing this configuration on the fly is a more cumbersome process that will probably lead to poor results in the beginning. Besides these benefits, there is a limitation that needs to be considered. Potentially, the huge amount of data that could be stored in the database of the IoT platform, could not be loaded in the memory at once. Nonetheless, in most of the cases, processing the data in batches or even selecting a portion of data could be enough to configure the scalers and also train the algorithms to provide meaningful results in the beginning.

In the second step, before any model predictions are started, FPP establishes a connection to the message broker in the IoT platform and starts receiving live data. Again, the data goes through the processing step and lands on a specific topic. At this point, model predictions can be started individually for each model. A module receives the data and processes it further if required. Then, the model makes a prediction and the controller sends it to its corresponding predictions topic. Finally, the predictions are retrieved and visualized on the dashboard. The data flowing through the system is always saved permanently on each topic, expect for the topics assigned for the commands and status or progress information.

5 Algorithms

This chapter presents the selection and comparison of ML algorithms, showing the suitability of the algorithms to be integrated in the FPP and to predict failures of IoT devices. Before the comparison, data used to train the algorithms and the data generation and preparation processes are described. The choice of the evaluation metric is elaborated and the evaluation results for each of the algorithms are presented. Finally, the prediction with lead time and the impact on maintenance schedules is discussed.

5.1 Data Generation

A problem of the real-world IoT data are the relatively rare device failures that could be used to train ML algorithms. Combined with the short time interval of data measurements (required for some metrics, such as CPU or memory utilization), this will result in a huge amount of data with a very small number of failures. For example, considering a 1 second interval for the duration of a year, the total number of data entries will be around 31.5 million. During this time, an IoT device may or may not fail at all if it is not put under a significant stress. This huge amount of data is not likely to fit in the memory, and solutions, such as online learning, would typically require a very long time to execute. Therefore, the choice was made to generate synthetic data that has the characteristics of real IoT data. These characteristics include:

- Relatively large number of entries generated with an interval of 1 second. However, the duration is restricted to two weeks, resulting in approximately 1.2 million entries. The sample allows to test the algorithms in an appropriate time interval.
- Small number of failures compared to overall data, resulting in data imbalance. The failures, however, were increased to hundreds or few thousands to be able to train the algorithms.
- Overlapping failures and non-failures. Data context preceding a failure may sometimes not lead to a failure. This is typically the reason why ML algorithms are used, otherwise the fixed thresholds can be easily estimated by using simple methods.
- Included timestamp, environment data, monitoring data and failures. This includes measures with discrete but also measures with continuous values. In practice, sensors and actuators provide data about the environment, while the devices communicate their internal state. An automated system or a maintainer can record the failures.

However, some simplifications were also included during the generation process that, generally, should not influence the comparison of the algorithms:

- Data is assumed to arrive from a single device. Although, the IoT environments consists of many devices, they can be treated individually.
- After a failure occurs, the maintenance of the device, i.e., the time in which the device does not send any data, is not included.
- There are no anomalies in the data. The focus is to compare the algorithms on regular data. Moreover, anomalies could be processed in the pre-processing component.
- Although short measuring intervals make sense for CPU or memory, for temperature or humidity it is exaggerated and can be extended to several seconds or minutes. This results in data with different intervals and, consequently, missing values after integration. This problem is solved by generating all values with a 1 second interval.

The first ten entries of the generated dataset are shown in Figure 5.1. Excluding *timestamps* and *failures*, there are 11 features divided in 3 categories: time, monitoring and environment features. Time features include *hour*, *weekday* and *running days*, which is the number of successive days without a failure. These features can usually be extracted from the timestamp during the pre-processing step. Besides hour and weekday, it is also possible to get further information from the timestamp, such as minutes, months or holidays. However, these two are sufficient to inject meaningful failures in this particular case. Monitoring features provide information about the device. They are *CPU* utilization, *memory* utilization, *storage* occupancy and *CPU temperature*. Finally, environment features represent the data retrieved from sensors or actuators. They are *temperature*, *humidity*, *air conditioner* and *heater*. The description of the data is shown in Figure 5.2. All values are generated as numeric values to avoid the encoding later on. This also includes weekdays, which start with zero (Monday) and end with six (Sunday). Storage occupancy, as well as CPU and memory utilization, are displayed in percentages. CPU temperature, environment temperature and humidity are decimal numbers with a digit after the decimal point. The temperatures are portrayed in degrees Celsius and humidity in percentage. Air conditioner and heater can have the values 0 meaning off or 1 meaning on. In case of failures, 1 stands for failure and 0 for no failure. The failure value in a data entry should be interpreted as the condition of the device after all the other measures were sent. In case of a failure, it is the last data sent by the device which can provide information why the failure occurred. Although the data is synthetic, an attempt was made to make it as real as possible. This can be inferred from the data statistics shown in Figure 5.2, such as mean, standard deviation, minimum and maximum. The data is analyzed with Python *pandas*[1].

The data was generated with a Java implementation that outputs a csv file. The generator uses a set of predefined rules for each feature and applies a degree of randomness. Each feature has a starting value appropriate to the time the generation process is started. The

[1] https://pandas.pydata.org/

values then change according to individual probabilities and intervals. Figure 5.3 illustrates the emerged patterns of the generated data, visualized with Python *matplotlib*[2]. CPU utilization is not clearly observable because of the very frequent and broad changes. Memory utilization, on the other hand, changes less frequently and remains in the range of 80%. The usage of the storage increases over time with a slow rate and random ups and downs. CPU temperature remains in a normal range with a low changing rate. There is no particular relation between the monitoring features. In contrast, the environment features depend on each other, mostly on the temperature value. The temperature depends on the time of the day, it starts increasing in the morning and decreasing in the afternoon. Additionally, it is limited by the air conditioner and heater. At 25 °C, the air conditioner is activated and remains active until the temperature drops to 23 °C. Similarly, the heater is activated at 15 °C and remains active until the temperature rises to 17 °C. This results in a clearly visible patterns displayed in Figure 5.3. Humidity depends only on the temperature value and has no random changes, resulting in a kind of inverted pattern.

The last two timelapses in Figure 5.3 show the failure patterns. A failure occurs during working days between 8 and 16 o'clock. Furthermore, CPU and memory utilization need to be above 92%, CPU temperature over 50.1 °C, and the environment temperature between 19 °C and 23.5 °C. Note that the temperature never exceeds 25 °C because of the air conditioner. Setting the limit to 23.5 °C covers both cases, failures for which the air conditioner is off and failures for which it is on. Heater is always off. Humidity and storage have no direct influence on the failures. There is no reason for choosing this specific data condition for a failure, the algorithms should be able to detect patterns regardless of the type of the failure. However, the ranges were chosen to get an appropriate number of failures. To compare the selected ML algorithms, two different datasets were generated. In the first one, the failures occur 70% of the time when the specified data condition is met. In the second one, 30% of the time. The first one should show the performance of the algorithms when there are enough failures to learn (approximately 4000), the second one should show their performance on a low number of failures (approximately 500).

The relations between the features and failures can be observed in the data correlation map shown in Figure 5.4. The correlations were computed with *Pearson correlation coefficient* [Agg15] and visualized with Python *seaborn*[3]. By looking at the failure column or row, the greatest correlation to the failures have CPU and memory, followed by heater, temperature, humidity, weekday and running days. Overall, the correlation values correspond to the defined failure condition. The map also shows relations between individual features. The dependencies between environment features are clearly visible, compared to the monitoring and time features.

[2] https://matplotlib.org/
[3] https://seaborn.pydata.org/

	timestamp	hour	weekday	running_days	cpu	memory	storage	cpu_temperature	temperature	humidity	air_conditioner	heater	failure
0	10.05.2021 15.21.07	15	0	0	30	70	50	50.0	18.0	40.0	0	0	0
1	10.05.2021 15.21.08	15	0	0	27	70	50	50.0	18.0	40.0	0	0	0
2	10.05.2021 15.21.09	15	0	0	30	70	50	50.0	18.0	40.0	0	0	0
3	10.05.2021 15.21.10	15	0	0	34	70	50	50.0	18.0	40.0	0	0	0
4	10.05.2021 15.21.11	15	0	0	31	70	50	50.0	18.0	40.0	0	0	0
5	10.05.2021 15.21.12	15	0	0	35	70	50	50.0	18.0	40.0	0	0	0
6	10.05.2021 15.21.13	15	0	0	37	70	50	50.0	18.0	40.0	0	0	0
7	10.05.2021 15.21.14	15	0	0	40	70	50	50.0	18.0	40.0	0	0	0
8	10.05.2021 15.21.15	15	0	0	34	70	50	50.0	18.0	40.0	0	0	0
9	10.05.2021 15.21.16	15	0	0	31	70	50	50.0	18.0	40.0	0	0	0

Figure 5.1: Data sample

	hour	weekday	running_days	cpu	memory	storage	cpu_temperature	temperature	humidity	air_conditioner	heater	failure
count	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0	1209600.0
mean	11.5	3.0	0.4	52.1	85.8	84.6	56.1	19.5	36.9	0.2	0.3	0.0
std	6.9	2.0	0.7	29.0	14.7	9.7	2.8	3.5	7.0	0.4	0.4	0.1
min	0.0	0.0	0.0	1.0	20.0	48.0	49.7	14.9	26.0	0.0	0.0	0.0
25%	5.8	1.0	0.0	27.0	80.0	80.0	54.3	16.2	29.4	0.0	0.0	0.0
50%	11.5	3.0	0.0	52.0	90.0	87.0	56.1	19.0	38.0	0.0	0.0	0.0
75%	17.2	5.0	1.0	77.0	97.0	92.0	58.2	23.3	43.6	0.0	1.0	0.0
max	23.0	6.0	2.0	100.0	100.0	95.0	62.5	25.0	46.2	1.0	1.0	1.0

Figure 5.2: Data description

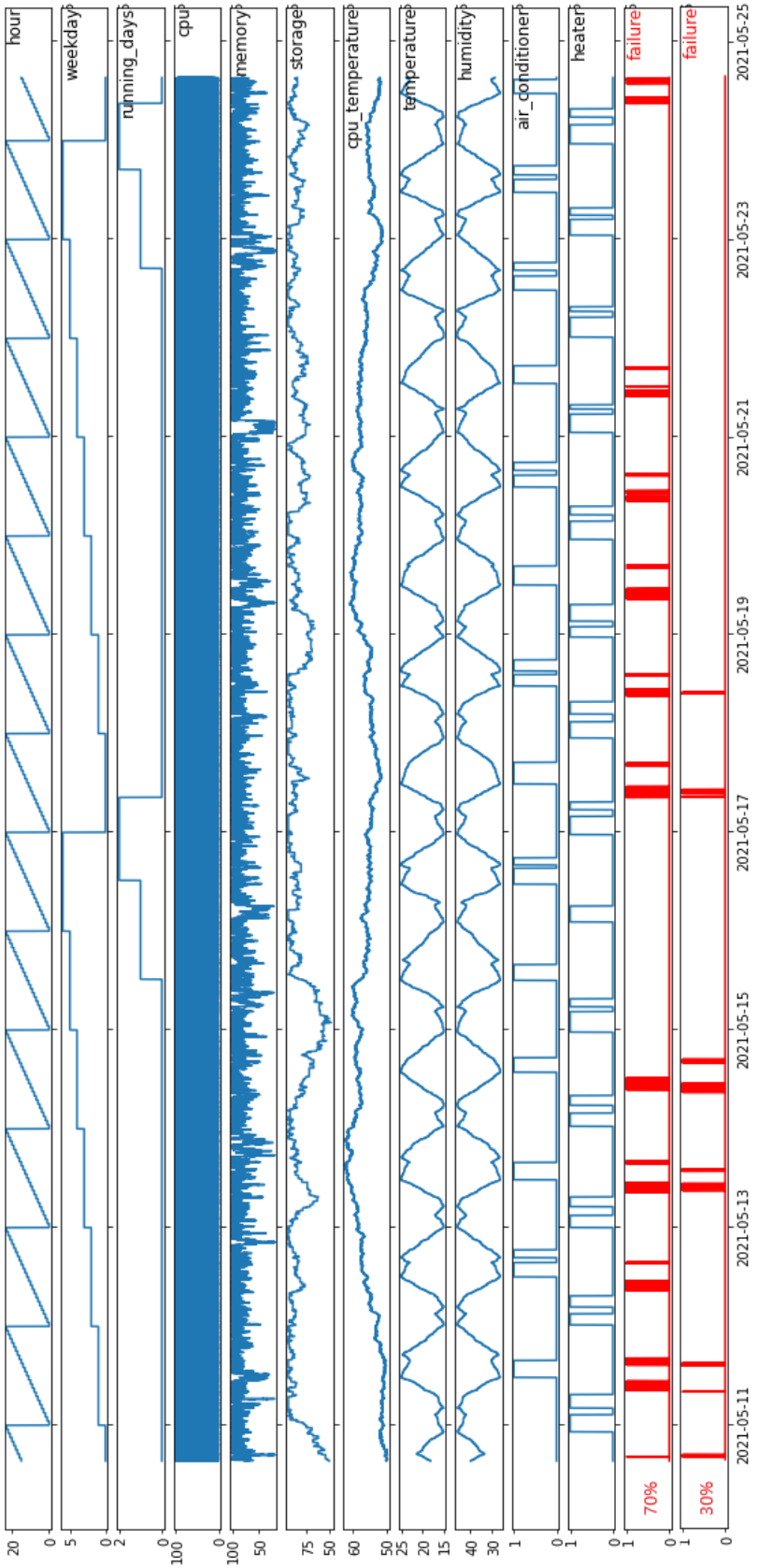


Figure 5.3: Data patterns

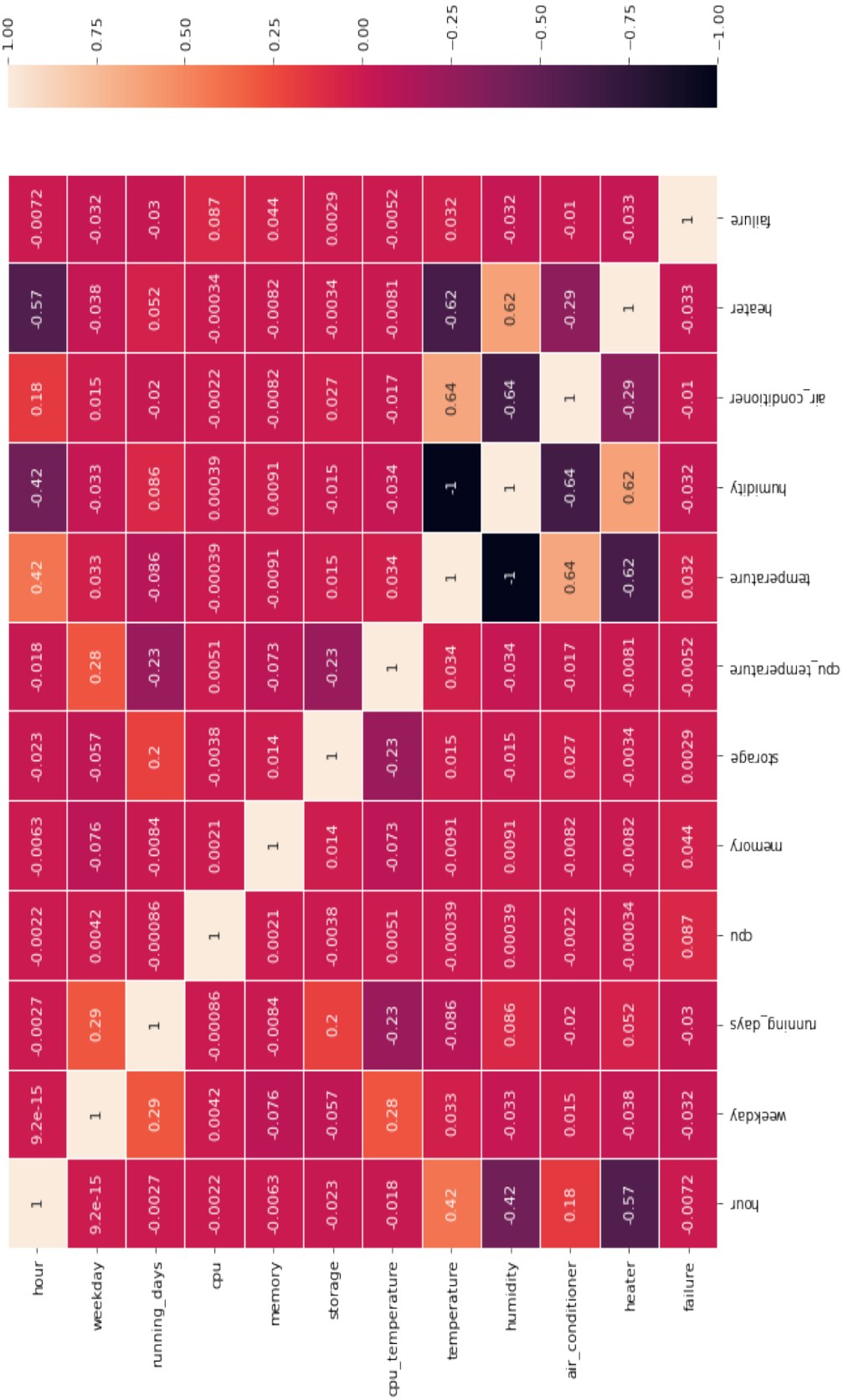


Figure 5.4: Data correlation

5.2 Data Pre-processing

The comparison of the ML algorithms is performed on four different data representations: original, balanced, normalized and standardized. *Original* data is unchanged from the representation in Figure 5.1, except for removing the timestamp values. *Balancing* is performed by under-sampling non-failures in the training data. Data entries without a failure are randomly selected and removed from the dataset, resulting in a similar number of failures and non-failures. Under-sampling may erase important information from the data, resulting in the algorithms to perform worse. The major benefit of data balancing is that it reduces the resources and time needed to train the algorithms. Data *normalization* in this thesis refers to the scaling of feature values in the range between zero and one. The scaling is performed with the Equation (5.1), separately for each feature by finding its minimum and maximum values. The data is also *standardized* by considering each feature individually. The feature values are subtracted by the mean and then divided by the standard deviation, as depicted in Equation (5.2). This results in values centered around zero with unit variance. An additional pre-processing step, executed before the data is normalized and standardized, is the transformation of the time features *weekday* and *hour*. To indicate that the difference between the hours 23 and 0 is the same as the difference between 22 and 23, the values are transformed in cyclic representations with the *Fourier Transform* [BB86]. In the transformation process, sine and cosine values are computed for each feature, as shown in the Equation (5.3). Hence, each feature is replaced by a corresponding sine and cosine feature. The computation depends on the total number of different feature values N, which is 24 for hours and 7 for weekdays.

$$x_{norm} = \frac{x - \min}{\max - \min} \tag{5.1}$$

$$x_{stand} = \frac{x - \mu}{\sigma} \tag{5.2}$$

$$x_{sin} = \sin\left(\frac{2\pi x}{N}\right) \qquad x_{cos} = \cos\left(\frac{2\pi x}{N}\right) \tag{5.3}$$

Some of the algorithms may suffer from issues with dimensionality and perform significantly slower compared to the other algorithms. The problem is solved by reducing the number of dimensions with *Principal Component Analysis (PCA)* [SA21]. The method is applied only for specific algorithms and specific data representations, depending on the training and prediction speed. Furthermore, some of the algorithms may expect a specific input format. The data is then reshaped accordingly.

5.3 Algorithm Selection

There are many ML algorithms that can be used to predict device failures. They can differ across multiple properties and features. They can be supervised or unsupervised, they can solve classification, regression or clustering tasks, or they can belong to different families, such as deep learning, tree-based, probabilistic or linear. The total number of algorithms considered for comparison was restricted because of the long configuration and training times, ensuring results of higher quality for the few selected algorithms. The selection of algorithms was performed based on several criteria:

- **Supervised learning**. Based on the assumption that the data, that will be used to train the algorithms, contains failure entries as well, supervised learning algorithms were selected.
- **Practical use**. As described in Chapter 3 within the related work, some of the algorithms are used more than the others for predictive maintenance.
- **Diversity**. Algorithms were selected to represent a variety of different families, tasks and features, such as online learning or prediction with lead time.

Based on these criteria, the choice was made on ten algorithms, nine of which are implemented as classification algorithms and one as time series regression algorithm. There are representatives of different families and types. All of the algorithms support online learning, either implicitly or through some variation. The selected algorithms are briefly described in the following sections, mainly based on the explanations provided by Charu Aggarwal [Agg15], Hastie et al. [HTF09] and Goodfellow et el. [GBC16].

5.3.1 k-Nearest Neighbor

k-Nearest Neighbor (kNN) is one of the simplest ML algorithms that can be used for both classification and regression tasks. In this thesis, it is used as a supervised classification algorithm. kNN for classification searches for k-nearest neighbors in the training dataset, typically by using the Euclidean distance, and then chooses the most frequent class of the neighbors. The main challenge is to appropriately choose the parameter k. Choosing k too small implies high sensitivity to outliers with limited generalization, while choosing k too large implies exaggerated generalization, losing sensitivity to data locality. The parameter is estimated by probing different values and selecting the value with the lowest classification error. kNN belongs to lazy learners without a training phase in traditional sense. However, it can prepare the data to enable efficient search during the prediction phase. It implicitly supports online learning because of the way its implemented. However, if there are any optimization steps, they should also support dynamic updates.

5.3.2 Decision Tree

Decision Tree (DT) is a simple algorithm that defines a hierarchical set of rules during the training process, which are used for classification or regression later on. It is organized as a tree with a root node, intermediate splitting nodes and leaf nodes. The root node is the starting point of the classification or the regression process. The splitting nodes use specific rules on feature values to split the data in two or more branches. The leaf nodes contain class labels in case of classification or values for regression. The construction of the tree in the training process may lead to perfect separation of class labels in the leaf nodes. However, this typically implies overfitting and bad performance on new data. To avoid overfitting, pruning mechanisms can be applied to change the structure of the tree. In its standard implementation, the tree is built by considering the entire dataset in defining splitting criteria. Reconstructing the tree for each new data entry is expensive, however, there exist variations that support online learning [JSC13].

5.3.3 Random Forest

Random Forest (RF) is a representative of the *bagging* algorithms that uses the majority vote or average result of a number of independently constructed trees. Since the trees are independent, they can be constructed in parallel. Each tree is constructed by applying a degree of randomness, specifically in selecting a subset of feature candidates for each split. Tuning the RF algorithm includes choosing the appropriate number of trees and number of features considered for splits. Typically, the greater the values, the better prediction performance of the algorithm, but also longer training time. RF is robust to outliers, irrelevant features, it reduces overfitting and typically yields better prediction results than a single DT. However, it usually requires more resources and training time. Another drawback compared to DT is the interpretability of the model. While a single tree can be easily visualized and the predictions confirmed by following the paths, this is hard or impossible to achieve for the RF. Random forest can also be adapted to support online learning as an incremental classifier [WWCL09].

5.3.4 Extreme Gradient Boosting

Extreme Gradient Boosting (XGB) is a representative of the *boosting* algorithms, where trees are constructed sequentially using the information from previous trees. It was initially developed by Tianqi Chen and described by Chen and Guestrin [CG16]. The authors provide an open-source library[4] with multiple functionalities and configuration possibilities. XGB is an improved version of the *Gradient Boosting Machine* [Fri01], aiming the attention on efficiency and scalability. As DT and RF, it can be used for both classification and

[4] https://github.com/dmlc/xgboost

regression. Apart from trees, the algorithm can also use linear functions as base learners. For comparison later on, XGB will be used for classification with tree as base learner. Among others, the library implicitly supports online learning, automatic feature selection and computation on a distributed system.

5.3.5 Naive Bayes

Naive Bayes (NB) is a probabilistic algorithm that makes use of the *Bayes theorem* to compute conditional probabilities of classes and features. The drawback of NB is that it assumes the independence of features, which is rarely true in real-world scenarios. The probability estimations depend on the occurrence of the feature values. The problem arises if a specific feature value never occurs in the training data for a specific class, resulting in zero probability all the time. To avoid this problem, different smoothing strategies are applied, such as the *Laplace smoothing* or the more general *Lidstone smoothing*. They increase the values consistently through all features. The implementation of NB depends on the type of the data, whether the feature values are discrete, binary, continuous, etc. This results in different types of NB, such as *Bernoulli*, *Multinomial*, *Categorical* or *Gaussian*. By storing and updating data statistics, it is easy to enable online learning.

5.3.6 Support Vector Machine

Support Vector Machine (SVM) makes use of hyperplanes for classification or regression. In case of classification, SVM builds hyperplanes in n-dimensional space maximizing the margins between different classes. They are then used as decision boundary for classification. In its standard implementation, it can only be used for binary classes, but different tricks can be applied for multi-class problems. It can also be applied on linearly inseparable data by employing the *kernel trick* and mapping the feature space in higher dimensions. Apart from the linear kernel, different other kernel functions can be used, such as polynomial, radial basis, sigmoid, or even a custom function can be defined for a specific use case. Bordes and Bottou [BB05] present the *HULLER*, a variation of the SVM created to support online learning. Their results show that the algorithm has the same accuracy but lower time and memory requirements compared to the state-of-the-art SVM algorithms.

5.3.7 Logistic Regression

Logistic Regression (LGR) is typically used to solve binary classification problems by computing the probabilities of the classes with the logistic function. For two classes, the algorithm computes probabilities between 0 and 1 ensuring they sum to one. The classification is performed by choosing a specific threshold and mapping the probabilities either to class 1 or class 0. As SVM, LGR belongs to linear classifiers that perform well with linearly separable data, but struggle otherwise and need the data to be transformed.

There are many variations of LGR built to solve different problems, including variations that support multiple classes and online learning. Paul and Ueno [PU20] propose *RILR (Robust Incremental Logistic Regression)*, a variation of LGR that learns incrementally. The evaluation of RILR shows that the algorithm requires shorter time to learn and less storage for the data.

5.3.8 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [Ket17; Zha04] is an iterative optimization method rather than an ML algorithm. It is used to train other ML algorithms, such as LGR, SVM or DNN. With the goal to minimize the loss function, SGD iteratively uses a subset of the training data to update the gradient of the loss function. Multiple runs over the training data can be performed until the algorithm converges. The alternative to the stochastic approach is the batch approach, that uses the entire dataset for the gradient update. Furthermore, SGD can use only a single entry from the dataset or a small subset referred to as mini-batch. In this thesis, SGD refers to a linear classifier with stochastic gradient descent learning. The classifier can be LGR or SVM depending on the optimal parameter configuration. It is included in the comparison because of its efficiency and implicit support for online learning.

5.3.9 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) is a feedforward neural network with an input layer, one or more hidden layers, and an output layer. The feedforward neural networks process the data from the input to the output without providing feedback. The type of algorithms that provide feedback are called recurrent neural networks. For classification, MLP tries to approximate a classification function by using the *backpropagation algorithm* for training. Tuning the MLP typically consists of finding the optimal architecture, activation functions, optimizer, learning rate, etc. Since MLP can use SGD for training, it implicitly supports online learning. Its layers and non-linear activation functions enable its application on linearly inseparable data. MLP can be used to solve complex problems, but typically requires long training time compared to other ML algorithms. Multiple hidden layers make it a representative of the DNNs, together with LSTM described in the following section.

5.3.10 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a recurrent neural network that implements memory cells, gates and loops. It can be built with multiple hidden layers, each with multiple memory cells. The information flow in the cells is controlled by a set of gating units, namely input, output and forget gates. They decide which information is relevant to keep in the internal cell state. LSTM allows incremental learning and it can be used for time

series predictions by learning and predicting entire data sequences. The corresponding problems that can be solved include speech recognition, handwriting recognition, machine translation, etc. Similar to MLP, LSTM can solve complex problems but typically requires longer training times. In this thesis, it is used to make a sequence of failure predictions up to a specific point in the future.

5.4 Algorithm Implementation

All algorithms are implemented in Python with the *scikit-learn*[5] [PVG+11] library, except for XGB, NB and LSTM, which use different libraries. Scikit-learn and its functions were also used for scaling and PCA, as well as for cross-validation and hyperparameter tuning described later on. XGB is implemented with *XGBoost*[6] [CG16] and LSTM with *Keras*[7] [Cho+15]. As there are discrete and continuous features in the generated dataset, a mixed implementation of NB is required. This is realized with Mixed Naive Bayes[8], that combines Categorical and Gaussian NB. Using separate predictions and combining the results is feasible because of the naive assumption of NB that the features are independent.

A major challenge, as typically the case in machine learning, is to find the optimal parameters of the algorithms for a particular dataset. The approach used in this thesis consists of an automated randomized search with cross-validation. This approach is used for all algorithms, except for NB and LSTM, which were easier to tune manually because of their different implementations. For the automated randomized search, a set of parameters needs to be defined for each algorithm, including their potential values or ranges that should be tested. The search is completed after a specific number of iterations, chosen individually for each algorithm as a trade-off between search time and result quality. In each iteration, a parameter configuration is randomly selected and tested with a 3-fold cross-validation. The result of the search is the best configuration for the specified number of iterations. The alternative of the randomized search is to test every parameter configuration from the predefined parameter space. This can result in a very large number of iterations and is, therefore, not suited for the algorithms in this thesis. During the tuning process, it is also important to let the algorithms know of the highly imbalanced dataset. This is typically done by setting the class weights over a specific parameter. Finally, training and prediction are performed with the 4-fold cross-validation, retuning the parameters for each split. The prediction performance is estimated by averaging the results.

The execution of kNN appeared to be significantly slower on the normalized and standardized data compared to the other algorithms. As kNN is especially sensitive to higher number of dimensions, the dimensions of normalized and standardized data were reduced from

[5] https://scikit-learn.org/
[6] https://github.com/dmlc/xgboost
[7] https://keras.io/
[8] https://github.com/remykarem/mixed-naive-bayes

13 to 10 using PCA. The resulting dataset improved the speed without reducing the prediction performance. Dimensionality reduction is only performed for kNN, the other algorithms handled the higher number of dimensions sufficiently well. Individual data pre-processing is also required by LSTM. Since LSTM works with sequences, it requires a three-dimensional input consisting of samples, time steps and features. The dataset was reshaped accordingly.

5.5 Evaluation Metrics

There are various evaluation metrics that are used to compare ML algorithms. The choice depends on several criteria, such as the category of the algorithms, number of classes, class distributions, etc. The majority of the ML algorithms in this thesis are binary classification algorithms on imbalanced data. Hence, the metrics are chosen appropriately. Some of the most used evaluation metrics for binary classification are shown in Table 5.1.

Metric	Formula	Description
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Ratio of correct predictions to overall predictions.
Precision	$\frac{TP}{TP+FP}$	Ratio of correct positive predictions to overall positive predictions.
Recall	$\frac{TP}{TP+FN}$	Ratio of correct positive predictions to total positives.
F1 Score	$2 \cdot \frac{P \cdot R}{P+R}$	Harmonic mean of precision and recall.
ROC AUC	$\frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$	Probability that a uniformly drawn random positive is ranked higher than a uniformly drawn random negative.

TP = True Positive; TN = True Negative; FP = False Positive; FN = False Negative; ROC = Receiver Operating Characteristic; AUC = Area Under the Curve.

Table 5.1: Evaluation metrics for classification (based on [FHM09; HM15; SL09])

Compared to the *F1 score*, *accuracy* and *ROC AUC* perform typically worse on imbalanced data [BDA13; SR15]. Moreover, the F1 score captures both *precision* and *recall* into a single measure. Therefore, it is used to compare the algorithms regarding their prediction performance in this thesis. It is worth to mention, that both accuracy and ROC AUC can be adapted for imbalanced data, resulting in more complex *balanced accuracy* and *weighted AUC* [BDA13]. Consequently, they become acceptable evaluation metrics as well. To treat both classes with equality, the *macro average* of the F1 score is used [NPK+16; OB21]. It

is estimated by computing the F1 scores for both classes and finding their unweighted mean. Macro F1 is also used as a metric for the hyperparameter tuning. The values range from 0 (worst result) to 1 (best result). The precision in predicting failures is measured as well. It is used to better understand the results, as it directly correlates to the failure probability in the generated datasets. Because macro F1 and precision are used for classification algorithms, the predictions of LSTM need to be mapped to either 0 or 1. LSTM is implemented as a network with three layers. The output layer uses *sigmoid activation function* [HTF09] and outputs values between 0 and 1, which are then mapped by using an optimized threshold.

5.6 Evaluation Results

The prediction performance of the algorithms presented in Section 5.3 is estimated by using the generated datasets and the metrics macro F1 and precision. The algorithms were initially trained and evaluated on the dataset with the 70% failure probability for the specified failure condition. The macro F1 scores for each algorithm, on four different data representations, are shown in Figure 5.5. By looking at the original data representation, it is clear that the tree-based algorithms DT, RF and XGB, provide the best results, followed by kNN and NB. The F1 score of 0.9 is considered a very good result. The results are worse for the NNs, MLP and LSTM, and linear classifiers SVM, LGR and SGD, which have the worst prediction performance. Generally, data balancing has no influence on tree-based algorithms, as they were able to provide a very high F1 score even on the significantly reduced data size. They were able to recognize the failure patterns by using only a small representation of non-failures. Linear classifiers had also similar results for balanced as for original data. Other algorithms performed generally worse on balanced data. A visible impact of the normalized and standardized data can be noticed at MLP and LSTM. LSTM has seen an improvement of 0.13 compared to the original data. Linear classifiers performed slightly better as well. The tree-based algorithms and kNN show no difference to the original data. However, NB has significantly worse results on normalized and standardized data. Figure 5.6 shows the precision for the same dataset. Since the failures are injected with the probability of 70%, a precision of 0.7 is considered a good precision. There are no significant changes in the performance order of the algorithms compared to the F1 score. However, MLP shows very good precision results, meaning that the lower F1 score can be explained with the low prediction performance on non-failures.

In the second dataset, the failures are injected with the probability of 30% for the specified failure condition. As the number of failures is very low, it is interesting to see how the algorithms behave if there is limited data to learn. The macro F1 scores are shown in Figure 5.7. Compared to the results of the previous dataset, there are some similarities but also noticeable changes. By looking first at the original data, the tree-based algorithms showed again the best results, followed by NB and kNN. A noticeable change is that the results of MLP and LSTM are on the level with the linear classifiers, which again have the worst prediction performance. The scores on the balanced data are either worse or similar

to the ones on the original data. This time, tree-based algorithms have lower scores on balanced data. The score for LSTM is zero, indicating that the algorithm was not able to use the very restricted amount of data to provide any correct predictions. The F1 scores for normalized and standardized data are similar to the scores in the previous dataset, except for the MLP, which shows no significant improvement compared to the original data. Finally, Figure 5.8 shows the corresponding precision values. The NNs, MLP and LSTM, have zero precision on the original data explaining the low F1 scores. But again, they show far better results on normalized and standardized data. MLP has even the best precision overall, with 0.32 on standardized data.

Apart from the prediction performance, the training and prediction time of the algorithms was measured. It is hard to compare the algorithms regarding their training and prediction speed, as they highly depend on the parameter configuration. For example, increasing the number of trees in RF to achieve better results will also increase the training time. The training time is not the same for RF with 10 and RF with 500 trees. Prediction time is more robust to these changes but still dependent. The approach used in this thesis, to be able to compare the algorithms to some degree, is to find the best configuration for a particular dataset. Regardless of how the parameter configuration looks like, it is used to measure the training and prediction time. The time was measured on the original data representation of the 70% dataset, by using 75%-25% train-test split without cross-validation. It is performed on a 2.60 GHz CPU and a 24 GB RAM, employing the parallelism provided by the underlying libraries. The results are illustrated in Figure 5.9. The left diagram shows the training time and the right diagram the prediction time of the algorithms. The slowest algorithms regarding the training time are MLP and LSTM, followed by LGR. The results for NNs are as expected, since they are typically slow to train because of their complexity [Agg15]. LSTM requires a training time of over 5 minutes for 907,200 entries. The fastest algorithms regarding training time are DT, NB and SVM, which require less than 3 seconds for 907,200 entries. In general, the prediction is much faster than training. However, because kNN is a lazy learner, its prediction takes longer than training for this particular data split. With the total time of over 10 seconds, it is the slowest algorithm compared to the others. The fastest algorithms are DT and the linear classifiers, with prediction time under 0.03 seconds. The prediction was performed on 302,400 entries.

The evaluation results presented in this section are summarized in Table 5.2 by highlighting the winners and losers for each evaluation criteria. The table shows only the highest values for macro F1 and precision across the different data representations. Additionally, the average for the two datasets is computed. The tree-based algorithms have the highest prediction performance, while the linear classifiers perform the worst. For these particular datasets, a simple DT is enough to get the optimal prediction results, and it also requires short training and prediction time. The overall results should be interpreted with caution, as they depend on the dataset and the parameter configuration of the algorithms. Providing more time for parameter tuning and data for training, the results of the other algorithms can also be improved. The evaluation results should provide a brief comparison of different types of ML algorithms, focusing on the failure prediction performance on the IoT data, and the training and prediction speed.

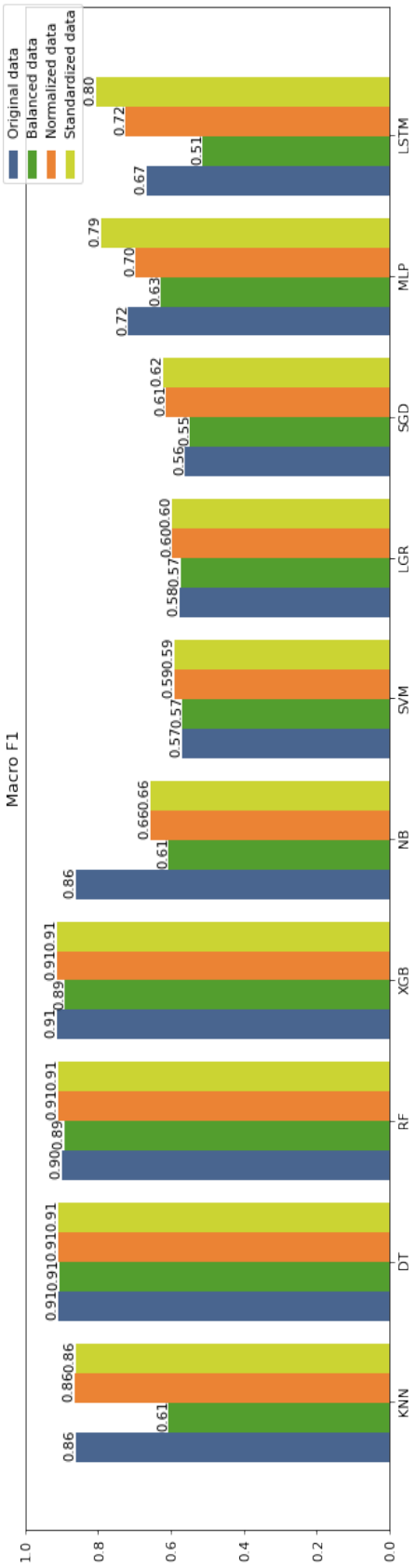


Figure 5.5: Macro F1 score with 70% failures

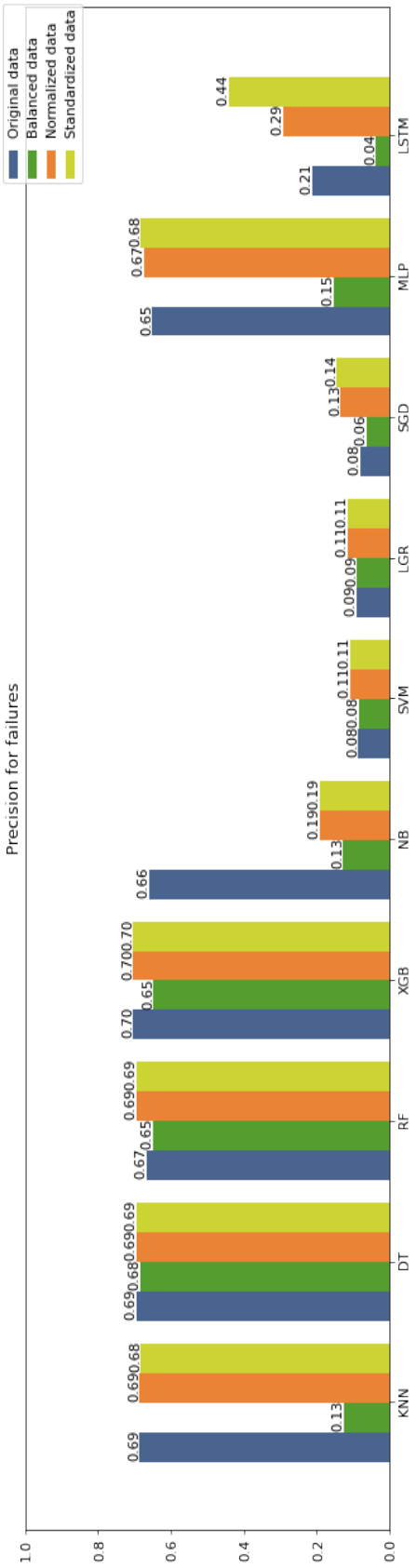


Figure 5.6: Precision score with 70% failures

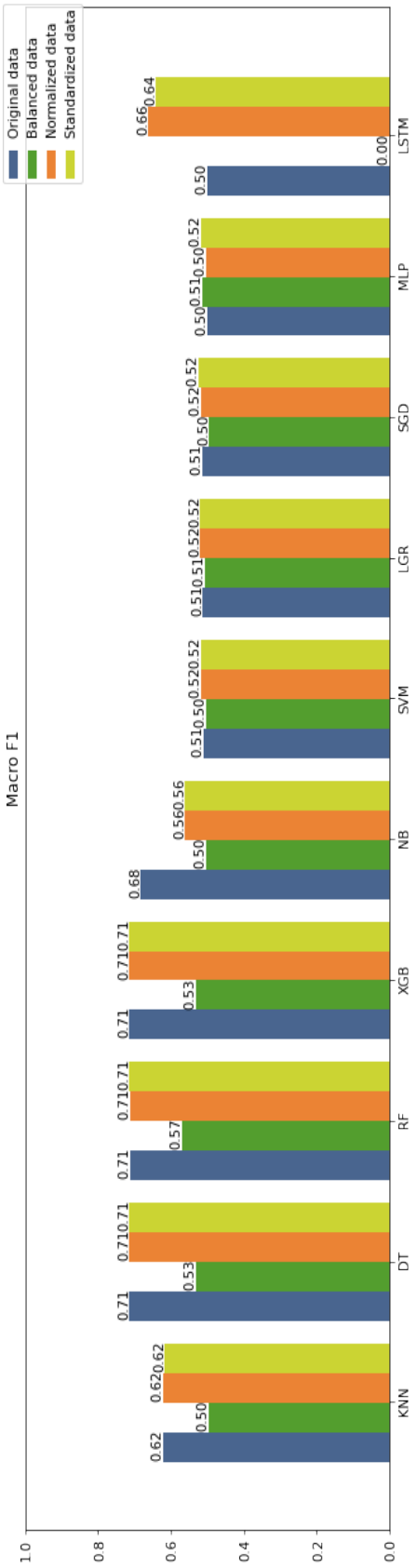


Figure 5.7: Macro F1 score with 30% failures

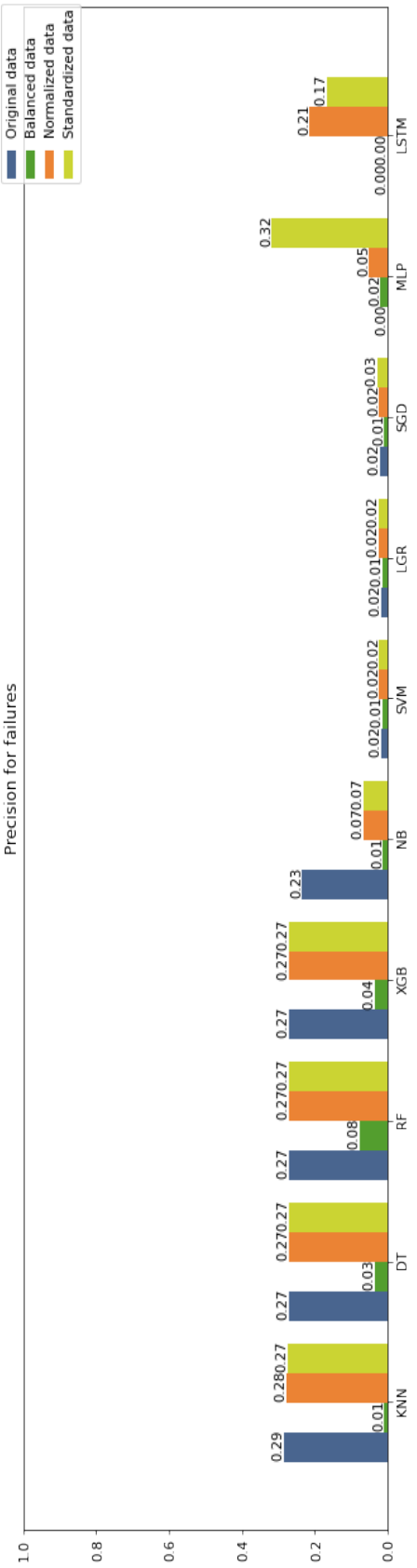


Figure 5.8: Precision score with 30% failures

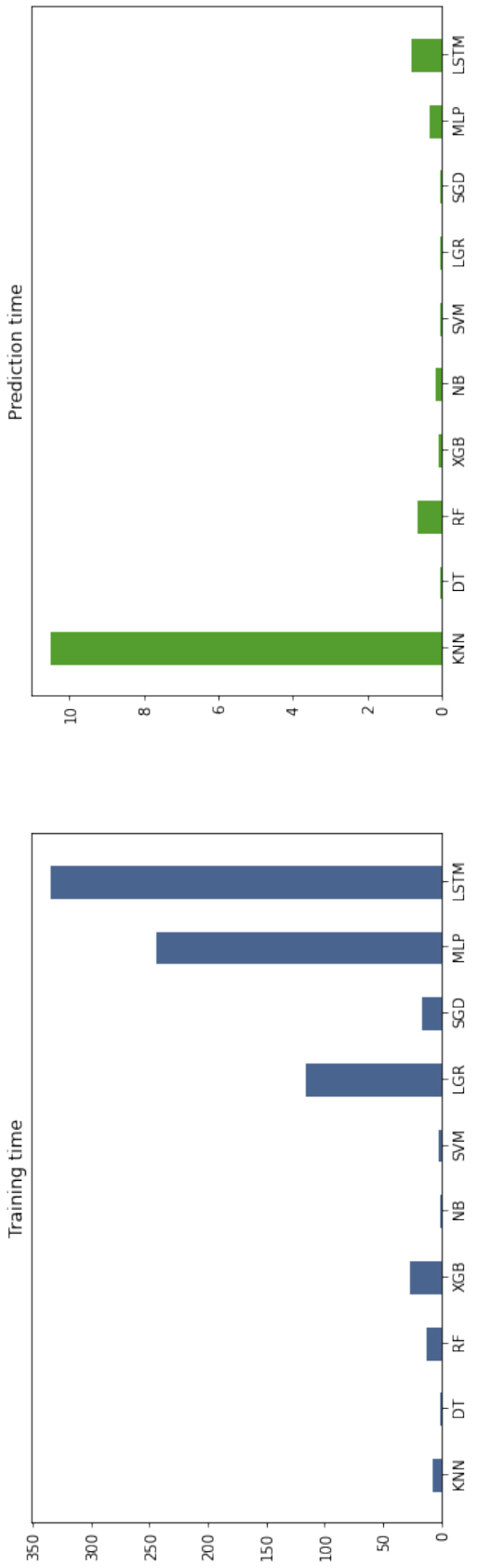


Figure 5.9: Training and prediction time

Alg.	Macro F1 70%	Macro F1 30%	Precision 70%	Precision 30%	Macro F1 Average	Precision Average	Training time	Prediction time
kNN	0.86	0.62	0.69	0.29	0.74	0.49	7.478	**10.51**
DT	**0.91**	**0.71**	0.69	0.27	**0.81**	0.48	1.297	0.022
RF	**0.91**	**0.71**	0.69	0.27	**0.81**	0.48	12.06	0.633
XGB	**0.91**	**0.71**	**0.70**	0.27	**0.81**	0.49	27.12	0.070
NB	0.86	0.68	0.66	0.23	0.77	0.45	**0.634**	0.147
SVM	**0.59**	**0.52**	**0.11**	**0.02**	**0.56**	**0.07**	2.748	0.016
LGR	0.60	**0.52**	**0.11**	**0.02**	**0.56**	**0.07**	116.1	0.019
SGD	0.62	**0.52**	0.14	0.03	0.57	0.09	17.09	**0.014**
MLP	0.79	**0.52**	0.68	**0.32**	0.66	**0.50**	244.2	0.319
LSTM	0.80	0.66	0.44	0.21	0.73	0.33	**334.7**	0.812

For macro F1 and precision, the highest value across data representations is selected.

Table 5.2: Evaluation results

5.7 Lead Time Prediction

In Section 2.3, different methods for failure prediction with lead time are presented. The algorithms evaluated in Section 5.6 can also provide lead time for maintenance by changing how the data is used for learning. For example, instead of using every data entry with the corresponding failure value for training, which provides no lead time, the failure values can be aggregated over a specific time interval into a single value: failure, if there is any failure in the time interval, non-failure otherwise. The algorithms are then trained by using each data entry and its corresponding aggregated failure value for the subsequent time interval. Consequently, each algorithm predicts the state of the device up to a specific point in the future. A similar approach is described by Rafiuzzaman et al. [RGPG19]. Instead of aggregating the failure values, they select individual points in the future and use the corresponding device state to train a kNN algorithm.

The approach presented in this section adopts the benefits of the LSTM for time series prediction. LSTM is implemented with Keras as a sequential model with two LSTM layers and a dense output layer. It receives a sequence of data entries (normalized feature values without failures) and outputs a sequence of failure values. Different sizes of input sequences

were tested. The choice was made on a single data entry, as it significantly reduces the training time and is sufficient for the algorithm to recognize the failure patterns. The length of the output sequence defines the length of the lead time. The prediction performance is tested on three different output sequences: 1 (1 second), 300 (5 minutes), and 1800 (30 minutes). Consequently, three different LSTM models were built. The dataset with 70% failures is used, creating a 75%-25% train-test split without shuffling. Furthermore, the data is prepared by creating output sequences for each data entry, which are then used for training. After the successful training process, the three LSTM models were evaluated on the test data, generating predictions shown in Figure 5.10. The green dots represent the true values, while the blue dots represent the predictions. The values range between 0 and 1 because of the sigmoid activation function in the output layer. The test data covers the days Friday, Saturday, Sunday and Monday. Because of the failure condition defined in Section 5.1, failures occur only during working days, which can be observed in the three diagrams. By looking at the green dots, the failures occur only at the start (Friday) and the end (Monday) of the time interval. The models were able to recognize the failure patterns, except for the false positive predictions on Saturday. This can be explained by the limited size of the training data. The data contains values for two weeks and there was only one other Saturday to learn from. The degrading prediction quality for increasing lead time can also be observed by the height of the peeks in the diagrams, starting from the 1 second predictions on the left, over 5 minutes in the middle, to 30 minutes on the right. Since the predictions are made each second for each data entry, the failure values in the subsequent output sequences are overlapping. As a result, there are multiple failure predictions for a single point in the future from different output sequences. To be able to interpret the results, the sequences are flattened out. For the lead times of 5 and 30 minutes, this results in far more predicted failure values than the size of the test data. However, the failure pattern is still recognizable as shown in Figure 5.10, and the transformation allows to compute the macro F1 and precision scores. Additional requirement to compute macro F1 and precision is to map the prediction values to either 1 or 0. They values are mapped by finding an optimal threshold. The macro F1 and precision scores are displayed in Figure 5.11. As expected, the highest F1 score has the prediction with 1 second lead time. Although the prediction with 5 minutes lead time has higher F1 score than the one with 30 minutes, the difference is very small and barely noticeable. The precision results are similar, prediction with 1 second lead time has the highest precision, followed by 5 and 30 minutes, which have almost the same score. The low overall precision can be explained by the specific data split and the high number of false positives displayed in Figure 5.10.

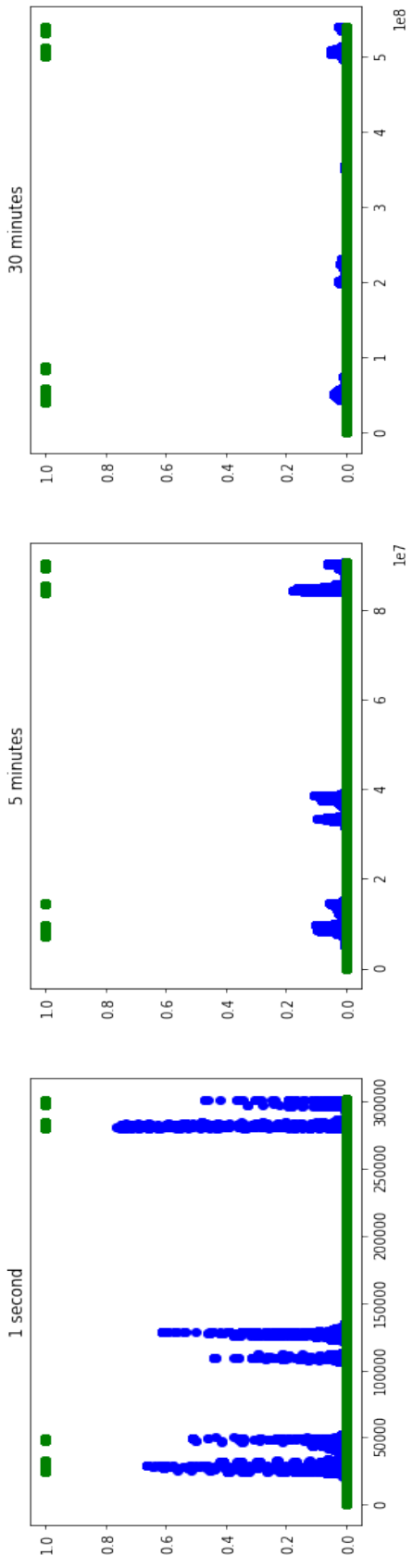


Figure 5.10: Lead time predictions

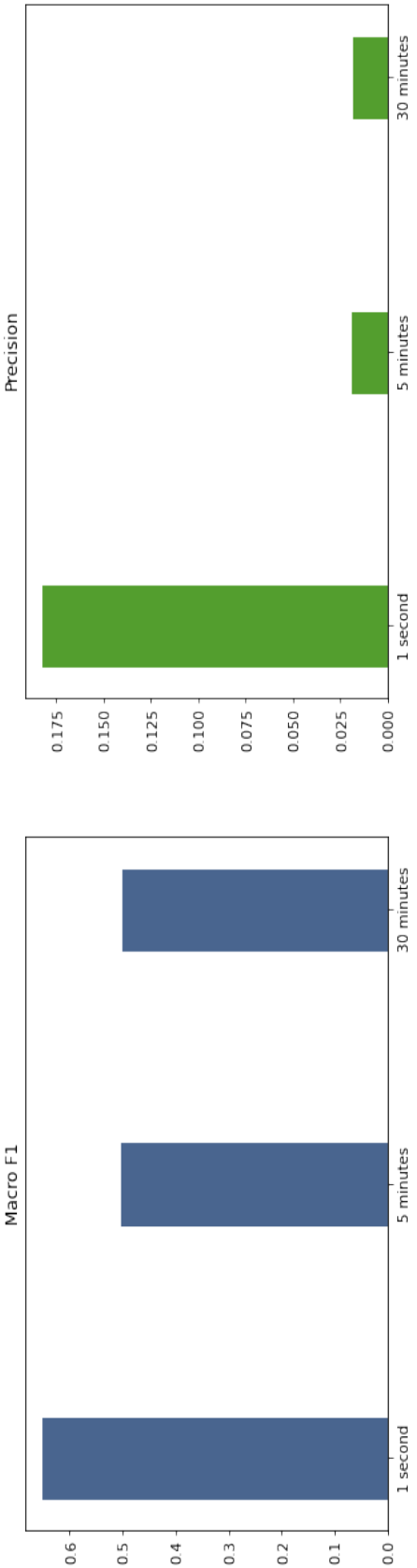


Figure 5.11: Macro F1 and precision for lead time predictions

6 Implementation

This chapter describes the prototype implementation of the Failure Prediction Platform presented in Chapter 4. Furthermore, an LSTM model is integrated in the platform to demonstrate its functionality.

6.1 Platform Implementation

The FPP was implemented with the JHipster[1] generator. JHipster generates a complete web application providing user management, responsive UI, monitoring, and other benefits. The prototype implementation uses a Java backend implemented with the Spring[2] framework, and an Angular[3] frontend. Furthermore, it stores data in the MongoDB database and uses Kafka[4] as the message broker. The FPP dashboard is shown in Figure 6.1.

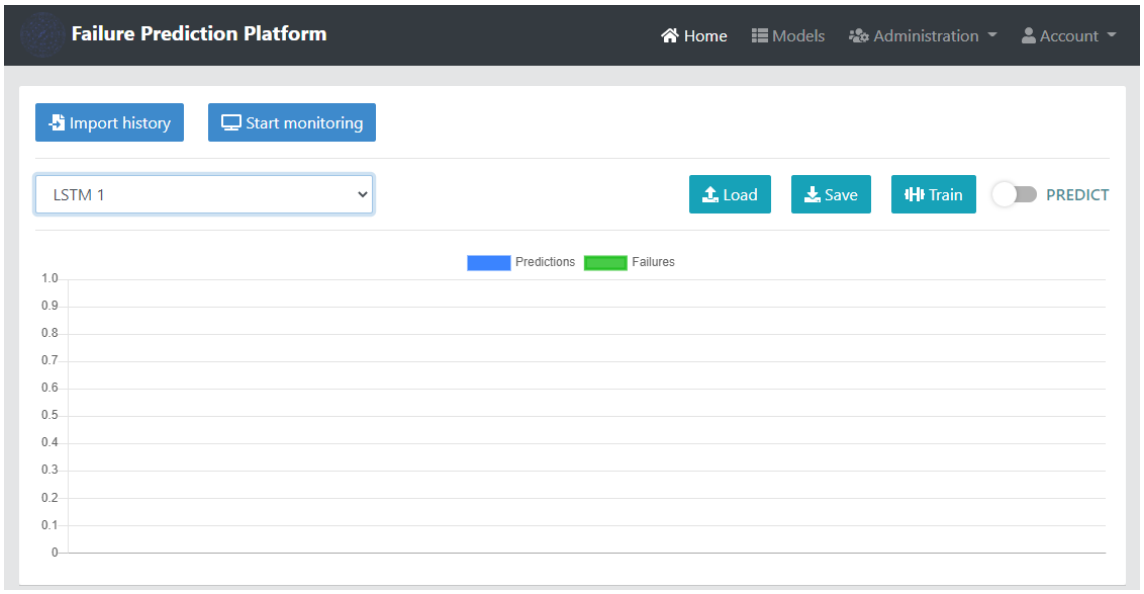


Figure 6.1: FPP dashboard

[1]https://www.jhipster.tech/
[2]https://spring.io/
[3]https://angular.io/
[4]https://kafka.apache.org/

The MBP, specifically MongoDB and Mosquitto, are used as data sources. As a result, an MBP adapter was implemented that builds a connection to the MongoDB database and a separate connection to the Mosquitto broker. However, the generated dataset was injected in the database and broker to demonstrate the training and prediction processes. Apart from the models and their individual functions visible in the dashboard, there are also two general functions required by all models. The first one imports the old data from the database and saves it on a specific topic unchanged. Kafka is a fast streaming platform that also provides permanent storage for the data. In the next step, the data is forwarded to the general data processing component implemented in Java. Since the generated dataset is used, the data is already processed except for removing the MongoDB IDs from each entry. This is conducted by defining a template of a data entry (which has no MongoDB ID) and mapping the incoming data by dropping the ID in the process. The processed data is saved again on another topic. The second function starts the monitoring process by establishing a connection to the Mosquitto broker and starting to receive the live data. The live data goes through the same path as the old data until it is saved again after processing. The data flow of the two processes is illustrated in Figure 6.2. Currently, there is no synchronization mechanism implemented and the two functions may produce duplicates if executed at the same time.

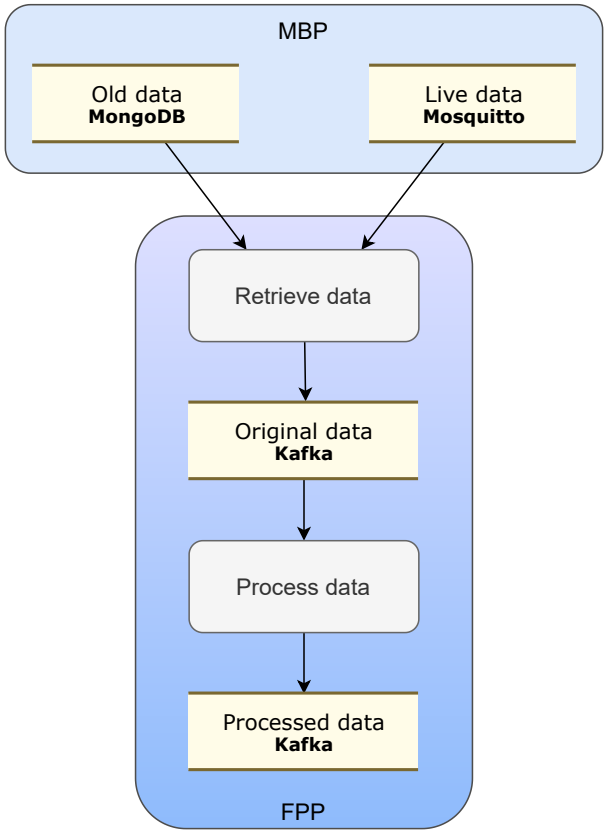


Figure 6.2: FPP data flow

6.2 Model Integration

An ML model can be integrated in FPP in three steps. An excerpt of the FPP architecture and the components relevant for the integration are shown in Figure 6.3.

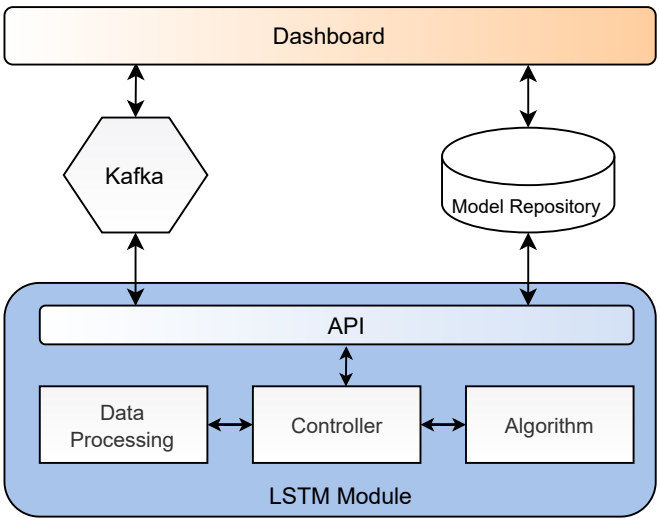


Figure 6.3: Excerpt of the FPP architecture for model integration

First of all, the model is implemented independently from the platform as an ML module. The module can be implemented in Python or in any other programming language, but it needs to implement a Kafka producer for predictions, and two Kafka consumers, for data and commands. Appropriate names for the topics need to be selected. In the second step, the model is registered over the dashboard of the platform, by providing a name, description, type and the topic names that are used for commands and predictions. The data is stored in the model repository. Figure 6.4 shows the model registration area. The registration area loads all the registered models, that can be viewed, edited and deleted. It further allows to register new models.

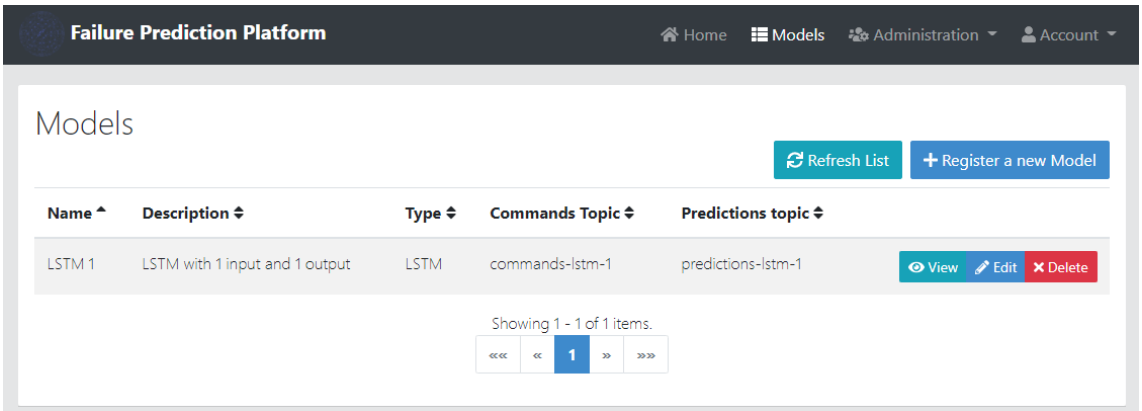


Figure 6.4: FPP model registration

In the final step, the module implementation is executed independently, enabling the interaction with the dashboard over Kafka. A simple LSTM model with an output sequence of a single failure value is used for the demonstration purpose. LSTM was chosen because of its support for online learning and time series prediction. The corresponding module is implemented in Python and integrated by following the steps described above. It uses an abstract API component with Kafka producer and consumers, and predefined abstract methods to implement the interface. This allows consistent API implementation across the models. Furthermore, a custom data processing component is implemented for LSTM that transforms the features, prepares sequences and uses a scaler to normalize the data. The final components of the LSTM module are the controller and the algorithm. The controller processes the commands and performs specific actions. It orchestrates the components in the module, and also starts and stops the training and prediction of the algorithm.

6.3 Training and Prediction

On dashboard initialization, all the existing models and their prediction histories are loaded, as shown in Figure 6.5. Each model has four associated functions. The models can be trained by accessing the topic with the processed data entries. In this implementation, a limited number of data entries are used, allowing all the data to be loaded at once. However, the entries should be retrieved in limited numbers or batches. After the training is completed, the prediction can be started by listening on new data entries on the same topic. The prediction can also be started immediately if online learning is implemented for the particular model. Furthermore, each model, and the corresponding scaler, can be saved as files and loaded at any time.

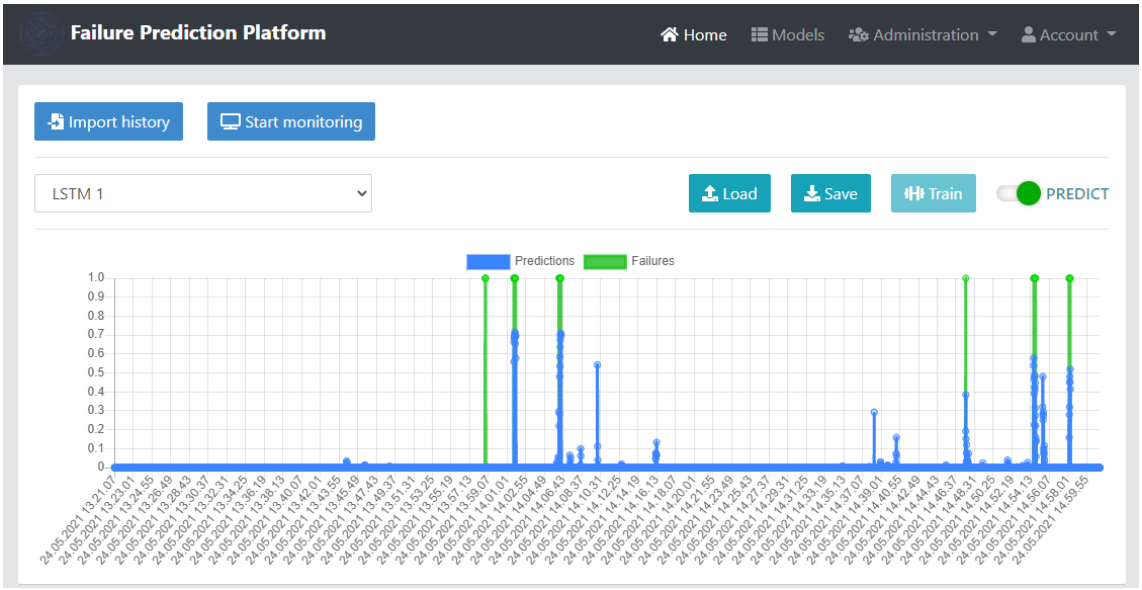


Figure 6.5: FPP training and prediction

The corresponding commands are first sent to the backend over the REST API, and the backend implements a Kafka producer that forwards the commands to the modules. The backend also implements a Kafka consumer, that receives the predictions from the modules and sends them to the dashboard as push notifications in an open HTTP connection. The predictions are visualized with chart.js[5], as displayed in Figure 6.5 for LSTM. Apart from the predictions displayed in blue, the green dots represent the true values added for better interpretation. In this particular implementation, the LSTM algorithm is first trained using the old data from the database and then the predictions were started. Since it is implemented as an online learning approach, it also learns with every prediction. The prediction process can be stopped and resumed at any point.

[5]https://www.chartjs.org/

7 Conclusion and Outlook

As part of this thesis, several ML algorithms were selected and compared regarding their failure prediction performance. The algorithms were evaluated on synthetically generated IoT datasets by simulating failures of IoT devices. Furthermore, a concept and a prototypical implementation of a platform intended to host multiple ML models for failure predictions was presented.

Since real-world IoT data is typically characterized by very rare failure occurrences in relation to its size, synthetic IoT data was generated to train and test the algorithms in a short time interval. A specific failure pattern was defined and injected in the data. It occurs 70% and 30% of the time, generating two different datasets. The idea is to show the behavior of the algorithms on two different levels of imbalanced data. In the beginning of the Chapter 5, the data generation process is described and the characteristics of the generated datasets are displayed. The datasets were modified creating four different representations: original, balanced, normalized and standardized. Each algorithm was tested separately on each representation. Furthermore, additional pre-processing steps were performed on the data. The algorithms were selected based on several criteria, such as the type of the task they solve, their application in practice, diversity, etc. As a result, ten algorithms were selected, described and compared. After implementing the algorithms following a set of rules, they were tested on the different data representations using 4-fold cross-validation and randomized automated hyperparameter tuning. The randomized search for the parameters showed to be an efficient and sufficient approach to reach relatively high performances. The metrics selected to evaluate the prediction performance of the algorithms were macro F1 and precision. The algorithms were then evaluated on the generated datasets by using the selected metrics. There is no particular difference in the performance order of the algorithms for the two datasets, except for MLP and LSTM, which performed worse on the dataset with lower number of failures. Additionally, balancing the data in the same dataset further decreased the prediction performance. Overall, the tree-based algorithms DT, RF and XGB provided the best and most optimal results, while the linear classifiers SVM, LGR and SGD performed the worst. There were also differences regarding the data representation. As expected, balancing the data almost always decreased the prediction performance. Normalization and standardization of the data typically improved the performance, especially for NNs and linear classifiers. The second criteria used to compare the algorithms was the training and prediction time. After discovering the optimal parameters and executing the algorithms on the original data, the training and prediction time was measured. By looking at the training speed, MLP and LSTM followed by LGR were the slowest algorithms. The best results provided DT, NB

and SVM. The fastest algorithms with respect to the prediction time were DT and linear classifiers, while kNN as lazy learner was the slowest algorithm. The overall result for the prediction performance and execution speed should be taken with caution. They depend on the parameter configuration and the generated datasets, and can typically be improved by providing more training time. The results provide only a brief comparison of different types of algorithms regarding their failure prediction performance and execution time on data with IoT characteristics. Chapter 5 ends with a short description of lead time prediction techniques. The LSTM approach for time series prediction was evaluated on three different lead times (1s, 5min, 30min). Even though the prediction quality decreased with increased lead time, LSTM was still able to recognize the failure patterns.

The other part of the thesis describes the architecture and workflow of the FPP. The FPP assumes the existence of an IoT platform that provides data sources for historic and live data in a lambda-based architecture. The main components and interactions with the environment are further described. The platform uses a message broker to deal with the data streams and a general processing component to prepare the data for the algorithms. The ML models can be integrated as uniform modules in a loosely coupled manner. They consists of the same type of components, providing consistency and reusability. The platform also provides a dashboard, which the maintainers can use to monitor the state of the devices and eventually prevent failures. Moreover, a prototype of the platform was implemented and described in Chapter 6. Using LSTM as an example, the training and prediction processes in the prototype are described and visualized. LSTM was integrated as an online learning and time series prediction technique. The FPP enables uncomplicated and concurrent management of different types of ML models.

Outlook

To improve the evaluation of the ML algorithms, more testing with more data needs to be completed. Better results can be achieved by increasing the time for training and parameter tuning. Furthermore, an IoT environment can be set up to collect real IoT data over longer periods of time. Online learning can be applied to deal with the huge amount of data. Another important aspect are the techniques that provide predictions with enough lead time for maintenance. The focus of the future work can be ML algorithms or variants that support online learning but also provide sufficient time for maintenance.

The FPP is a generic platform where ML models can be easily integrated as modules. The modules are implemented independently and require manual intervention in the platform for the integration. The integration process can be automated and further simplified. Since the modules have the same composition for all models, the structures of the components can be predefined and used as templates. The model can then be integrated over the FPP dashboard by providing only the configuration of the algorithm. Furthermore, by implementing automated hyperparameter tuning, the configuration can also be omitted to some degree. There are other aspects of FPP that can also be improved. The training

and prediction processes are not clearly separated regarding the data flow. A controlled synchronization between them is required to avoid double processing. Apart from the failure prediction, failure detection and analysis can be implemented to find out which features and values are causing the failures. A notification system can also be implemented to detect critical states of devices and send warnings to the maintainers.

Bibliography

[AG17] O. Aydin, S. Guldamlasioglu. "Using LSTM networks to predict engine condition on large scale data processing framework". In: *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*. 2017, pp. 281–285. DOI: 10.1109/ICEEE2.2017.7935834 (cit. on pp. 26, 27).

[Agg15] C. C. Aggarwal. *Data mining: the textbook*. Springer, 2015 (cit. on pp. 35, 40, 47).

[Aha13] D. W. Aha. *Lazy learning*. Springer Science & Business Media, 2013 (cit. on p. 19).

[AKYC20] M. H. Alsharif, A. H. Kelechi, K. Yahya, S. A. Chaudhry. "Machine Learning Algorithms for Smart Data Analysis in Internet of Things Environment: Taxonomies and Research Trends". In: *Symmetry* 12.1 (2020). ISSN: 2073-8994. DOI: 10.3390/sym12010088. URL: https://www.mdpi.com/2073-8994/12/1/88 (cit. on p. 19).

[AMB20] T. Ara, P. M, M. Bali. "Fault Prediction in Wireless Sensor Networks using Soft Computing". In: *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. Oct. 2020, pp. 532–538. DOI: 10.1109/ICSTCEE49637.2020.9277216 (cit. on p. 24).

[AS08] A. Andrzejak, L. Silva. "Using machine learning for non-intrusive modeling and prediction of software aging". In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. 2008, pp. 25–32. DOI: 10.1109/NOMS.2008.4575113 (cit. on p. 25).

[ASZS15] A. Abu-Samah, M. Shahzad, E. Zamai, A. Said. "Failure Prediction Methodology for Improved Proactive Maintenance using Bayesian Approach". In: *IFAC-PapersOnLine* 48.21 (2015). 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015, pp. 844–851. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.09.632. URL: http://www.sciencedirect.com/science/article/pii/S2405896315017619 (cit. on p. 27).

[ATBG10] J. Alonso, J. Torres, J. L. Berral, R. Gavaldà. "Adaptive on-line software aging prediction based on machine learning". In: *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*. 2010, pp. 507–516. DOI: 10.1109/DSN.2010.5544275 (cit. on p. 25).

[AXDS19] J. Alter, J. Xue, A. Dimnaku, E. Smirni. "SSD Failures in the Field: Symptoms, Causes, and Prediction Models". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019. ISBN: 9781450362290. DOI: 10.1145/3295500.3356172. URL: https://doi.org/10.1145/3295500.3356172 (cit. on p. 25).

[BB05] A. Bordes, L. Bottou. "The Huller: a simple and efficient online SVM". In: *In Machine Learning: ECML 2005, Lecture Notes in Artificial Intelligence, LNAI 3720*. Springer Verlag, 2005, pp. 505–512 (cit. on p. 42).

[BB86] R. N. Bracewell, R. N. Bracewell. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York, 1986 (cit. on p. 39).

[BDA13] M. Bekkar, H. K. Djemaa, T. A. Alitouche. "Evaluation measures for models assessment over imbalanced data sets". In: *J Inf Eng Appl* 3.10 (2013) (cit. on p. 45).

[ÇAZ+20] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, B. Safaei. "Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0". In: *Sustainability* 12.19 (Oct. 2020), p. 8211. ISSN: 2071-1050. DOI: 10.3390/su12198211. URL: http://dx.doi.org/10.3390/su12198211 (cit. on pp. 13, 26, 27).

[CCV19] J. R. Campos, E. Costa, M. Vieira. "Improving Failure Prediction by Ensembling the Decisions of Machine Learning Models: A Case Study". In: *IEEE Access* 7 (2019), pp. 177661–177674. DOI: 10.1109/ACCESS.2019.2958480 (cit. on p. 26).

[CG16] T. Chen, C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785 (cit. on pp. 41, 44).

[Cho+15] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras (cit. on p. 44).

[CPV+17] C. A. C. Rincón, J. Pâris, R. Vilalta, A. M. K. Cheng, D. D. E. Long. "Disk failure prediction in heterogeneous environments". In: *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. 2017, pp. 1–7. DOI: 10.23919/SPECTS.2017.8046776 (cit. on p. 25).

[CSV+19] T. Carvalho, F. Soares, R. Vita, R. Francisco, J. Basto, S. G. Soares Alcalá. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (Sept. 2019), p. 106024. DOI: 10.1016/j.cie.2019.106024 (cit. on pp. 13, 26, 27).

[CVC18] J. R. Campos, M. Vieira, E. Costa. "Exploratory Study of Machine Learning Techniques for Supporting Failure Prediction". In: *2018 14th European Dependable Computing Conference (EDCC)*. 2018, pp. 9–16. DOI: 10.1109/EDCC.2018.00014 (cit. on p. 26).

[CWS+18] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, B. Yin. "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges". In: *IEEE Access* 6 (2018), pp. 6505–6519. DOI: 10.1109/ACCESS.2017.2783682 (cit. on pp. 13, 17).

[CYM+18] R. Cao, Z. Yu, T. Marbach, J. Li, G. Wang, X. Liu. "Load Prediction for Data Centers Based on Database Service". In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 01. 2018, pp. 728–737. DOI: 10.1109/COMPSAC.2018.00109 (cit. on p. 26).

[DH18] J. Deutsch, D. He. "Using Deep Learning-Based Approach to Predict Remaining Useful Life of Rotating Components". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.1 (2018), pp. 11–20. DOI: 10.1109/TSMC.2017.2697842 (cit. on pp. 26, 27).

[DL18] X. Du, C. Li. "Memory Failure Prediction Using Online Learning". In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS '18. Alexandria, Virginia, USA: Association for Computing Machinery, 2018, pp. 38–49. ISBN: 9781450364751. DOI: 10.1145/3240302.3240309. URL: https://doi.org/10.1145/3240302.3240309 (cit. on p. 25).

[DNP+18] J. C. Duenas, J. M. Navarro, H. A. Parada G., J. Andion, F. Cuadrado. "Applying Event Stream Processing to Network Online Failure Prediction". In: *IEEE Communications Magazine* 56.1 (2018), pp. 166–170. DOI: 10.1109/MCOM.2018.1601135 (cit. on p. 25).

[DRH20] D. Del Gaudio, M. Reichel, P. Hirmer. "A Life Cycle Method for Device Management in Dynamic IoT Environments". In: Jan. 2020, pp. 46–56. DOI: 10.5220/0009340900460056 (cit. on p. 29).

[EACF17] S. Eke, T. Aka-Ngnui, G. Clerc, I. Fofana. "Characterization of the operating periods of a power transformer by clustering the dissolved gas data". In: *2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*. 2017, pp. 298–303. DOI: 10.1109/DEMPED.2017.8062371 (cit. on p. 27).

[FEV+18] X. Fafoutis, A. Elsts, A. Vafeas, G. Oikonomou, R. Piechocki. "On Predicting the Battery Lifetime of IoT Devices: Experiences from the SPHERE Deployments". In: *Proceedings of the 7th International Workshop on Real-World Embedded Wireless Systems and Networks*. RealWSN'18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 7–12. ISBN: 9781450360487. DOI: 10.1145/3277883.3277892. URL: https://doi.org/10.1145/3277883.3277892 (cit. on p. 24).

[FHM09] C. Ferri, J. Hernández-Orallo, R. Modroiu. "An experimental comparison of performance measures for classification". In: *Pattern Recognition Letters* 30.1 (2009), pp. 27–38. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2008.08.010. URL: https://www.sciencedirect.com/science/article/pii/S0167865508002687 (cit. on p. 45).

[FHS+20] A. C. Franco da Silva, P. Hirmer, J. Schneider, S. Ulusal, M. T. Frigo. "MBP: Not just an IoT Platform". In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2020, pp. 1–3. DOI: 10.1109/PerComWorkshops48775.2020.9156156 (cit. on pp. 14, 18).

[Fri01] J. H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 00905364. URL: http://www.jstor.org/stable/2699986 (cit. on p. 41).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 40).

[GBF+16] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, L. Reinfurt. "Comparison of IoT platform architectures: A field study based on a reference architecture". In: *2016 Cloudification of the Internet of Things (CIoT)*. 2016, pp. 1–6. DOI: 10.1109/CIOT.2016.7872918 (cit. on pp. 17, 18).

[GO21] H. Guo, A. A. Ofori. "The Internet of Things in Extreme Environments Using Low-Power Long-Range Near Field Communication". In: *IEEE Internet of Things Magazine* 4.1 (2021), pp. 34–38. DOI: 10.1109/IOTM.0011.2000063 (cit. on p. 13).

[GSWB17] I. Giurgiu, J. Szabo, D. Wiesmann, J. Bird. "Predicting DRAM Reliability in the Field with Machine Learning". In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*. Middleware '17. Las Vegas, Nevada: Association for Computing Machinery, 2017, pp. 15–21. ISBN: 9781450352000. DOI: 10.1145/3154448.3154451. URL: https://doi.org/10.1145/3154448.3154451 (cit. on p. 25).

[GUL+20] H. A. Gohel, H. Upadhyay, L. Lagos, K. Cooper, A. Sanzetenea. "Predictive maintenance architecture development for nuclear infrastructure using machine learning". In: *Nuclear Engineering and Technology* 52.7 (2020), pp. 1436–1442. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2019.12.029. URL: http://www.sciencedirect.com/science/article/pii/S1738573319306783 (cit. on pp. 26, 27).

[GWB17] I. Giurgiu, D. Wiesmann, J. Bird. ""Memory Lossïn Commodity Hardware? Predicting DIMM Failures with Machine Learning". In: *Proceedings of the 10th ACM International Systems and Storage Conference*. SYSTOR '17. Haifa, Israel: Association for Computing Machinery, 2017. ISBN: 9781450350358. DOI: 10.1145/3078468.3078486. URL: https://doi.org/10.1145/3078468.3078486 (cit. on p. 25).

[HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. "Automating the Provisioning and Configuration of Devices in the Internet of Things." In: *CSIMQ* 9 (2016), pp. 28–43 (cit. on pp. 14, 18).

[HHX+20] L. Hu, L. Han, Z. Xu, T. Jiang, H. Qi. "A disk failure prediction method based on LSTM network due to its individual specificity". In: *Procedia Computer Science* 176 (2020). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020, pp. 791–799. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.09.074. URL: https://www.sciencedirect.com/science/article/pii/S1877050920319700 (cit. on p. 22).

[HJ18] T. Huuhtanen, A. Jung. "PREDICTIVE MAINTENANCE OF PHOTOVOLTAIC PANELS VIA DEEP LEARNING". In: *2018 IEEE Data Science Workshop (DSW)*. 2018, pp. 66–70. DOI: 10.1109/DSW.2018.8439898 (cit. on pp. 26, 27).

[HM15] M. Hossin, S. M.N. "A Review on Evaluation Metrics for Data Classification Evaluations". In: *International Journal of Data Mining & Knowledge Management Process* 5 (Mar. 2015), pp. 01–11. DOI: 10.5121/ijdkp.2015.5201 (cit. on p. 45).

[HTF09] T. Hastie, R. Tibshirani, J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009 (cit. on pp. 40, 46).

[Hua17] X. Huang. *Hard Drive Failure Prediction for Large Scale Storage System*. June 2017. URL: https://escholarship.org/uc/item/11x380ng (cit. on p. 25).

[IVD15] I. Irrera, M. Vieira, J. Duraes. "Adaptive Failure Prediction for Computer Systems: A Framework and a Case Study". In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. 2015, pp. 142–149. DOI: 10.1109/HASE.2015.29 (cit. on p. 26).

[JDC+18] W. Ji, S. Duan, R. Chen, S. Wang, Q. Ling. "A CNN-based network failure prediction method with logs". In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, pp. 4087–4090. DOI: 10.1109/CCDC.2018.8407833 (cit. on p. 25).

[JM18] A. Jaiswal, R. Malhotra. "Software reliability prediction using machine learning techniques". In: *International Journal of System Assurance Engineering and Management* 9.1 (2018), pp. 230–244 (cit. on p. 26).

[JSC13] F. Jiang, Y. Sui, C. Cao. "An incremental decision tree algorithm based on rough sets and its application in intrusion detection". In: *Artificial Intelligence Review* 40.4 (2013), pp. 517–530 (cit. on p. 41).

[JYS19] D. Jauk, D. Yang, M. Schulz. "Predicting Faults in High Performance Computing Systems: An in-Depth Survey of the State-of-the-Practice". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019. ISBN: 9781450362290. DOI: 10.1145/3295500.3356185. URL: https://doi.org/10.1145/3295500.3356185 (cit. on p. 26).

[Kar20] L. Karlsson. "Predictive Maintenance for RM12 with Machine Learning". MA thesis. Halmstad, Sweden: Halmstad University, 2020 (cit. on pp. 26, 27).

[KC17] S. Khunteta, A. K. R. Chavva. "Deep Learning Based Link Failure Mitigation". In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2017, pp. 806–811. DOI: 10.1109/ICMLA.2017.00-58 (cit. on p. 25).

[Ket17] N. Ketkar. "Stochastic Gradient Descent". In: *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, pp. 113–132. ISBN: 978-1-4842-2766-4. DOI: 10.1007/978-1-4842-2766-4_8. URL: https://doi.org/10.1007/978-1-4842-2766-4_8 (cit. on p. 43).

[KMM+15] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja. "Lambda architecture for cost-effective batch and speed big data processing". In: *2015 IEEE International Conference on Big Data (Big Data)*. 2015, pp. 2785–2792. DOI: 10.1109/BigData.2015.7364082 (cit. on p. 29).

[KÖUG17] İ. Karakurt, S. Özer, T. Ulusinan, M. C. Ganiz. "A machine learning approach to database failure prediction". In: *2017 International Conference on Computer Science and Engineering (UBMK)*. 2017, pp. 1030–1035. DOI: 10.1109/UBMK.2017.8093426 (cit. on p. 26).

[KS12] P. Kumar, Y. Singh. "An empirical study of software reliability prediction using machine learning techniques". In: *International Journal of System Assurance Engineering and Management* 3.3 (2012), pp. 194–208 (cit. on p. 26).

[KSH11] J.-M. Kang, S.-s. Seo, J. W.-K. Hong. "Personalized battery lifetime prediction for mobile devices based on usage patterns". In: *Journal of Computing Science and Engineering* 5.4 (2011), pp. 338–345 (cit. on p. 24).

[KVIT18] N. Kolokas, T. Vafeiadis, D. Ioannidis, D. Tzovaras. "Forecasting faults of industrial equipment using machine learning classifiers". In: *2018 Innovations in Intelligent Systems and Applications (INISTA)*. 2018, pp. 1–6. DOI: 10.1109/INISTA.2018.8466309 (cit. on pp. 26, 27).

[KWH13] H. Kagermann, W. Wahlster, J. Helbig. "Securing the future of German manufacturing industry: Recommendations for implementing the strategic initiative INDUSTRIE 4.0". In: *Final report of the Industrie* 4.0 (2013) (cit. on p. 26).

[LCH19] G. Lian, W. Chen, S. Huang. "Cloud-Based Online Ageing Monitoring for IoT Devices". In: *IEEE Access* 7 (2019), pp. 135964–135971. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2941998 (cit. on p. 24).

[LLM18] H. Li, X. Liu, Q. Mei. *Predicting Smartphone Battery Life based on Comprehensive and Real-time Usage Data*. 2018. arXiv: 1801.04069 [cs.HC] (cit. on p. 24).

[LLP+20] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, W. Shi. "Making Disk Failure Predictions SMARTer!" In: *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 151–167. ISBN: 978-1-939133-12-0. URL: https://www.usenix.org/conference/fast20/presentation/lu (cit. on p. 25).

[MMST16] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma. "A gap analysis of Internet-of-Things platforms". In: *Computer Communications* 89-90 (2016). Internet of Things Research challenges and Solutions, pp. 5–16. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.03.015. URL: https://www.sciencedirect.com/science/article/pii/S0140366416300731 (cit. on p. 17).

[Mob02] R. K. Mobley. *An introduction to predictive maintenance*. Elsevier, 2002 (cit. on pp. 13, 21).

[MRT18] M. Mohri, A. Rostamizadeh, A. Talwalkar. *Foundations of machine learning*. MIT press, 2018 (cit. on pp. 13, 19).

[MSZL18] J. Ma, H. Su, W.-l. Zhao, B. Liu. "Predicting the remaining useful life of an aircraft engine using a stacked sparse autoencoder with multilayer self-learning". In: *Complexity* 2018 (2018). DOI: https://doi.org/10.1155/2018/3813029 (cit. on pp. 26, 27).

[MTS+17] V. Mathew, T. Toby, V. Singh, B. M. Rao, M. G. Kumar. "Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning". In: *2017 IEEE International Conference on Circuits and Systems (ICCS)*. 2017, pp. 306–311. DOI: 10.1109/ICCS1.2017.8326010 (cit. on pp. 22, 27).

[MWL+20] R. Ma, F. Wu, Z. Lu, W. Zhong, Q. Wu, J. Wan, C. Xie. "BlockHammer: Improving Flash Reliability by Exploiting Process Variation Aware Proactive Failure Prediction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (2020), pp. 4563–4574. DOI: 10.1109/TCAD.2020.2981025 (cit. on p. 25).

[NG20] T. R. N, R. Gupta. "A Survey on Machine Learning Approaches and Its Techniques:" in: *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*. 2020, pp. 1–6. DOI: 10.1109/SCEECS48394.2020.190 (cit. on p. 19).

[Nie15] M. A. Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, 2015 (cit. on p. 20).

[NPK+16] H. Narasimhan, W. Pan, P. Kar, P. Protopapas, H. G. Ramaswamy. "Optimizing the Multiclass F-Measure via Biconcave Programming". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1101–1106. DOI: 10.1109/ICDM.2016.0143 (cit. on p. 45).

[OB21] J. Opitz, S. Burst. *Macro F1 and Macro F1*. 2021. arXiv: 1911.03347 [cs.LG] (cit. on p. 45).

[PHG13] T. Pitakrat, A. van Hoorn, L. Grunske. "A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection". In: *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*. ISARCS '13. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2013, pp. 1–10. ISBN: 9781450321235. DOI: 10.1145/2465470.2465473. URL: https://doi.org/10.1145/2465470.2465473 (cit. on p. 25).

[PRF+18] M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni, J. Loncarski. "Machine Learning approach for Predictive Maintenance in Industry 4.0". In: *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*. 2018, pp. 1–6. DOI: 10.1109/MESA.2018.8449150 (cit. on pp. 26, 27).

[PSA15] A. Pellegrini, P. D. Sanzo, D. R. Avresky. "A Machine Learning-Based Framework for Building Application Failure Prediction Models". In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. 2015, pp. 1072–1081. DOI: 10.1109/IPDPSW.2015.110 (cit. on p. 25).

[PU20] T. Paul, K. Ueno. "Robust Incremental Logistic Regression for Detection of Anomaly Using Big Data". In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2020, pp. 1167–1173. DOI: 10.1109/ICMLA51294.2020.00187 (cit. on p. 43).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 44).

[PVK19] N. Pandey, O. P. Verma, A. Kumar. "A framework for usage pattern–based power optimization and battery lifetime prediction in smartphones". In: *Personal and Ubiquitous Computing* (2019), pp. 1–16 (cit. on p. 24).

[PYAS20] R. Pinciroli, L. Yang, J. Alter, E. Smirni. *The Life and Death of SSDs and HDDs: Similarities, Differences, and Prediction Models*. 2020. arXiv: 2012.12373 [cs.LG] (cit. on p. 25).

[RGPG19] M. Rafiuzzaman, J. Gascon-Samson, K. Pattabiraman, S. Gopalakrishnan. "Failure Prediction in the Internet of Things Due to Memory Exhaustion". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, pp. 292–301. ISBN: 9781450359337. DOI: 10.1145/3297280.3297311. URL: https://doi.org/10.1145/3297280.3297311 (cit. on pp. 13, 22, 23, 51).

[RT17] B. L. Risteska Stojkoska, K. V. Trivodaliev. "A review of Internet of Things for smart home: Challenges and solutions". In: *Journal of Cleaner Production* 140 (2017), pp. 1454–1464. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2016.10.006. URL: https://www.sciencedirect.com/science/article/pii/S095965261631589X (cit. on pp. 13, 17).

[RZL+19] Y. Ran, X. Zhou, P. Lin, Y. Wen, R. Deng. "A Survey of Predictive Maintenance: Systems, Purposes and Approaches". In: *ArXiv* abs/1912.07383 (2019) (cit. on pp. 21, 22, 27).

[SA21] B. M. Salih Hasan, A. M. Abdulazeez. "A Review of Principal Component Analysis Algorithm for Dimensionality Reduction". In: *Journal of Soft Computing and Data Mining* 2.1 (Apr. 2021), pp. 20–30. URL: https://publisher.uthm.edu.my/ojs/index.php/jscdm/article/view/8032 (cit. on p. 39).

[SB14] S. Shalev-Shwartz, S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014 (cit. on p. 19).

[SBB+20] M. Seifeddine, A. Bradai, S. Bukhari, Q. Pham Tran Anh, O. Ben Ahmed, M. Atri. "A survey on machine learning in Internet of Things: Algorithms, strategies, and applications". In: *Internet of Things* (Nov. 2020). DOI: 10.1016/j.iot.2020.100314 (cit. on p. 19).

[SCH+19] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, L. Jiang. "System-Level Hardware Failure Prediction Using Deep Learning". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA: Association for Computing Machinery, 2019. ISBN: 9781450367257. DOI: 10.1145/3316781.3317918. URL: https://doi.org/10.1145/3316781.3317918 (cit. on p. 25).

[SFK19] M. Soualhia, C. Fu, F. Khomh. "Infrastructure Fault Detection and Prediction in Edge Cloud Environments". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. SEC '19. Arlington, Virginia: Association for Computing Machinery, 2019, pp. 222–235. ISBN: 9781450367332. DOI: 10.1145/3318216.3363305. URL: https://doi.org/10.1145/3318216.3363305 (cit. on p. 23).

[SL09] M. Sokolova, G. Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information Processing & Management* 45.4 (2009), pp. 427–437. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2009.03.002. URL: https://www.sciencedirect.com/science/article/pii/S0306457309000259 (cit. on p. 45).

[SLF11] K. Su, J. Li, H. Fu. "Smart city and the applications". In: *2011 International Conference on Electronics, Communications and Control (ICECC)*. 2011, pp. 1028–1031. DOI: 10.1109/ICECC.2011.6066743 (cit. on p. 17).

[SR15] T. Saito, M. Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". In: *PLOS ONE* 10.3 (Mar. 2015), pp. 1–21. DOI: 10.1371/journal.pone.0118432. URL: https://doi.org/10.1371/journal.pone.0118432 (cit. on p. 45).

[SVSA19] G. Scalabrini Sampaio, A. R. d. A. Vallim Filho, L. Santos da Silva, L. Augusto da Silva. "Prediction of Motor Failure Time Using An Artificial Neural Network". In: *Sensors* 19.19 (Oct. 2019), p. 4342. ISSN: 1424-8220. DOI: 10.3390/s19194342. URL: http://dx.doi.org/10.3390/s19194342 (cit. on pp. 26, 27).

[SWM12] M. Sonoda, Y. Watanabe, Y. Matsumoto. "Prediction of failure occurrence time based on system log message pattern learning". In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 578–581. DOI: 10.1109/NOMS.2012.6211960 (cit. on p. 26).

[SYW+19] N. Suga, K. Yano, J. Webber, Y. Hou, T. Higashimori, Y. Suzuki. "Prediction of QoS Outage Probability for Wireless Communication in Factory Environments". In: *2019 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*. Dec. 2019, pp. 124–129. DOI: 10.1109/IINTEC48298.2019.9112098 (cit. on pp. 13, 24).

[VF13] O. Vermesan, P. Friess, eds. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communication. Aalborg: River, 2013. ISBN: 978-87-92982-73-5. URL: http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf (cit. on pp. 13, 17).

[WSCL13] E. Westkämper, D. Spath, C. Constantinescu, J. Lentes. *Digitale Produktion*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2013. ISBN: 9783642202599. URL: https://books.google.de/books?id=FhgeBAAAQBAJ (cit. on p. 26).

[WWCL09] A. Wang, G. Wan, Z. Cheng, S. Li. "An incremental extremely random forest classifier for online learning and tracking". In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 1449–1452. DOI: 10.1109/ICIP.2009.5414559 (cit. on p. 41).

[WZW+17] Z. Wang, M. Zhang, D. Wang, C. Song, M. Liu, J. Li, L. Lou, Z. Liu. "Failure prediction using machine learning and time series in optical network". In: *Opt. Express* 25.16 (Aug. 2017), pp. 18553–18565. DOI: 10.1364/OE.25.018553. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-25-16-18553 (cit. on p. 25).

[XHL18] S. Xiang, D. Huang, X. Li. "A Generalized Predictive Framework for Data Driven Prognostics and Diagnostics using Machine Logs". In: *TENCON 2018 - 2018 IEEE Region 10 Conference*. 2018, pp. 0695–0700. DOI: 10.1109/TENCON.2018.8650152 (cit. on pp. 26, 27).

[ZCW+18] M. Zhang, Z. Chen, H. Wang, Z. Zeng, X. Shan. "Research on Database Failure Prediction Based on Deep Learning Model". In: *IOP Conference Series: Materials Science and Engineering* 452 (Dec. 2018), p. 032056. DOI: 10.1088/1757-899X/452/3/032056 (cit. on p. 26).

[ZGFC11] X. Zhao, Y. Guo, Q. Feng, X. Chen. "A System Context-Aware Approach for Battery Lifetime Prediction in Smart Phones". In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. SAC '11. TaiChung, Taiwan: Association for Computing Machinery, 2011, pp. 641–646. ISBN: 9781450301138. DOI: 10.1145/1982185.1982327. URL: https://doi.org/10.1145/1982185.1982327 (cit. on p. 24).

[Zha04] T. Zhang. "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 116. ISBN: 1581138385. DOI: 10.1145/1015330.1015332. URL: https://doi.org/10.1145/1015330.1015332 (cit. on p. 43).

[ZKT+17] P. Zhao, M. Kurihara, J. Tanaka, T. Noda, S. Chikuma, T. Suzuki. "Advanced correlation-based anomaly detection method for predictive maintenance". In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2017, pp. 78–83. DOI: 10.1109/ICPHM.2017.7998309 (cit. on p. 22).

[ZLM+18] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang, Y. Chen, H. Dong, X. Qu, L. Song. "PreFix: Switch Failure Prediction in Datacenter Networks". In: *Proc. ACM Meas. Anal. Comput. Syst.* 2.1 (Apr. 2018). DOI: 10.1145/3179405. URL: https://doi.org/10.1145/3179405 (cit. on p. 25).

[ZW19] M. Zhang, D. Wang. "Machine Learning Based Alarm Analysis and Failure Forecast in Optical Networks". In: *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*. 2019, pp. 1–3. DOI: 10.23919/PS.2019.8817991 (cit. on p. 25).

[ZWZ+19] C. Zhang, M. Wang, M. Zhang, D. Wang, C. Song, L. Guan, Z. Liu. “Adaptive Failure Prediction Using Long Short-term Memory in Optical Network”. In: *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*. 2019, pp. 1–3. DOI: 10.23919/PS.2019.8817702 (cit. on p. 25).

[ZZY+20] H. Zhuang, Y. Zhao, X. Yu, Y. Li, Y. Wang, J. Zhang. “Machine-Learning-based Alarm Prediction with GANs-based Self-Optimizing Data Augmentation in Large-Scale Optical Transport Networks”. In: *2020 International Conference on Computing, Networking and Communications (ICNC)*. 2020, pp. 294–298. DOI: 10.1109/ICNC47757.2020.9049750 (cit. on p. 25).

All links were last followed on July 6, 2021.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature