POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's Degree in ICT for smart societies

Master's Degree Thesis

A maintenance prediction system development: analysis and integration

Supervisor

Prof. Maurizio ZAMBONI

Candidate

Bruno VALENTE

October 2022

Abstract

The Internet of Things enables us to improve operational efficiency and enjoy better lives. It also helps the industrial sector work more effectively and under complete control. The way that things and devices communicate with one another and with people is changing as more and more of them are connected to the Internet. Every day, enormous amounts of data are gathered and transported through networks, giving rise to the idea of Big Data. Companies are increasingly using data mining techniques to extract knowledge from large, complex, and varied data sets in order to reap benefits.

One of these benefits embodies the central idea of this thesis: assisting a company in determining when industrial machines require routine maintenance. The most recent investigation revealed that there has been a lot of research on subjects connected to predictive maintenance during the past few years. The ability to anticipate maintenance improves job scheduling efficiency and results in significant time and money savings.

The effectiveness of a few popular machine learning techniques has been examined and compared. A preliminary classification to assess whether the machine is approaching the next maintenance followed by a regression to actually make the prediction in terms of days until maintenance has been used to break down the forecasting problem into two simpler tasks. Basic machine learning approaches have been contrasted with more sophisticated ones, like ensemble methods. Some of these algorithms have produced good accuracy metrics, providing a strong basis for further research and development.

The suggestion of a potential approach to integrating the prediction system in a more complicated environment is another objective of this thesis. Widely used enterprise technologies have been employed to deliver comprehensive and practical solutions. With a thorough methodological definition, the structure of the system and how its components interact with one another have been described, along with potential difficulties that can develop while tackling this kind of issue and how to overcome them.

Table of Contents

List of Tables

List of Figures

Acronyms

IoT

Internet of Things

IT

Information technology

IIoT

Industrial Internet of Things

M2M

Machine to machine

IoE

Internet of Everything

ADAS

Advanced Driver-Assistance Systems

V2V

Vehicle-to-vehicle

V2I

Vehicle-to-infrastructure

KDD

Knowledge discovery in databases

AI

Artificial intelligence

ML

Machine learning

COSMO

Consensus self-organized models

CNN

Convolutional neural network

AR

Auto regressive

MA

Moving average

ANN

Artificial neural network

SVM

Support vector machine

SVR

Support vector regression

MIMO

Multiple-input multiple-output

EJB

Enterprise JavaBean

POJO

Plain old Java object

RM

Reactive Maintenance

PdM

Predictive Maintenance

PM

Preventive Maintenance

LR

Linear regression

RF

Random forest

XGB

eXtreme Gradient Boosting

RBF

Radial basis function

k-NN

k-nearest neighbors

ORDBMS

Oject–relational database management system

MVC

Model View Controller

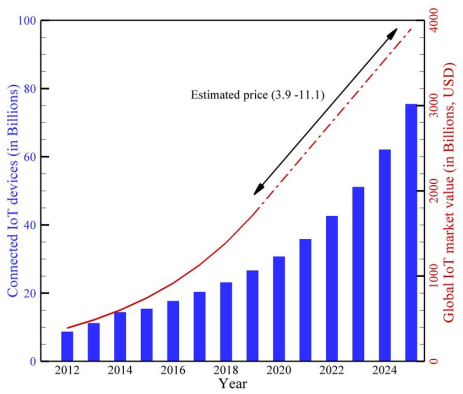
Chapter 1

Introduction

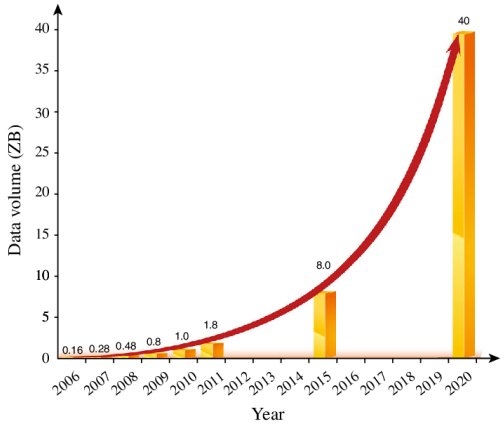
1.1 A technological revolution

Internet of Things The year was 1982, when a group of researchers at Carnegie Mellon University connected a vending machine to the Internet, allowing them to check if there were cold sodas available in the machine, before going to purchase one. This turned out to be one of the first non-computer objects to be connected to the Internet, many years before the definition of the Internet of Things (IoT) concept [1].
The phrase "Internet of things", indeed, was used for the first time as the title of a presentation Kevin Ashton made while working at Procter & Gamble, in 1999 [2]. Although, it took at least another decade for the technology to catch up with Ashton's vision; in fact, in a Cisco Systems' white paper it is estimated that the IoT was actually born sometime between 2008 and 2009, when the number of connected devices exceeded the world population [3]. Nowadays, the IoT is one of the most disruptive trends, and it is almost impossible that someone never heard of this term. Objects like fridges, light bulbs, watches, vehicles, etc. are now connected to the Internet, and the variety of connected objects is growing so much that Cisco started to speak about the Internet of Everything (IoE).

During the late 20th century, electronics and information technology (IT) began to be largely used to automate the manufacturing processes and a shift from analog to digital domain started. On the basis of this industrial revolution, often called the



(a) Connected IoT devices and relative market value over time [4].



(b) Data volumes over time [6].

Figure 1.1: Number of IoT devices and data volumes over years.

Digital Revolution, today the world is experiencing a Fourth Industrial Revolution (also known as Industry 4.0). So, even the industrial field is interested in a series of new technologies and capabilities derived from recent technical inventions and communication paradigms. For example, we can now speak of the Industrial Internet of Things (IIoT), where all the machines involved in manufacturing are online and can interact with each other (M2M communication) and with humans.

Data mining In [3], it was predicted that around 50 billion IoT devices would have been connected to the Internet by 2020. More recent research studies ([4], [5]) evidenced that the actual number of IoT devices in 2020 was not so high, but it is still considerable – around 30 billion, as shown in Figure 1.1a.
Even assuming that each device daily produces and ingests an amount of data which is in the order of a few megabytes, a huge overall volume of data is easily reached worldwide. Moreover, it is important to consider the rate at which the number of IoT devices able to go online grows, and accordingly the data they produce. In Figure 1.1 the exponential growth of both connected IoT devices and data volumes over time is clearly visible.

We live in a world where not only the amount but also the variety of data collected and stored daily is overwhelming – scientific data, medical data, financial data, marketing data – and analyzing them is a necessity to draw useful knowledge

and information [7]. Research in statistics, visualization, artificial intelligence, and machine learning are helping find new ways to automatically analyze and classify this vast quantity of raw data. Data mining emerged during the late 1980s and continues to develop and evolve into the new millennium. It consists of the automatic discovery of structure and patterns in large and complex data repositories [8]. Treating data mining as an equivalent of knowledge discovery from data (KDD) is the main trend among many people – including the industry, the media, and the research field – but not the only one [7]. For some people, data mining is an essential step in a wider process, going from data preprocessing to postprocessing of data mining results. Data mining tasks are generally classified into two major categories: descriptive and predictive [9]. Descriptive mining tasks are usually exploratory and aim to describe patterns and properties of the data in order to identify underlying relationships among them. Predictive mining tasks perform induction on the current data in order to predict the outcome of a future observation.

Data mining is interrelated with Artificial Intelligence (AI) and one of its branches: machine learning [10]. To be considered intelligent, a system placed in a changing environment should have the ability to learn. Machine learning (ML) consists in designing efficient algorithms to teach machines to use experience – all the past information available – to improve their performance automatically [11]. This phase is usually known as *training* of a machine learning model.
The main practical objectives of machine learning are focused on accurately predicting unseen items and originating effective algorithms to produce these predictions. The greatest the quantity and quality of data collected, the better will the accuracy of the prediction be.

1.2 Objective and challenges

In this section, we are going to investigate in further detail which are the main objective and the challenges of this thesis.

The objective of this thesis can be summarized in a simple statement: design a service that is able to predict when generic industrial machine maintenance is due, and propose some methods to integrate it into an existing industrial environment.

Let us focus, for now, on the first part of the statement, i.e. the prediction problem. The first question that arises is: how can we predict something that will happen in the future? As introduced in Chapter 1.1, data can hide useful knowledge from which it is possible to extract patterns; so, by analyzing past behaviors, one can estimate how and when a specific event will occur.
Answering this first question, we alluded to the necessity of knowing something about the past. This something consists of data collection. Speaking about industrial machines, meaningful data can be represented by usage hours, voltages, tire pressure, power consumption, etc. A collection of this kind of parameters indexed in time order, with a specified granularity (daily, hourly, and so on), is called *time series*. At this point, from the previous question, another one can naturally arise: who will provide us with the data? The data used to train and test the algorithms described in the thesis will be synthetic data. The application domain is for sure the Internet of Things and the amount of data can allow many data mining applications, including the achievement of our objective.

Now, we can focus on the second part of the objective statement, i.e. integrate the prediction service into a company complex solution. It is a classical engineering problem (very common in the IT field) of system integration, that arises each time a new service has to communicate with other pieces of software. The main challenges involved in this process of adding a sub-system to a large software solution are described at the end of this section. Many are the benefits that a good integration strategy can bring.

To summarize, the main challenges of the project can be divided into two macro-categories: the ones belonging to the machine learning field and the ones related to system integration.
In the first class, it is possible to define the following challenges:

- Preliminary models comparison: an initial subset of machine learning algorithms should be opportunely designed, in order to compare their prediction performances on some datasets.
- Analysis on a large set of machines: the analysis described at the previous

point should be conducted on a considerable number of machines, in order to better generalize the results obtained.

- Obtain good performances close to maintenance: maintenance technicians are particularly interested in knowing how many days are left for the next maintenance, especially when we get closer to it.

- Design machine-specific prediction models: try as much as possible to have an ML model designed on purpose for each machine, and find other strategies for machines that do not have enough past data to train their own model.

- Take into account the execution time of the training procedure: considering the previous point, a large number of models might be trained, so it is important to analyze also their training time.

The main challenges related to the system integration area can be described as follows:

- Comply with the technologies used by a company: it is needed to make the code compliant with the programming languages, architectural structure, and communication protocols used to build the existing blocks and services. This would not only improve the coupling among the services but also guarantee readability and understanding by other developers, allowing future developments.

- Training automation: the system should be able to "understand" on its own when a model can be trained again and updated because new data are available.

- Efficient database design: since we spoke about data used to train machine learning algorithms, an efficient design of the database and its tables is required in order to minimize the time of reading and writing operations.

- Achieve scalability and modularity of the system: the system should be scalable and handle a growing number of machines if more resources are added to the system. Designing a modular system would enhance flexibility, making the code more easily maintainable.

1.3 Fields of application

As already introduced in Chapter 1.1, the domains of interest of this thesis are the IoT, and more specifically the IIoT.
The term IIoT refers to every connected device used in the energetic, manufacturing, and industrial sectors. It can offer multiple advantages, that allow increasing the levels of automation and machine monitoring, in order to improve reliability and efficiency. Data coming from sensors installed on industrial equipment can be collected and analyzed in real-time, enhancing the whole production and control processes of a company.

One of the most interesting goals of the IIoT, which is particularly linked to the work of this thesis, consists of optimizing the production process by processing the data coming from industrial machines. It can lead to several benefits in terms of predictive analysis, which leads to substantial savings of money for maintenance and failures.

Finally, it is worth citing some IoT applications to the automotive field. The first thing that can come to mind, combining vehicles to IT, could be self-driving cars. Besides that trendy topic, several other applications exist, enhancing convenience, safety, and efficiency. Some of them are [12]: Advanced Driver-Assistance Systems (ADAS), systems that provide useful information to the driver improving driving performance and road safety; Connected Vehicles: which refers to all the applications, services, and technologies that connect a vehicle to its surroundings through vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications.
Several other applications can be found, such as fleet management solutions, going from optimizing the maintenance and logistics of fleets to monitoring drivers' and vehicles' performances through real-time vehicle telematics.

Speaking about industrial vehicles, a lot of IT companies provide solutions to make the machines "smart".
Usually, a large amount of data can be easily collected from a set of heterogeneous vehicles, machinery, and objects, in order to be stored, processed, and managed. This task is accomplished by installing an onboard device on the assets: different devices have to be produced to suit any type of vehicle, from surveying to construction, from agriculture to automotive. Data about the assets' activity (e.g. location,

movement, oil pressure, fuel consumption, etc.) are collected and processed to get meaningful and readable charts. Most solutions allow the users to check the analytics in real-time, through web platforms and/or mobile applications.

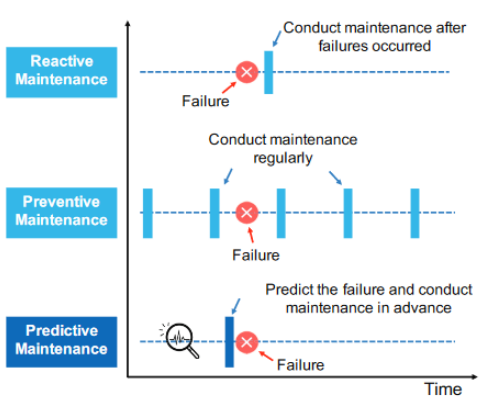
1.4 Predictive vs Preventive vs Reactive Maintenance

Nowadays, plant management in manufacturing is still governed, in most cases, by traditional preventive manifestation systems. The initial assumption is that by regularly maintaining machinery, the risks of malfunctions are reduced. Predictive maintenance overcomes this paradigm in favor of a system in which maintenance is managed according to the actual state of equipment, supplies, and machinery through a system of monitoring, analysis, and prediction.

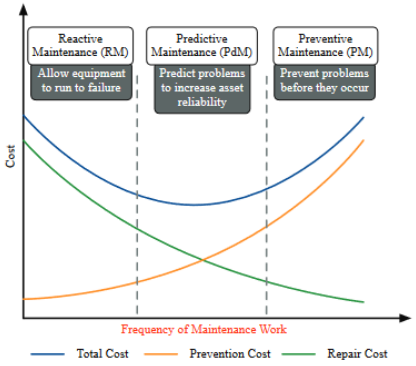
Compared to reactive maintenance (RM), in which repairs are carried out only after a malfunction or failure has occurred, or preventive or scheduled maintenance, in which interventions are planned against predefined intervals dictated by the technical usage specifications of component consumption, predictive maintenance (PdM) is a methodology that uses historical data from monitoring phases to track equipment performance during normal operation and to detect any anomalies and resolve them before they give rise to failures.

By pushing the machinery to its limits, RM achieves maximum equipment utilization and, as a result, maximum production output. When utilizing run-to-failure management, a business waits to invest in maintenance until equipment or system breaks down. However, it is possible that the cost of fixing or replacing a component might exceed the production value gained from operating it until the breakdown. Additionally, if parts start to shake, overheat, and fail, equipment damage might continue, possibly necessitating more expensive repairs. A business should also keep a large supply of spare parts for all essential machinery and components to be prepared for any breakdowns. [13]

On the other hand, utilizing a system or component's real working state is the basis of PdM. The predictive analysis is based on information gathered from meters



(a) Maintenance plans of RM, PM and PdM. [13]



(b) Comparison of RM, PM and PdM on the cost and frequency of maintenance work. [13]

Figure 1.2: Predictive vs Preventive vs Reactive Maintenance.

and sensors attached to tools and machines, including vibration data, thermal imaging, ultrasonic data, operation availability, etc. The predictive model uses predictive algorithms to evaluate the data, identify trends, and forecast when equipment will need to be fixed or replaced. By performing maintenance tasks only when absolutely essential, PdM helps businesses optimize their strategies rather than using equipment or components until they break or replace them when they still have a useful life. PdM can reduce unplanned and scheduled downtime, high maintenance costs, superfluous inventory, and unneeded maintenance on operating equipment. The price of the condition monitoring equipment required for PdM is frequently higher than that of RM and PM, though. Additionally, as a result of data gathering, analysis, and decision-making, the PdM system is getting more and more sophisticated. [13]

In order to reduce the likelihood of failures, preventive maintenance (PM) plans routine maintenance procedures for specified equipment. To prevent unexpected malfunctions with the resulting downtime and costs, maintenance is performed even when the machine is still operational and functioning normally. PM could prevent unplanned downtime and repair expenses, but it might also cause catastrophic failures or unneeded repairs. Instead of using actual statistics on the condition of the particular piece of equipment, the theoretical rate of failure is used to predict

when an item of equipment will reach the wear-out phase. This frequently leads to expensive and wholly useless maintenance procedures being performed either before a real issue arises or once the possibly catastrophic harm has started. [13]

We conclude by summarizing the distinctions between the three categories of maintenance techniques in terms of costs, advantages, difficulties, and applications that are appropriate and inappropriate. First, we include in 1.2a the maintenance schedules for RM, PM, and PdM. Additionally, in 1.2b, we contrast the price of these three maintenance procedures. As a result of adopting run-to-failure management, RM has the lowest prevention cost; PM has the lowest repair cost, and PdM may achieve the best balance between repair cost and prevention cost. To prevent unplanned RM, PdM ideally enables the maintenance frequency to be as low as feasible without paying the costs associated with performing excessive PM. Note that the repair cost refers to the corrective replacement cost once a failure has occurred, whereas the prevention cost primarily includes inspection cost, preventative replacement cost, etc.

1.5 Thesis outline

A short summary of all the thesis chapters is here provided:

In Chapter 1 a general introduction is given. We spoke about the domains of interest in which this project is collocated: the Internet of Things and Data mining. A short overview of the possible fields of application involved in the thesis and a comparison among Predictive (PdM), Preventive (PM), and Reactive Maintenance (RM) are also provided. Finally, the objective is defined and so are the challenges that arise during its achievement.

Chapter 2 provides the state of the art about some topics related to this thesis development. Scientific papers and books discussing predictive maintenance, time series forecasting, and IT management are reviewed, reporting some useful information for the development of the thesis.

Chapter 3 covers the conceptual formulation of the problem, its mathematical representation, how we altered it to fit a machine learning challenge, and the algorithms we used to solve it.

The problem formulation will be presented first. The applied regression and classification algorithms will next be examined mathematically and analytically, along with comparisons and some advantages and disadvantages of using them.

Chapter 4 presents an overview of the data used in the thesis and how they were generated, taking into consideration all the features that artificially created data should have. The two main tables of the database are described: a register that keeps track of the equipment's basic information and the usage dataset containing the information needed by the machine learning algorithms.

In Chapter 5, using the approaches outlined in Chapter 3, we offer the results and a few outputs from the dataset that was studied in Chapter 4. A brief summary of the most popular metrics used to assess the performance of classifiers and regressors is offered because the main focus of this chapter will be an investigation of machine learning methods. Results for both the classification and regression algorithms' hyperparameter adjustment are shown, and they are discussed. The various algorithms are then examined using graphs, charts, and performance indicators.

Chapter 6 simply draws the final conclusions and insights from the work done. Some future developments and improvements are also proposed and examined.

Chapter 2

State of the art

In this chapter, we are going to perform a literature review and state-of-the-art analysis of some important topics involved in the development of this thesis. The three analyzed areas are: predictive maintenance with a special focus on the industrial environment; generic methods for time series forecasting; IT management and integration of a simple service into a more complex system.

2.1 Predictive maintenance

Since the maintenance strategies should be developed so that the maintenance operations are carried out at the appropriate times in order to prevent breakdown, the following work is focused on Predictive maintenance (PdM).

The method proposed in this thesis cannot be defined as a predictive maintenance system in a very real sense, because it is only based on the past usage of machines and does not use information collected by a set of sensors installed on them.

A heterogeneous set of scientific papers about predictive maintenance from a different point of view has been selected.

- Through the article [15], the possibility of using predictive maintenance (PdM) to optimize plant operations and decrease system downtime, which will result

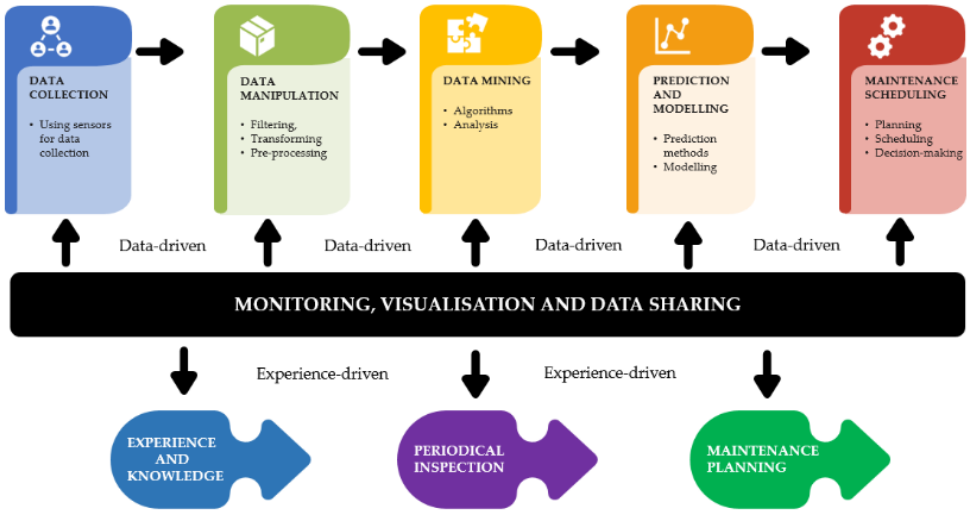


Figure 2.1: Experience- and data-driven predictive maintenance. [14]

in lower production costs, has been explored. The authors propose a predictive maintenance architecture: a sensor node was developed and the parameters commonly required in industry to monitor were uploaded to the cloud; further steps include using the data in the database and developing machine learning algorithms in order to develop a predictive analysis and elect appropriate maintenance task and schedule them.

- In paper [16] a multiple classifier machine learning methodology for Predictive Maintenance (PdM) is presented. The suggested PdM methodology works with high-dimensional and censored data problems and enables the use of dynamical decision rules for maintenance management. This is accomplished by using several classification modules that have been trained with various prediction horizons to provide various performance trade-offs in terms of the frequency of unexpected breaks and unutilized lifetime and then applying this knowledge in an operating cost-based maintenance decision system to reduce anticipated costs.

- In the article [14], Farooq et al. make a distinction between experience-driven and data-driven maintenance. Experience-driven preventative maintenance

bases its maintenance planning on accumulated information about manufacturing equipment. Data-driven preventative maintenance, on the other hand, is based on analyzing a lot of data. The difference is presented in Figure 2.1. This strategy is appropriate for Industry 4.0 settings since it is built on artificial intelligence, specifically machine learning and statistical modeling.

- Authors of [17] analyzed a popular fault detection method, the Consensus self-organized models approach (COSMO), applied to a fleet of buses. They proposed an IoT architecture for predictive maintenance, based on a semi-supervised machine learning algorithm with the aim of improving the sensor feature selection performed in COSMO. A prototype of the proposed architecture has been installed on some test vehicles, with the intent to collect data to continue data analytic research.

- In [18], a system to find the root cause of malfunctioning in vehicles is presented. The authors tested different hardware/software solutions and ML techniques, including convolutional neural network (CNN). They designed a method to define what data to collect and when, in order to perform a root cause diagnostic based on time series analysis.

2.2 Time series forecasting

Predicting the future by means of time series analysis is something that humans have been trying to achieve for a long time, way before the advent of computers. Anyway, one of the first formalizations came out in 1970, with the first publication of the book "Time Series Analysis" by Box and Jenkins [19], which contains a detailed modeling procedure, going from the time series specification to the forecasting problem. Box-Jenkins models represent the base of most forecasting techniques widely used nowadays, known as Autoregressive (AR) and Moving Average (MA) models.
Many research projects about time series forecasting, especially in financial, electric, and energetic domains, can be found in the literature. A collection of some of them is reviewed hereafter:

- A review of the existing machine learning techniques for forecasting buildings'

energy consumption is presented in [20]. Since energy historic data is often related to other time series – e.g. outdoor weather or environmental conditions, the study is not limited to the analysis of the single energetic consumption time series. Performances of nine popular forecasting methods and some hybrid models, obtained by combining two or more techniques, are reported.

- Paper [21] proposes a combination of random walk and artificial neural network (ANN) for financial time series forecasting, to counteract two key problems in financial data – noise and non-stationarity. Their approach consists of two steps: the linear part of the financial time series is processed by the random walk model, and the non-linear residuals are processed by feed-forward ANN and Elman ANN models. The results show that the proposed hybrid method outperforms each of the isolated methods.

- Authors of [22] try to improve the accuracy of long-future horizon forecasting. They propose a novel multiple-step-ahead time series forecasting approach, employing multiple-output support vector regression (M-SVR) with multiple-input multiple-output (MIMO) prediction strategy. The results demonstrate that their solution achieves better performances and computational loads when compared to standard SVR.

2.3 Information technology management

Being a project of integration of service into an existing company web solution, the IT management is a crucial aspect that is worth examining.
Integration is a word that could have many meanings. However, it is a universally accepted need in the development of complex engineering systems. It derives from the decomposition of a complex problem into several simpler problems, which can be more easily solved by teams of specialists. For this reason, a well-defined system integration process facilitates the connections of the separate solutions into a whole system. [23]

These are the same concepts applied by a relatively recent and emerging software architecture pattern: the micro-services approach. It enables the development of an application as a set of small, independent, and loosely coupled services. This

approach is opposite to the so-called monolithic architecture: a single program containing multiple functions, but deployed as a united solution.
Some of the numerous benefits the micro-services approach brings are [24]:

- Understanding and modifying a small service is easier than working on a huge application. Moreover maintaining and updating a big monolithic application could become very difficult.
- It allows the development of each service in parallel by different independent teams.
- Each micro-service can be implemented with its own set of technologies and programming languages, without impacting other services.
- The scalability of the system can be improved because micros-services run in their own independent processes.

A common programming framework used by a lot of companies is the Spring Framework[1] for the Java platform. So, we will spend some words on this widespread enterprise framework.
A recent version of this framework is called *Spring Boot* and aims to simplify Spring development. For this reason, a portion of our service will be based on this framework.
Since it is a very complex framework, it would be impossible to describe every aspect of it. Some basic information hereafter reported is taken from [25].

Before Spring, Java Enterprise Edition represented the de-facto standard in enterprise Java programming. The first aim of Spring was to offer a lightweight alternative to Java Enterprise Edition, substituting heavyweight components as Enterprise JavaBeans (EJBs) with plain old Java objects (POJO) and utilizing dependency injection and aspect-oriented programming.
In the beginning, Spring offered this lightweight solution, but with an important drawback: it was quite difficult to be configured. Spring Boot highly simplifies this step, allowing for the reduction of the development friction of configuration.

[1] https://spring.io/

One of the most common programming languages used by data scientists for machine learning and data mining applications is Python[2].
For this reason, all the machine learning algorithms described later are designed in Python.
Here is a list with some descriptions of the Python libraries used for this thesis:

Pandas is a library for data manipulation and analysis. It includes specific data structures and procedures for working with time series and mathematical tables.

NumPy adds support for large matrices and multidimensional arrays along with a large collection of high-level mathematical functions to operate efficiently on these data structures.

Matplotlib is a library for creating graphs and plots.

Scikit-learn is a machine learning library. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, etc.

Another technology that is worth introducing is Node-RED[3] since it has been employed in order to develop a dashboard for providing a user interface.
Node-RED is a flow-based visual programming tool that was initially created by IBM for the Internet of Things to connect hardware components, APIs, and web services.
An online flow editor powered by Node-RED is available for developing JavaScript functions. Applications' components can be shared or saved for later use.

As concerns data storage, a PostgreSQL[4] database has been employed. PostgreSQL, also simply called Postgres, is an object-relational database system (ORDBMS) that is open source and free. Its main features are reliability, data integrity, functionality, and extensibility. For querying and interacting with data, Postgres uses a subset language of SQL.

[2]https://www.python.org/

[3]https://nodered.org/

[4]https://www.postgresql.org/

Chapter 3

Methodology

In this chapter, we discuss the conceptual formulation of the problem, its mathematical representation, how we modified it to match a machine learning challenge, and the algorithms we employed to solve it.

First of all, the global structure of the system will be presented. Then, the problem formulation is discussed, with a particular focus on the forecasting problem resolved using classification and regression algorithms. At the end of the chapter, the regression and classification algorithms employed will be analyzed from a mathematical and analytical point of view, providing some comparisons among them and some pros and cons of using each of them.

3.1 Global structure

As we introduced in Section 1.2 when discussing the objectives and the challenges that arose in this work, we want to design a system that is able to predict when generic industrial machine maintenance is due, and propose some methods to integrate it into an existing industrial environment.

A sort of microservices approach has been chosen in order to improve the scalability, modularity, and maintenance of the system. Each block is independent, loosely coupled to the others, and has been developed using different programming languages and technologies.

Figure 3.1 shows how the blocks of the system are built and presents how they

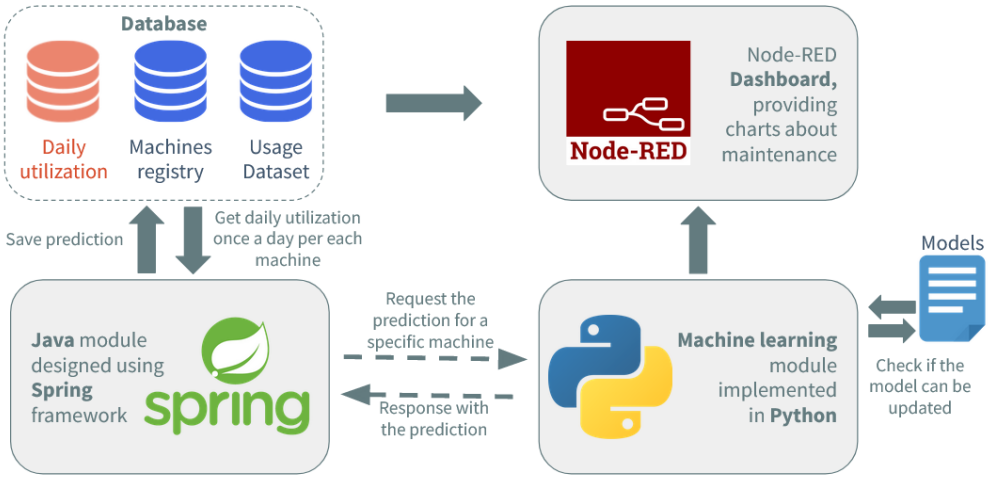


Figure 3.1: Structure of the project.

are interconnected among them.
The first block represents all the databases involved in the system: an artificial external database in which the daily utilization is retrieved, representing an external service, and two internal databases; the first one where the information of each machine is stored (a machine registry) and the second where the dataset for the machine learning algorithms is built and stored. All the tables containing the data are stored in a PostgreSQL database.

A second block is a dashboard that has been developed using Node-RED. It provides a graphical user interface with plots and user-friendly information about the prediction, the last update of the model, etc.

The micro-service of the project which is coded using Spring Boot is a sort of coordinator. It handles the communications between the external world and the service itself. Once a day gets the new parameters from the databases and saves the new predictions. The predictions are obtained by interrogating the last block.

The last block is the machine learning module which has been coded in Python. It is where the predictions are made. First of all, it checks if new data can be retrieved to train a new model, then predict the new maintenance prediction and send it to the Spring module previously described. Since very often, companies are interested in having good accuracy when we are near maintenance, the machine learning module performs two steps. Firstly a classifier predicts if we are close to

the maintenance or not. Accordingly, the actual prediction is performed using a regression model trained on the whole dataset or only on the days close to the maintenance.

3.2 Prediction problem formulation

Our aim is to forecast the next maintenance operation for a given industrial machine m. Let T_m be the permitted usage times (in seconds) for m between two consecutive maintenance operations, and let N_m be the number of days for which historical data about m usage is accessible.

A cycle will now be used to refer to the interval between maintenance procedures, so T_m will represent the cycle duration for the machine m. The countdown to the next maintenance on m changes day by day. Consider the series of the aforementioned daily counts as $D_m(t)$. We want to forecast this value, where t stands for the current day. The series of utilization left to the next maintenance, $L_m(t)$, can be evaluated this way:

$$L_m(t) = T_m - \sum_{i=t-C_m(t)}^{t-1} U_m(i)$$

where $U_m(t)$ is the time series of the daily usage in seconds and $C_m(t)$ represents the days passed from the last maintenance.

We create a relational dataset including the historical utilization series U(i) for each machine m. To be more precise, each record represents a distinct day t and is made up of a number of variables that represent the previous use in seconds. The features include the values of $U_m(i)$ in a time window $[t - W \leq i \leq t - 1]$, where W is the window size. Other features are the current time until the following maintenance, denoted by $L_m(t)$, and the target variable, denoted by the number of days till maintenance, denoted by $D_m(t)$. Only for the classifier, we add another feature: a binary flag, $F[D_m(t)]$ which is $True$ if $D_m(t) > 30$ and $False$ otherwise. This variable will be used by the classifier to decide whether or not we are close to maintenance. In particular, the maintenance is reputed close when less than 30 days remain.

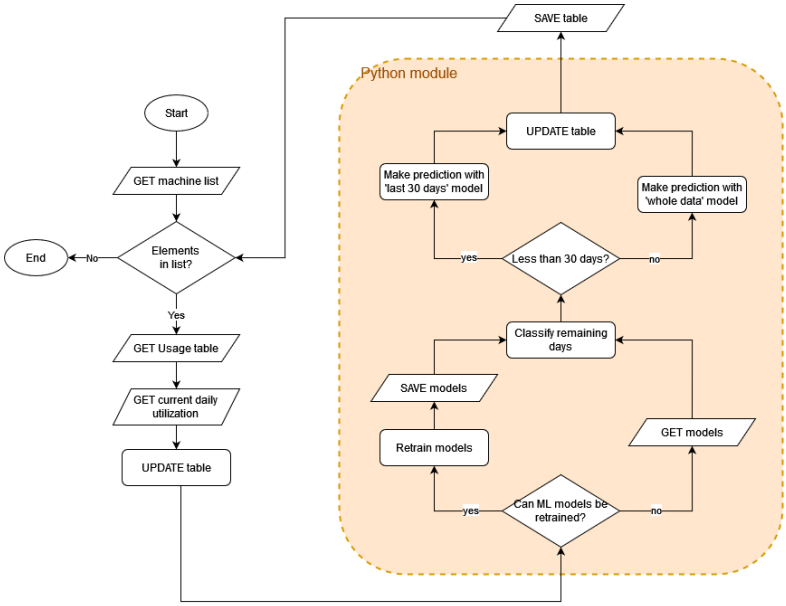


Figure 3.2: The workflow that leads to the prediction.

Now, it is important to state a difference regarding the dataset of the regression algorithm, because actually, we will have two versions of the predictor for each machine: one will be trained on the whole dataset and will be used for making predictions when the classifier predicts we are far from maintenance (more than 30 days are left), the other will be trained only on the last 30 days of each cycle in the dataset and will be employed when the classifier predicts the machine is near the next maintenance. This solution, as we will see in the results in Chapter 5, improves the prediction quality when the maintenance is close and high performance would be important for scheduling purposes.

Figure 3.2 highlights how the process that leads to the prediction goes. The "handler" of the system (the Spring Boot piece of the application) makes some requests to the machine learning module (coded in Python, which can be consulted in Appendix A.3). It is here that the above-described classification and the related regression are executed.

3.3 Classification algorithms

As said in the previous section, first of all, we have to decide whether or not the equipment is close to maintenance. This is achieved using a classifier that has to decide if the days remaining to the maintenance are less or more than a predefined threshold.
In particular, the four analyzed classification algorithms are: a simple Naive Bayes classifier, k-nearest neighbors (k-NN), a Support Vector Machine (SVM), Random Forest classifier (RF) and AdaBoost (short for Adaptive Boosting).

3.3.1 Naive Bayes classifier

A group of classification algorithms built on Bayes' Theorem is known as naive Bayes classifiers. It is a family of algorithms rather than a single algorithm, and they all conform to the same basic idea. They are some of the most basic Bayesian network models, but when combined with kernel density estimation, they could achieve high levels of accuracy.

The Bayes theorem which acts as the basis of the classifier can be summarized as follow [26]:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

where, y is the class variable and $X = (x_1, x_2, x_3, ..., x_n)$ is a dependent feature vector.
By replacing X and expanding using the chain rule, we obtain:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, we may eliminate the denominator and introduce a proportionality since, for each given input, that term is constant:

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Finding the class $\hat{y}$ with the highest probability is the final goal:

$$\hat{y} = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

3.3.2 K-nearest neighbors

The k-nearest neighbors, abbreviated as k-NN, is an algorithm used in pattern recognition for classifying objects based on the characteristics of objects close to the one being considered. The result of the k-NN classification is a class membership. An object is classified by a plurality vote of its neighbors, with the object being allocated to the class with the most members among its k closest neighbors, where k is a positive integer, typically small. The object is merely put into the class of its one nearest neighbor if k = 1; in this case, the method makes no error on the training set, but it generalized poorly to novel test data and could result in overfitting.
Increasing k and tuning its value provides for sure more interesting results.

3.3.3 Support Vector Machine

A Support Vector Machine (SVM) is a supervised learning algorithm, used in many classification and regression problems. In case of regression problems, it is usually called support vector regression (SVR).
The goal of an SVM algorithm is to find a hyperplane that separates, to the best degree possible, the data points of one class from those of another class. "Best" means the hyperplane that has the greatest margin between the two classes. Margin means the maximum width of the line parallel to the hyperplane that has no internal data points. The algorithm is only able to find such a hyperplane for linearly separable problems, while for more practical problems the algorithm maximizes the soft margin, which allows for a reduced number of misclassifications.

A mathematical formulation of the method is presented for the respective regression problem in Section 3.4.2.

3.3.4 Random forest classifier

Random forest is a versatile, user-friendly machine learning approach that typically yields excellent results even without hyper-parameter adjustment. As a result of its versatility and simplicity, it is also one of the most widely used algorithms (it can be used for both classification and regression tasks).

An ensemble of decision trees, typically trained using the "bagging" approach, make up the "forest" that it constructs. The bagging method's general premise is that combining learning models improves the end outcome.

Simply put, a random forest creates many decision trees and integrates them to get a prediction that is more accurate and reliable.

The random forest has the key benefit of being applicable to both classification and regression problems, which make up the majority of modern machine-learning systems.

Additional information will be provided in Section 3.4.3, where the regression version of the random forest will be treated.

3.3.5 AdaBoost

Since it was first introduced by Freund and Schapire in 1997 [27], boosting is an ensemble modeling method that is frequently used to solve binary classification issues. By transforming a number of weak learners into strong learners, these methods increase prediction ability.
The basic idea behind boosting methods is that after creating a model using the training dataset, we create a second model to fix any mistakes in the original one. This process is repeated until the mistakes are reduced and the dataset can be accurately forecasted.

AdaBoost, also known as Adaptive Boosting, is an ensemble method used in machine learning. Decision trees with only one split, or those with one level, are the most frequent algorithm employed with AdaBoost. This algorithm creates a model while assigning each data point an equal weight. Then, it gives points that were incorrectly categorized as larger weights. The next model now gives more weight to all the points with higher importance. It will continue to train models until a smaller error is observed.

Following is the formula to determine the sample weights at the beginning:

$$w = 1/N \in [0,1]$$

where N is the overall number of observations in the dataset.
After that, we use the following formula to determine the classifier's actual influence in identifying the data points:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Where α_t expresses how much of an impact this stump will have on the final classification.
ϵ_t is just the total error; the sum of all incorrect classifications for that training set divided by the size of the training set.

The sample weights are updated by entering the actual ϵ_t values for each stump using the following formula:

$$w_i = w_{i-1} e^{\pm \alpha_t}$$

It means that the old sample weight will be multiplied by Euler's number, raised to plus or minus α_t, and the new sample weight will be equal to that result.
The sign of α_t depends on whether the predicted and the actual class agree $(+)$ or do not agree $(-)$.

3.4 Regression algorithms

In the following sections, a theoretical presentation of the regression algorithms used in this thesis for the maintenance forecasting problem is provided. In particular, the four analyzed regression algorithms are: a simple Linear Regression (LR), Support Vector Regression (SVR), Random Forest Regression (RF), and an improved version of the gradient boosting regression called eXtreme Gradient Boosting (XGB).

3.4.1 Linear regression

Linear regression analysis is used to predict the value of one variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you use to predict the value of the other variable is called the independent variable.
This form of analysis estimates the coefficients of the linear equation and involves one or more independent variables that best predict the value of the dependent variable. Linear regression corresponds to a straight line or surface that minimizes discrepancies between predicted and actual output values. There are simple linear regression calculators that use a method called "least squares" to find the optimal straight line for a paired data set. Then, the value of X (dependent variable) is calculated from Y (independent variable).

A general mathematical formulation of the problem can be described by the following formula:

$$y = Xw + \nu$$

Where y is the target variable vector ($Nx1$), X is the feature matrix (NxF), w is a vector of weights ($Fx1$) and ν is the vector $Nx1$ that identifies noise; being N the number of observations and F the number of features.
The Least Squares approach is used to achieve the objective of estimating the weights w, minimizing the square error:

$$e(w) = ||y - Xw||^2$$

3.4.2 Support vector regression

An approach for supervised learning called Support Vector Regression (SVR) is used to forecast discrete values. This is a regression algorithm derived from a Support Vector Machine (SVM) which examines data used for classification and regression analysis using supervised learning models and associated learning algorithms.

SVR uses the same principle of SVM to minimize an error, finding the best fit line called hyperplane. The SVR seeks to fit the best line within a threshold value, in contrast to other regression models that aim to reduce the error between the real

and predicted value. The distance between the boundary line and the hyperplane is the threshold value.

Instead of minimizing the squared error, like with the Least square method, SVR's objective function is to minimize the coefficients, or more precisely, the l2-norm of the coefficient vector.
The error term is dealt with in the constraints, where we set the absolute error less than or equal to a predetermined margin, known as the maximum error (ϵ). The maximum error ϵ can be adjusted to give our model the necessary level of precision. The following describes our new objective function that has to be minimized and constraints:

$$f(x) = \frac{1}{2}||w||^2$$

subject to:

$$|y_i - w_i x_i| \leq \epsilon$$

You might immediately realize that not all data points are compatible with this approach. Even though the algorithm did its best to solve the objective function, some of the points are still beyond the acceptable range. As a result, we must consider the potential of errors bigger than ϵ. Slack variables will help us with this. Slack variables have a straightforward concept: we may write every value that is outside of the range ϵ as having a margin deviation of ξ.

Although we are aware that these deviations could occur, we nonetheless want to do everything in our power to prevent them. So, these deviations can be added to the goal function.

$$f(x) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}|\xi_i|$$

subject to:

$$|y_i - w_i x_i| \leq \epsilon + |\xi_i|$$

Now that C has been added, we have to tune this other hyperparameter. We become more accepting of values outside of ϵ as C rises. The simplified equation, though occasionally impractical, emerges as C approaches zero and the tolerance gets closer to zero.

Since some regression problems cannot adequately be described using a linear

model, usually Nonlinear SVR is more suitable for this kind of problem. Nonlinear SVR is obtained by exploiting the so-called kernel trick. Kernel functions are able to map the points into higher dimensional spaces in which they become more easily separable.

In conclusion, the hyperparameters to be tuned in an SVR model are the following:

- The kernel: it specifies the kernel function used by the algorithm (e.g. linear, polynomial, Radial basis function (RBF), sigmoid)
- Degree: the degree of the polynomial kernel function. Ignored by all other kernels.
- The γ multiplier: it is a kernel coefficient for RBF, polynomial and sigmoid.
- The C parameter: the regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.
- ϵ: it specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.

3.4.3 Random forest regression

Here we are going to further discuss the random forest technique (already introduced in Section 3.3.4), with a particular focus on the regression version.
Random forest regression leverages the ensemble learning approach for regression. The ensemble learning method combines predictions from various machine learning algorithms to provide predictions that are more accurate than those from a single model. It is an averaging method, which means that it creates a number of weak learners (independent estimators) before averaging their predictions. During training, a Random Forest builds many decision trees and outputs the mean of the classes as the forecast of all the trees.

While random forests frequently outperform a single decision tree in terms of accuracy, they do so at the expense of decision trees' inherent interpretability. In

addition to linear models, rule-based models, and attention-based models, decision trees are part of a relatively limited family of machine learning models that are simple to understand. One of the most desirable characteristics of decision trees is their interpretability. It enables developers to verify that the model has drawn accurate conclusions from the data, and it enables end users to have faith and confidence in the model's judgments.

The hyperparameters of the random forest that have to be tuned are:

- The number of estimators: how many trees there are in the forest. Although it will take longer to compute, usually the larger the better. After a certain number of trees, the results will not improve considerably any further.
- Criterion (squared error, absolute error, Poisson): the function to measure the quality of a split. Squared error is equal to variance reduction as a feature selection criterion, absolute error for the mean absolute error, and Poisson which uses reduction in Poisson deviance to find splits.
- The maximum depth of the tree
- The minimum number of samples required to split an internal node
- The minimum number of samples required to be at a leaf node: this may have the effect of smoothing the model, especially in regression.
- The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.
- The number of features to consider when looking for the best split.

3.4.4 eXtreme Gradient Boosting

A class of ensemble machine learning methods known as gradient boosting can be applied to classification or regression predictive modeling issues.

Decision tree models are used to build ensembles. In order to repair the prediction mistakes caused by earlier models, trees are added one at a time to the ensemble and fitted. Boosting is a term used to describe this kind of ensemble machine learning model.

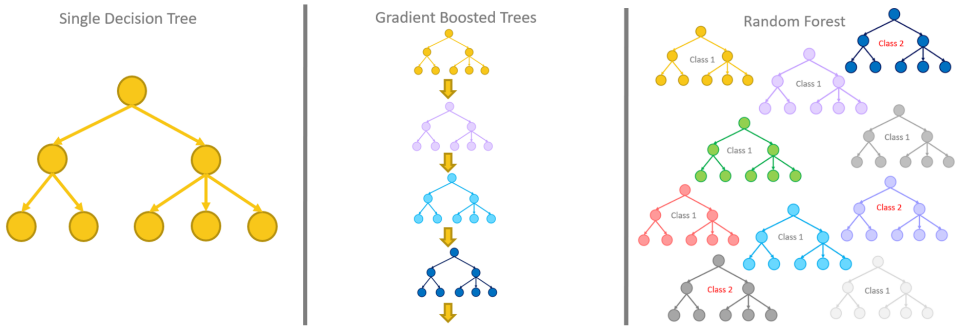


Figure 3.3: Difference between random forest and gradient boosting techniques.

EXtreme Gradient Boosting can be considered a tree-based algorithm too. While Random forest regression is an averaging method, eXtreme Gradient Boosting is a boosting method. Although more effective, eXtreme Gradient Boosting is similar to the gradient boosting architecture. It features both tree learning methods and linear model solvers. Therefore, its ability to perform parallel processing on a single machine is what makes it quick.

Figure 3.3 shows the differences between the boosting methods (such as gradient boosting) and the bagging methods (e.g. random forest): in the first technique, the output is obtained by combining multiple sequential simple regression trees into a stronger model; bagging models as random forest trains a number T of decision trees, each one on a different subset of observations and features, randomly selected with replacement at each iteration. The final model is then an ensemble of $T = 1, ..., N$ slightly differently trained decision trees.

Chapter 4

Data characterization

The big data process of characterization is used to create descriptive parameters that accurately characterize the properties and behavior of a specific piece of data. In this chapter, we are going to provide an overview of the data and a general characterization of it. First of all, it is important to specify that the data does not come from collections in a real-world environment, but are artificially generated.

Synthetic data produced by algorithms are utilized in model datasets for validation or training. In order to test or train machine learning models, synthetic data might simulate operational or production data.

Synthetic data has a number of key advantages, including the ability to generate large training datasets without the need for manual labeling of data and the reduction of restrictions associated with the use of regulated or sensitive data. Synthetic data can also be used to customize data to match circumstances that real data does not permit.

The data employed in this thesis can be divided into two tables belonging to a PostgreSQL database. The first one is a sort of register that, as the name suggests, is a list of machines (for example all the equipment owned by a company) and contains all the useful information for the machines. The second table contains the information needed for the machine learning algorithms to work, i.e. the daily usage time for each machine registered into the system and other important variables that will be discussed in the relative section.

4.1 The equipment registry

An important piece of the database is the one that will be referred to as *Equipment registry*. It contains basic information about the synthetic industrial machines involved in this work.

The fields for each machine are the following:

- Universally unique identifier (UUID). A 36-character string with dashes, digits, and letters, is intended to be universally unique in the system.
- Service ID. An identifier used to have more than one maintenance cycle for each machine.
- Cycle duration. An integer number representing the amount of time in seconds between two consecutive maintenance.
- Time zone. An integer from -12 to +14 represents the time zone where the machine is located.
- Total utilization. The total amount of time in hours the equipment has been used.
- Average utilization. It represents the daily average amount of time, in seconds, a piece of equipment has been used.
- Last update. It is the last date, in year-month-day format, when the prediction model has been updated.

Table 4.1 shows some records of the equipment registry. Each row represents a specific machine, with the exception of machines that have more than one maintenance service with different cycle duration (e.g. the equipment with `UUID` equal to df7f5...11701 has three different maintenance with cycle duration 150, 200 and 550 hours, hence three different rows characterized by `service_id` equal to 0, 1 and 2). From the machine learning algorithms' point of view, each row represents an entity, so, for example, a machine with three different maintenance services will be treated as three different machines.

It is clear that the primary key of the database is a composite primary key, made from the columns `UUID` and `service_id`.

UUID	service_id	cycle_dur	time_zone	tot_util	avg_util	last_up
0c217...db055	0	200	+8	7582	1422	2021-07-12
f0a64...ed234	0	350	-10	2566	3885	2021-11-05
df7f5...11701	0	150	+2	573	2351	2022-05-03
df7f5...11701	1	200	+2	573	2351	2022-05-13
df7f5...11701	2	550	+2	573	2351	2022-06-08
83ef6...045b7	0	500	-4	865	1278	2020-08-22
...	...	...	...	...	...	...
...	...	...	...	...	...	...
5bee2...0924a	0	300	-8	5524	6879	2022-04-25
5475f...a868a	0	300	+8	75	574	2022-03-03
de426...31c97	0	200	+11	548	1524	2021-06-21
2ad43...6bd89	0	300	+5	129	6512	2022-01-15
4d6be...7233a	0	250	+9	752	3289	2021-12-03
69eca...79daa	0	500	-6	6452	1528	2021-10-24

Table 4.1: Equipment registry samples.

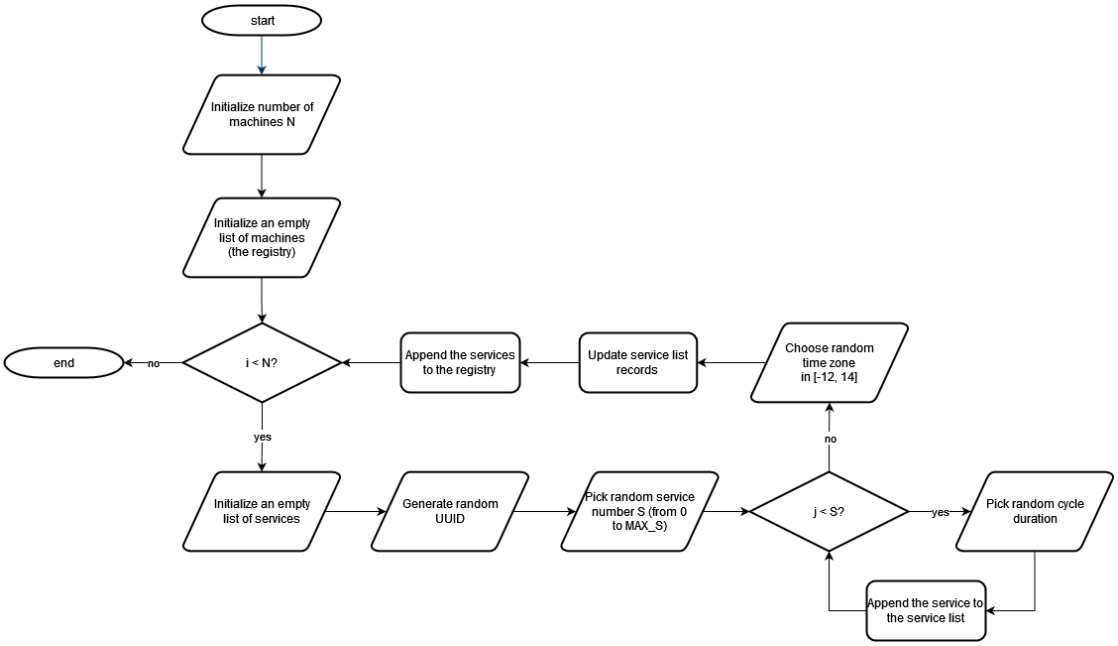


Figure 4.1: How the equipment registry is created and populated.

The flowchart in Figure 4.1 represents the process that leads to the registry creation. First of all, it is necessary to define how many machines will compose the table. Secondly, an empty list that represents the registry in which each record is stored is declared. Then, a cycle starts in order to create the i-th machine. In

this cycle, a random UUID is generated and a number of services for the current machine is randomly chosen from 1 to a maximum arbitrary value (MAX_S in the flowchart). The probability of picking only one service has deliberately been set as higher in order to avoid too many records per equipment. A nested cycle creates the j-th service, with its own cycle duration. In the end, a random time zone from -12 to +14 is chosen and put in all the service records of the current machine. Lastly, the service list is saved into the registry. Once the desired number of machines is reached the process stops and the registry creation is completed.

Note that the fields containing the total utilization, the average utilization, and the last update of the model have not been mentioned. This is because the utilization time and the machine learning models will be generated later, and these values are originally set to null. In particular, the process that generates the utilization for each machine is analyzed in the next section.

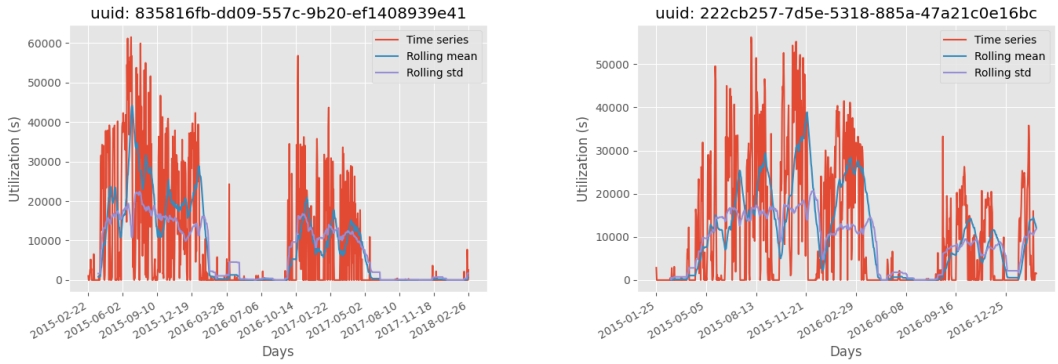
The Spring Model that works as an interface to map a database entry of the registry to a Java object can be found in Appendix A.1.

4.2 Usage dataset

The usage dataset is the database where the data used as input by the machine learning models are stored, so it represents the most interesting part of the data characterization.
The main idea at the basis of the data that can be of interest for this thesis work is the following: the analysis will be based on the usage time of industrial machine samples (e.g. robots, vehicles, etc.), so what we are looking for is a time series representing for each day the number of hours a piece of equipment has been used.

This statement leads to the following assumptions on the synthetic usage time we want to obtain:

- Day-night seasonality. Usually, equipment is mostly used during the day and less or not used during the night. This is not particularly important for our application, since we are interested in the overall usage time in a day, but has been considered just to be thorough and for possible future developments.



(a) Sample #1. **(b)** Sample #2.

Figure 4.2: Two time-series samples of equipment's daily usage time in seconds versus the day.

- Week seasonality. Usage will be concentrated on the weekdays and the probability of using an industrial machine during the weekend will be considered low.

- Some machines are more "stressed" during relatively short periods of time and they are almost unused during the remaining time.

- The amount of data is different for each piece of equipment. Old machines have more records with respect to recently introduced machines.

- The last trivial constraint is that the number of hours per day a machine can be used can not exceed 24 (or 86400 in seconds).

In Figure 4.2 the usage time, expressed in seconds, per day of two example machines, and its rolling mean and standard deviation evaluated on a 30-days window are presented. In particular, it is possible to notice some of the features previously described when speaking about the assumptions on synthetic data. From the rolling metrics, the different types of seasonality come to light. The fact that machines are not used for relatively long periods of time is clearly visible; the first machine remained almost not used from 2016-03-28 to 2016-10-2014 and from 2017-05-02 to 2018-02-26, and the second one presents very low usage from February to September 2016. The amount of data for the two machines is different; both of

UUID	service_id	date	util	util_to_m	days_to_m	pred
0c217...db055	0	2020-03-12	7582	15025	45	-
0c217...db055	0	2020-03-13	2468	12557	44	-
0c217...db055	0	2020-03-14	0	12557	43	-
5475f...a868a	0	2020-03-12	554	1250	-	1
df7f5...11701	0	2020-03-12	1545	85622	125	-
5475f...a868a	0	2020-03-13	0	1250	-	1
...	...	...	...	...	...	...
...	...	...	...	...	...	...
5bee2...0924a	0	2021-05-02	2548	17254	-	6
83ef6...045b7	0	2021-05-02	4862	48751	78	-
5bee2...0924a	0	2021-05-03	5488	11766	-	5
5bee2...0924a	0	2021-05-04	4587	7179	-	4
5bee2...0924a	0	2021-05-05	0	7179	-	4
83ef6...045b7	0	2021-05-03	1158	47593	77	-

Table 4.2: Usage dataset samples.

them start in 2015 but the first sample records end in 2018 while the other one in 2017.

Now, we should step into a more accurate description of the usage dataset creation and provide some examples.
Table 4.2 shows some sample records of the usage database.

The fields it contains are the following:

- Universally unique identifier (UUID). A 36-character string with dashes, digits, and letters, is intended to be universally unique in the system.

- Service ID. An identifier used to have more than one maintenance cycle for each machine.

- Date. A progressive date to store values for a specific day is used to define a time series.

- Utilization. It is the time in seconds the machine has been used during the day indicated in the previous field (Date). It is the field that stores the time series $U_m(t)$ defined in the problem formulation (Section 3.2).

- Utilization to maintenance. The remaining time in seconds a machine can be used before going to maintenance. It is the field where the time series $L_m(t)$ defined in the problem formulation (Section 3.2) is stored.
- Days to maintenance. The remaining calendar days before going to maintenance. This is the field used to train the machine learning algorithms and represents the target variable (i.e. what we want to predict). Obviously, it is not known day by day and its value is evaluated a posteriori. It is the field which contains the time series $D_m(t)$ defined in the problem formulation (Section 3.2).
- Predictions. The days remaining for maintenance, as predicted by the machine learning algorithms.

It is easy to understand that each row is uniquely identified by the fields `UUID`, `service_id` and `date`, that act as composite primary key of the table. As one can notice, the columns `days_to_maint` and `predictions` have some missing values. The reason is that, as we already said, the remaining days to maintenance can not be evaluated during the current cycle and we are interested in predictions only during the ongoing cycle, so past records do not include predictions. More details on how the days to maintenance are evaluated will be explained in the following paragraphs.

Now, let us analyze and further understand which is the purpose of the three fields `util`, `util_to_maint` and `days_to_maint`. The first one is simply the time in seconds the machine has been used during a specific day; it is synthetic data coming from the considerations previously made about the time series of usage time. The field `util_to_maint` is a counter used to keep track of the time a machine can still be used before going to maintenance. It is needed to understand when a cycle is over and the counter is restarted to its original value (the cycle duration contained in the *Equipment registry* discussed in Section 4.1).

The full code which performs the operations of creation and update of the table can be found in the Appendix A.2, while here the process is explained.
Figure 4.3 can help understand how the process works: each day the field `util_to_maint` is updated by making the difference between its value at the previous day and the utilization of the current day.

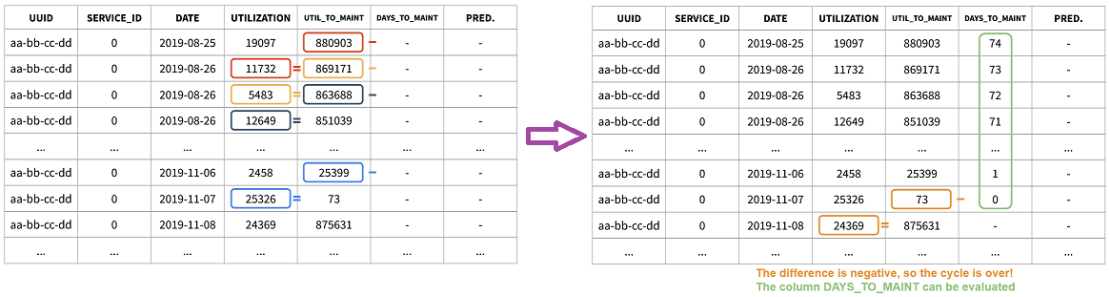
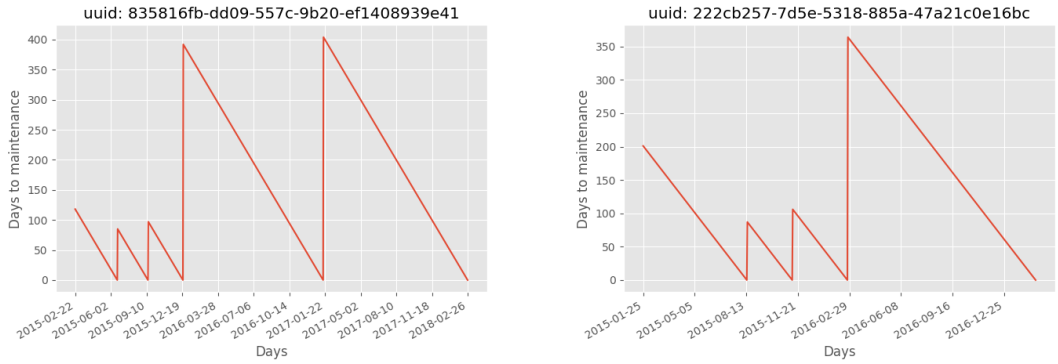


Figure 4.3: How the remaining utilization to maintenance (`util_to_maint`) is updated day by day and how the days left to next maintenance (`days_to_maint`) are evaluated at the end of a cycle.

When this difference is negative or equal to zero (line 71 of the code in the Appendix A.2), it means that there is no more usage time left and the machine should go to maintenance. In other words, the cycle is over and the field `days_to_maint` can be evaluated and inserted into the table from the beginning to the end of the cycle. At this point, the counter `util_to_maint` restarts from the value `cycle_dur` for that specific machine and service and the process is repeated.

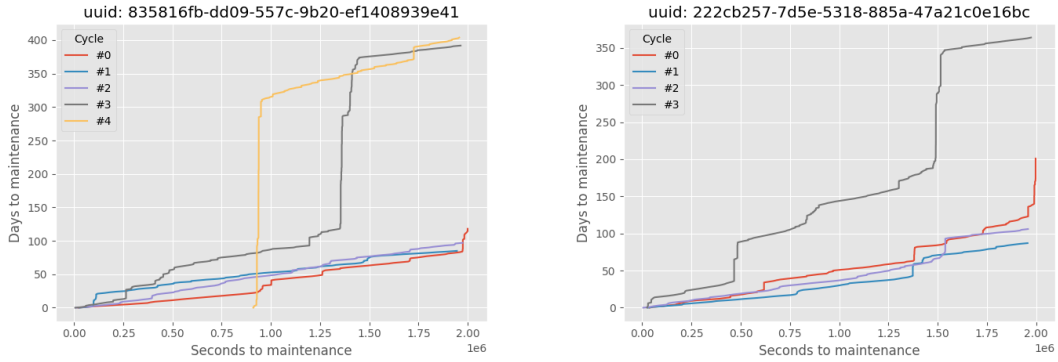
In conclusion, an analysis of the cycle duration is presented.
Two examples of the target series $D_m(t)$ representing the days left to the next maintenance are shown in Figure 4.4, The first sample with 5 cycles, the second sample with 4 cycles. When the days to maintenance reach 0, the machine enters maintenance mode. Once a new maintenance cycle has begun, the remaining days restart from a maximum value and monotonically decrease, by one day for every day that has gone, until the next maintenance operation is carried out. It is important to notice that the cycle duration varies a lot. The first sample's first three cycles have a duration of around 100 days, while for the last two cycles the duration is four times longer. This is because the usage does not follow specific patterns and low or zero utilization periods highly affect the cycle duration.

Figure 4.5 compares the number of usage seconds left for the subsequent maintenance to the number of days remaining for maintenance. When the seconds to maintenance are closer to 0, the functions appear to run at a constant rate, indicating that the usage rate is generally constant and not zero when the machine



(a) Sample #1. **(b)** Sample #2.

Figure 4.4: Two samples of remaining days to next maintenance versus the day.



(a) Sample #1. **(b)** Sample #2.

Figure 4.5: Utilization seconds left to maintenance versus the number of days to maintenance for all the cycles of two sample machines.

is approaching maintenance.

There are, however, a few vertical steps that signify consecutive days when there was no utilization. This demonstrates that the target variable is significantly affected by the occurrence of low or zero utilization periods. Consequently, determining the precise target value could be difficult. Hopefully, in the days leading up to the deadline, there will not be many extended periods of zero utilization.

Chapter 5

Results

In this chapter, we present the findings and some outputs obtained from the dataset that was examined in Chapter 4 using the methodologies described in Chapter 3.

Since the purpose of this chapter will mostly consist of an analysis of machine learning algorithms, a brief overview of the most common metrics used to evaluate the performance of classifiers and regressors is presented.

Results obtained for the hyperparameters' tuning both of classification and regression algorithms are shown and discussed. The different algorithms are then compared by means of plots, charts, and performance metrics.

5.1 Performance metrics

Performance metrics are an essential part of every machine-learning pipeline.
Your model might produce satisfactory results when measured with one metric, but unsatisfactory results when measured using another. For this reason, it is important to evaluate a machine learning model – classification as well as regression algorithms – by means of different metrics, in order to give it a fair evaluation.
We must pick our metrics for estimating ML performance very carefully because the statistic you select will determine exactly how the effectiveness of ML algorithms is assessed and compared. Moreover, the metric you select will have a significant impact on how you weigh the relative value of different variables in the outcome.

Various performance metrics that can be used to evaluate the results obtained

for classification and regression problems will be covered in this section.

5.1.1 Classification

Classification accuracy is the metric that is most frequently used to gauge how well a classification predictive model is doing. It may be calculated easily by dividing the number of examples in the test set that were successfully predicted by the total number of predictions made on the test set:

$$Accuracy = \frac{Correct\ Classifications}{Total\ Classifications} = \frac{TP + TN}{TP + TN + FP + FN}$$

On the other hand, the error rate may be determined by dividing the number of wrong predictions on the test set by the total number of predictions on the test set:

$$Error\ rate = \frac{Wrong\ Classifications}{Total\ Classifications} = \frac{FP + FN}{TP + TN + FP + FN}$$

Now, let us examine each of the elements in the aforementioned formulations individually:

- TP (True Positive) indicates the number of positive class samples that your model accurately predicted.
- TN (True Negative) indicates the number of negative class samples that your model accurately predicted.
- FP (False Positive) indicates the number of negative class samples that your model was inaccurate in predicting.
- FN (False Negative) indicates the number of positive class samples that your model was inaccurate in predicting.

Notice that the formulation with these factors only stands for a binary classifier, since only two classes are taken into consideration (positive and negative).

A tabular representation of the ground-truth labels and model predictions is called a confusion matrix (Figure 5.1). The instances in a predicted class are represented in each column of the confusion matrix, and the instances in actual

		Predicted class: Positive	Predicted class: Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

Figure 5.1: Binary confusion matrix.

classes are represented in each row. The confusion matrix is not really a performance statistic, but it serves as a sort of foundation for how other metrics assess the outcomes:

5.1.2 Regression

A common error metric for regression problems is called mean squared error, or MSE for short.

The average of the squared discrepancies between the predicted and expected target values in a dataset is used to calculate the MSE:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i and $\hat{y}_i$ are, respectively, the i-th observed and predicted values in the dataset. In order to remove the sign and get a positive error value, the difference between these two numbers is squared.
Large errors are also inflated or magnified as a result of the squaring. In other words, the wider the discrepancy between the expected and predicted values, the larger the squared positive error that results.

In addition to the mean squared error is the root mean squared error or RMSE. It is simply the squared root of the MSE:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Another popular metric is the mean absolute error (MAE). It is determined

by averaging the absolute errors, as suggested by its name. As a result, while calculating the MAE, the difference between an expected and forecasted value is forced to be positive.
The calculation is as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Unlike MSE and RMSE which punish bigger mistakes more severely than smaller ones, the MAE does not give distinct sorts of errors more or less weight; instead, the scores rise linearly as the number of errors increases.

The errors just defined do not account for or weigh the amount of time until the next maintenance. To put it another way, a 1-day error when we are close to the maintenance is equivalent to a 1-day error when we are distant from the maintenance. This problem is resolved by taking into account the mean residual error (MRE). It represents the average daily errors across a chosen range of days. We specifically want to calculate the average solely for specific D_m values found in a subset $\tilde{D}$, that is a list of days that are closer to the maintenance for each cycle (for example the last 30 days of each cycle).
The MRE is evaluated in the following way:

$$MRE = \frac{1}{|\tilde{D}|}\sum_{i:y_i \in \tilde{D}}|y_i - \hat{y}_i|$$

From all the errors' formulations is clear that the smaller they are, the better. Our main objective will be to minimize especially the MRE since we are most concerned about receiving precise forecasts when the machines are getting close to the conclusion of their maintenance cycle.

5.1.3 Hyperparameter tuning

Finding the optimal combination of hyperparameters to enhance the model's performance is known as hyperparameter tuning. It operates by conducting numerous trials within a single training procedure. Every trial entails the full execution of your training application with the values of the selected hyperparameters set within

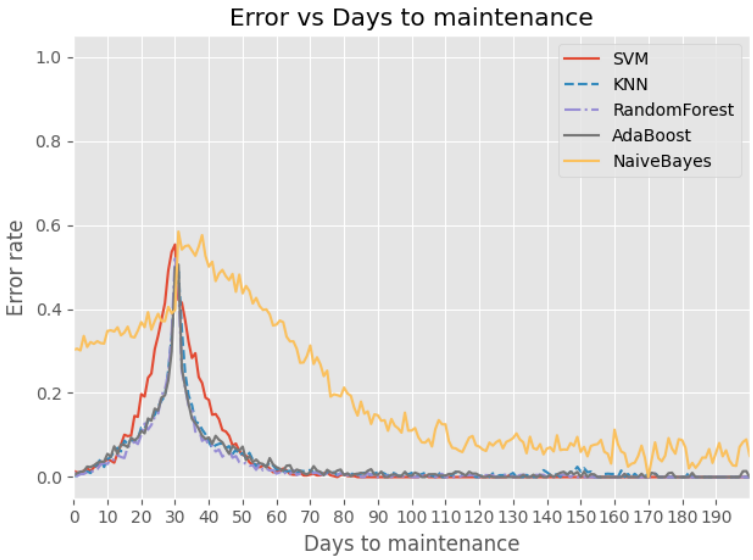


Figure 5.2: Overall error rates of all the classifiers vs. remaining days to next maintenance for all the machines in the database.

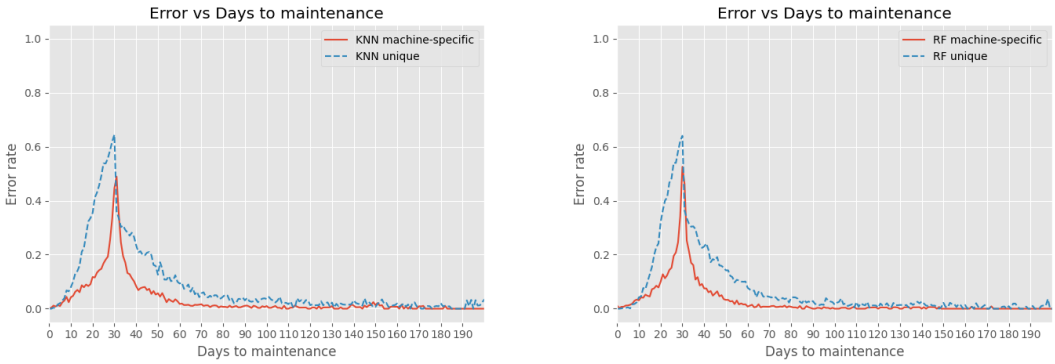
the predetermined bounds. Once this procedure is complete, you will have the set of hyperparameter values that the model requires to perform at its best.

The approach used to optimize the hyperparameters of the following classification and regression models is the grid search. With the grid search approach, we build a grid of potential hyperparameter values. Each iteration tries a set of hyperparameters in a certain sequence. It tracks the model performance when fitting the model with every conceivable set of hyperparameters. The best model with the best hyperparameters is then returned.

5.2 Classification models

In this section, we are going to analyze the performance of the classification models previously described, from a theoretical point of view, in Section 3.3.
The five analyzed classification methods are: Naive Bayes classifier, k-nearest neighbors (k-NN), Support Vector Machine (SVM), Random Forest classifier (RF) and AdaBoost.

The performance of the various classifier is analyzed in Figure 5.2 in terms of



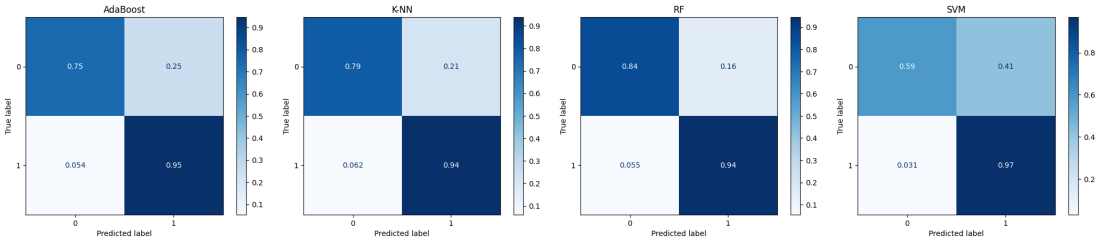
(a) K-nearest neighbors. **(b)** Random forest classifier.

Figure 5.3: Error rates of k-NN and RF classifiers vs. days to maintenance in two different scenarios: unique model and machine-specific models.

error rate. The results have been obtained by classifying the data of all the machines stored in the database, with a specific model for each machine. As expected all the classifiers have a maximum error rate at the threshold between the two classes (30 days to maintenance), which is almost 60% for all of them. The errors are really low when far from the threshold, both for smaller and larger values. The worst classification accuracy is obtained with the Naive Bayes classifier, which is the simplest one, so one could have expected this behavior. Random forest, k-NN, and AdaBoost have comparable error rates and result in the best classifiers. A bit worse is the SVM, especially in the area around the threshold, where the error rates are a bit higher with respect to the three just mentioned methods.

Figure 5.3 provides a different kind of analysis. K-NN and Random forest, which are the two best algorithms (AdaBoost has been discarded because it is a bit more time-consuming), have been tested under two different scenarios. A unique model is trained on the whole dataset and machine-specific models, each of them trained only on one machine's past data.

The unique model is obviously simpler to be trained and maintained. The downside is that, as expected, the accuracy of predictions is generally lower, both for k-NN and RF. The natural choice would be: going for a unique model in case simplicity and speed are fundamental, choosing to train a classification model for each machine if accuracy is important at the expense of having a more complex system.



(a) AdaBoost. **(b)** k-NN. **(c)** Random forest. **(d)** SVM.

Figure 5.4: Confusion matrix of AdaBoost, k-NN, RF, and SVM, classifying every machine's data with its own specific model.

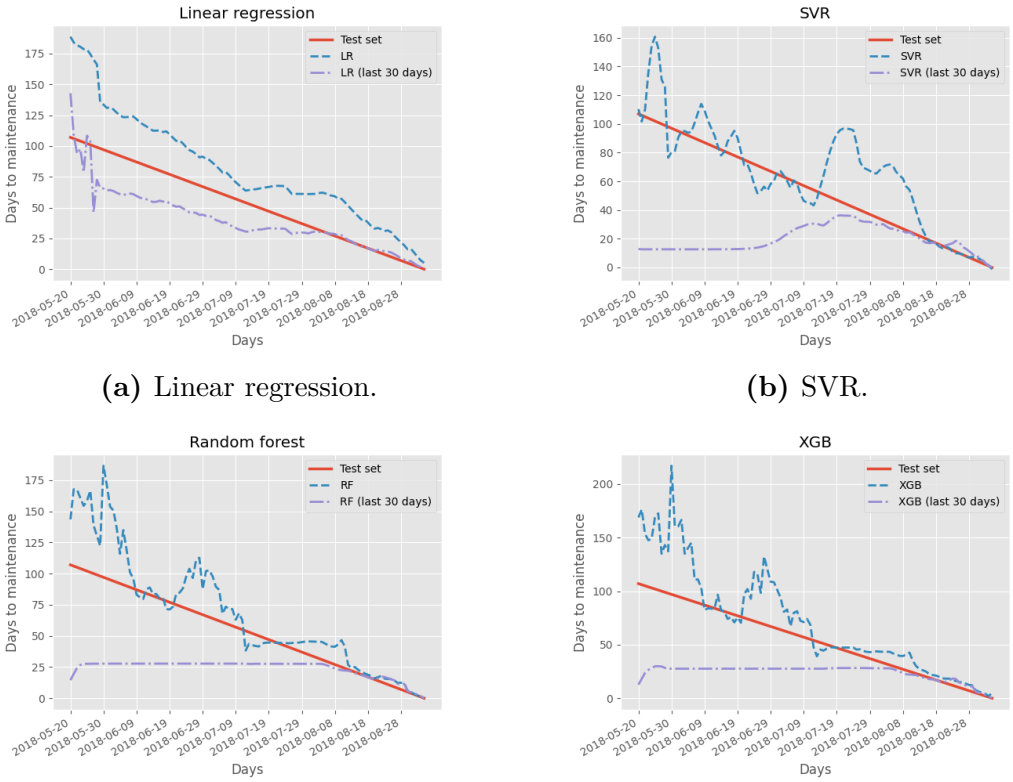
The last way of performance comparison is the confusion matrix, as in Figure 5.4. The label 1 is used to indicate the "more than 30 days left to maintenance" class, while 0 represents the opposite situation. The Naive Bayes classifier has been excluded from this analysis because, as already said, its performance is not satisfying when compared to the other algorithms. From a first look, the SVM is easily confirmed as slightly worse with respect to the other methods. In particular, the SVM outputs contain many false positives, meaning that the model is not able to accurately predict when less than 30 days are remaining. This can be due to the fact that cycles are usually way longer than 30 days, so the two classes are not balanced.

The best algorithm from this analysis results to be the random forest classifier, with a very low rate of false positives and false negatives, equal to 16% and 5.5% respectively.

5.3 Regression models

In this section, we are going to analyze the performance of the classification models previously described, from a theoretical point of view, in Section 3.4. The four analysed regression methods are: Linear Regression (LR), Support Vector Regression (SVR), Random Forest Regression (RF) and eXtreme Gradient Boosting (XGB).

The results presented in the following have been obtained considering one of the machines belonging to the system.



(a) Linear regression. **(b)** SVR.

(c) Random forest. **(d)** XGB.

Figure 5.5: Comparison between predicted values and test set of the four algorithms trained on the whole dataset and on the last 30 days of each cycle of one of the machines.

Some comparisons among the four algorithms of interest are shown in Figure 5.5. In red we find the actual values of the test set, i.e. what we aim to predict, in blue, there are the predictions made by the algorithms trained on the whole dataset, and in violet the predicted values obtained by the regression methods trained only on the last 30 days of each cycle. The violet lines are clearly very far from the ground truth at the beginning of the cycle (except for the LR which performs a bit better), while they become very precise in the last 30 days of the cycle. It is expected behavior since the algorithms are trained to work at their best when maintenance is forthcoming. The blue lines seem to follow the decreasing trend more accurately, although they are fluctuating close to the true values.

A similar performance comparison is reported in Table 5.1 from a metrics

	Whole data		Last 30 days	
Model	MAE	MRE	MAE	MRE
LR	28.314	19.869	14.973	1.439
SVR	16.350	7.129	35.038	2.440
RF	18.975	4.889	30.400	1.289
XGB	18.161	7.618	29.716	1.862

Table 5.1: Machines subset's average error metrics (MAE and MRE over the last 30 days) of the four algorithms trained on the whole dataset and on the last 30 days of each cycle.

perspective, which is more informative and precise in reflecting the actual differences among the methods. The MAE is the most important metric when speaking of models trained on the whole dataset. The worst performance has been obtained by linear regression, with a mean absolute error of 28.314 days. The other three algorithms have comparable errors in the range $[16, 18]$ days, almost halving the error of the LR. The MRE depicts the same situation, with the SVR, RF, and XGB that outperform the LR.

Looking at the version of the algorithms trained on the last 30 days of the cycles, we are more interested in the mean residual error evaluated on the subset composed by the last 30 days of the test set. In this case, all the methods have really low MRE, with the random forest resulting as the best with an MRE equal to 1.289 days. The MAEs are generally higher because are evaluated on the whole test set, while the algorithm is designed in order to have good accuracy on the last days before maintenance operations. As said from the analysis of Figure 5.5, the LR is the one that presents a good performance on the whole test set, even with the version trained on the last 30 days.

5.4 Dashboard

The Node-RED dashboard represents the graphical user interface to interact with the system. Figure 5.6 shows the output of the dashboard for one of the machines stored in the database.

On the left side, there is the time series of the daily usage time, in hours, versus

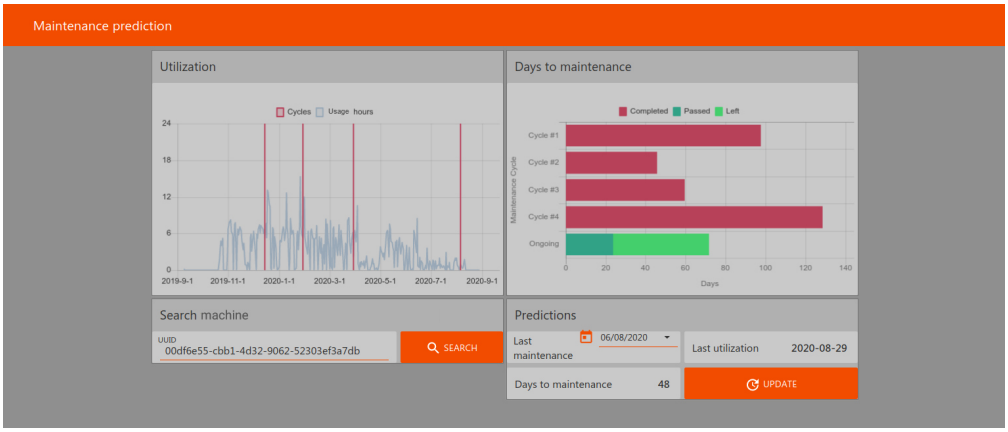


Figure 5.6: The dashboard interface.

the day, expressed as year-month-day. On the same plot, the vertical red lines mark the limits of the maintenance cycles.
Under this plot, we can find a search bar used to search a machine in the database.

On the right side of the dashboard, it is possible to find some useful information about past cycles' duration (in red) and the ongoing cycle (in green); the dark green bar are the days already passed during the ongoing maintenance cycle, while the light green amount represents the predicted days to maintenance.

The elements in the bottom-right corner are: the last time the machine was used, the predicted days to maintenance, a button to request an update of the model for the current machine, and an input box to enter the actual last maintenance. This is required because we cannot be sure that the last maintenance was carried out exactly at the end of the last cycle; the maintenance could have been anticipated or delayed for any reason and this should be taken into account.

Chapter 6

Conclusion

The final chapter is totally dedicated to drawing some conclusions and insights from the work that has been done. A short recap is provided, before discussing some ideas for future developments and possible improvements to the work.

The first sections of the introduction debate the important role of the Internet of Things in today's world. Having more and more devices and objects connected to the Internet is changing the way they interact with each other and with people. Huge volumes of data are collected and transferred over the networks every day, leading to the concept of Big Data. To extract insights from huge, varied, complicated data sets, a variety of approaches and technologies with novel forms of integration are needed. Data mining techniques are increasingly used by companies to extract knowledge from data, in order to achieve benefits.
One of these advantages represents the main topic of the thesis: help a company predict when industrial machines have to undergo periodic maintenance. The state-of-the-art analysis showed that a lot of research on topics related to predictive maintenance has been carried out during the last few years. The ability to predict maintenance brings efficiency in scheduling tasks, with substantial savings in terms of time and money.

Some common machine learning methods have been analyzed and their performance has been compared. The forecasting problem has been decomposed into two simpler problems: a preliminary classification to determine whether the machine is close to the next maintenance, and a regression actually making the prediction in

terms of days left to maintenance.

Simple methods like linear regression, or Naive Bayes classifier for classification, have been compared to more complex techniques, like ensemble methods. Good accuracy metrics have been obtained by using some of these algorithms, representing a solid foundation for future developments and improvements.

Another goal of this thesis is to propose a possible way to integrate the prediction system in a more complex environment. Common enterprise technologies, like the Spring Framework, or PostgreSQL, have been employed in order to present a complete and realistic solution. The structure and how the blocks of the system communicate with each other have been explained with a complete methodology definition, underlining the challenges that can arise when dealing with this kind of problem and how they can be solved. A prototype of a graphical interface allowing the user to interact with the system and look up the results has been proposed.

6.1 Future developments

Data collection and management systems used today are able to improve over time, leveraging artificial intelligence and machine learning solutions. Another aspect that should not be underestimated, particularly in cloud-based implementations, is that the predictive models used are constantly being updated and improved, making predictive maintenance systems increasingly effective over time. So, a predictive maintenance system can not be considered a static solution, but continuous adjustments and updates are required.

A possible future improvement of the work discussed in this thesis is particularly related to the machine learning models and could be the enrichment of the dataset with other features. Indeed, only temporal data, such as the historical records of daily working hours of previous days were employed in the study. Other changing parameters or characteristics that have an impact on the maintenance of the equipment, such as the type of machine (e.g. industrial robot, vehicle, etc.), measurements coming from sensors (e.g. pressure, voltage, etc.), details about the task a machine is working on, etc. can further enhance the features. These are merely a few suggestions for enhancing model correctness by adding extra features to the input, but others can be easily found.

Another important aspect is related to the source of the data. As discussed in Chapter 4, the data employed to train and test the predictive models are synthetic. Further validation of the system should include the use of real-world data coming from industrial equipment.

Considering a company with heterogeneous machines, in terms of usage patterns and type, another consideration could be done. Instead of developing a predictive model for each machine, some unsupervised learning techniques (like clustering) can be used to firstly group the equipment. With this approach, predictions can be made even for new machines (with few historical records), by using data coming from similar equipment.

In conclusion, we can say that the key to applying a predictive maintenance method is to collect historical data, so the first step is to equip the machines with sensors and establish a network infrastructure. The work presented in this thesis could represent a starting point and an initial analysis of a possible way to integrate a predictive maintenance system in a company environment. The technologies are easily adaptable and can also be changed in order to be compliant with the frameworks and tools already in use by a company.

All this surely involves costs and investments, but a company that today has maintenance among its major cost items should undoubtedly consider switching to a predictive system, which will ensure greater efficiency and cost containment.

Appendix A

Codes

A.1 Equipment registry model

Each equipment stored into the registry is mapped in a Spring *@Entity*. Using the MVC pattern, it represents a *Model* that handles the access to data needed by the application (machines in this case).

Listing A.1: Machine model

```
@Entity
@Table(name = "machines_registry")
@IdClass(MachineId.class)
public class Machine {

  @Id
    private String uuid;
    @Id
    @Column(name = "service_id")
    private int serviceId;
    private int cycle_dur;
    private int time_zone;
    private int tot_util;
    private int avg_util;
    private LocalDate last_update;

```

```
17    public Machine() {
18    }
19
20  public Machine(String uuid, int serviceId, int cycle_dur, int
      ↪ time_zone, int tot_util, int avg_util,
21      LocalDate last_update) {
22    super();
23    this.uuid = uuid;
24    this.serviceId = serviceId;
25    this.cycle_dur = cycle_dur;
26    this.time_zone = time_zone;
27    this.tot_util = tot_util;
28    this.avg_util = avg_util;
29    this.last_update = last_update;
30  }
31
32  public String getUuid() {
33    return uuid;
34  }
35
36  public void setUuid(String uuid) {
37    this.uuid = uuid;
38  }
39
40  public int getServiceId() {
41    return serviceId;
42  }
43
44  public void setServiceId(int serviceId) {
45    this.serviceId = serviceId;
46  }
47
48  public int getCycle_dur() {
49    return cycle_dur;
50  }
51
```

```
52  public void setCycle_dur(int cycle_dur) {
53    this.cycle_dur = cycle_dur;
54  }
55
56  public int getTime_zone() {
57    return time_zone;
58  }
59
60  public void setTime_zone(int time_zone) {
61    this.time_zone = time_zone;
62  }
63
64  public int getTot_util() {
65    return tot_util;
66  }
67
68  public void setTot_util(int tot_util) {
69    this.tot_util = tot_util;
70  }
71
72  public int getAvg_util() {
73    return avg_util;
74  }
75
76  public void setAvg_util(int avg_util) {
77    this.avg_util = avg_util;
78  }
79
80  public LocalDate getLast_update() {
81    return last_update;
82  }
83
84  public void setLast_update(LocalDate last_update) {
85    this.last_update = last_update;
86  }
87
```

```
88   @Override
89   public String toString() {
90     return "Machine [uuid=" + uuid + ", serviceId=" + serviceId + ",
     ↪ cycle_dur=" + cycle_dur + ", time_zone="
91         + time_zone + ", tot_util=" + tot_util + ", avg_util=" +
     ↪ avg_util + ", last_update=" + last_update
92         + "]";
93   }
94 }
```

A.2 Usage dataset creation and update

The usage table is created and updated by a Spring *@Component*. It contains a function `updateTable` scheduled to run every hour in order to update the machines located in time zones where the day is over (at midnight) and new information about utilization can be available.
The auxiliary function `updateCycle` update the table with days left to maintenance once a cycle is complete.

Listing A.2: Table Updater

```
1  @Component
2  public class TableUpdater {
3
4    private static final Logger logger =
       ↪ LoggerFactory.getLogger(TableUpdater.class);
5
6    @Autowired
7    private IMachineService machineService;
8
9    @Autowired
10   private IMyRecordService recordService;
11
12   @Autowired
13   private IUtiliztionService utilizationService;
```

```
14
15  private void updateCycle(List<MyRecord> records, int daysCount) {
16    int index = records.size() - 1;
17    for (int i=0; i<=daysCount; i++) {
18      MyRecord record = records.get(index);
19      record.setDays_to_maint(i);
20      records.set(index, record);
21      index--;
22    }
23  }
24
25  @Scheduled(initialDelay = 24*3600000, fixedRate = 3600000)
26  public void updateTable() {
27    Date now = new Date();
28    int hour = (int)(now.getTime() % 86400000) / 3600000;
29    int midnightTz;
30    if (hour <= 11) {
31      midnightTz = -hour;
32    } else {
33      midnightTz = 24 - hour;
34    }
35    logger.info("Updating usage dataset of time zone: " + midnightTz );
36
37    List<Machine> machines = (List<Machine>)
      ↪ machineService.findByTimezone(midnightTz);
38
39    for (Machine v:machines) {
40
41      List<MyRecord> records = new ArrayList<>();
42
43      LocalDate lastDate = recordService.getLastDate(v.getUuid());
44
45      ArrayList<DailyUtilization> dailyUtilization =
      ↪ utilizationService.findDailyUtilization(v.getUuid());
46      if (dailyUtilization.isEmpty()) {
47        // If the machine has no utilization, skip it
```

```
48          continue;
49        }
50        if (lastDate != null) {
51          dailyUtilization = (ArrayList<DailyUtilization>)
    ↪ dailyUtilization.stream()
52              .filter(e -> e.getDate().isAfter(lastDate))
53              .collect(Collectors.toList());
54        }
55        int utilToMaintenance = v.getCycle_dur();
56        int daysCount = -1;
57        for (int i=0; i<dailyUtilization.size(); i++) {
58          DailyUtilization u = dailyUtilization.get(i);
59          int utilization = 0;
60          if (u.getEngineHours() != null) {
61            utilization = u.getEngineHours().intValue();
62            if (utilization > 86400) {
63              utilization = 86400;
64            } else if (utilization < 0) {
65              utilization = 0;
66            }
67          }
68          LocalDate date = u.getDate();
69          LocalDate next_date = (i+1 < dailyUtilization.size()) ?
    ↪ dailyUtilization.get(i+1).getDate() : date.plusDays(1);
70          while ( date.isBefore(next_date) ) {
71            if (utilToMaintenance - utilization <= 0) {
72              updateCycle(records, daysCount);
73              utilToMaintenance = v.getCycle_dur();
74              daysCount = -1;
75            }
76            daysCount ++;
77            utilToMaintenance -= utilization;
78
79            // Append new record to list
80            records.add(new MyRecord(v.getUuid(), v.getServiceId(), date,
    ↪ utilization, utilToMaintenance, null, null));
```

```
81            date = date.plusDays(1);
82            utilization = 0;
83          }
84        }
85        // Insert the records in the table
86        recordService.saveAll(records);
87
88        // Update tot. util. and avg util of the machine
89        Integer totUtil = recordService.getTotalUtilization(v.getUuid());
90        Integer avgUtil = recordService.getAvgUtilization(v.getUuid());
91        if (totUtil == null || avgUtil == null) {
92          totUtil = 0;
93          avgUtil = 0;
94        }
95        machineService.save(new Machine(v.getUuid(),
96            v.getServiceId(),
97            v.getCycle_dur(),
98            v.getTime_zone(),
99            totUtil, avgUtil, null));
100     }
101     logger.info("Usage dataset of time zone " + midnightTz + "
      ↪ updated.");
102   }
103 }
```

A.3 The machine learning module

It is the piece of Python code that is in charge of all procedures related to machine learning. It has been designed as a REST web service using the Flask framework. The service exposes two routes. */update* takes as parameters a `UUID` and `service_id` in order to check if the models can be retrained. */predict* takes as parameters the `UUID`, the `service_id` and the date of last maintenance `last_maint`. It responds with the predictions of the remaining days to maintenance according to the parameters in the request.

Listing A.3: Machine learning module

```
app = Flask(__name__)

@app.route('/update')
def update_model():
    uuid = request.args.get('uuid', type=str)
    service_id = request.args.get('service_id', type=int)

    if uuid is None:
        abort(404, 'Invalid uuid, should be a string.')

    if service_id is None:
        abort(404, 'Invalid service_id, sould be an integer.')

    db_mgr = DbManager()
    db_mgr.connect()

    try:
        # Get machine's information
        rows = db_mgr.get_machine(uuid, service_id)
        keys = ['uuid', 'service_id', 'cycle_dur', 'time_zone',
    ↪ 'tot_util', 'avg_util', 'last_update']
        machine = dict(zip(keys, rows[0]))
    except IndexError:
        abort(404, f"Machine {uuid} with service_id {service_id} not
    ↪ found.")

    # Get table
    rows = db_mgr.get_table(uuid, service_id)
    keys = ['uuid', 'service_id', 'date', 'util', 'util_to_maint',
    ↪ 'days_to_maint', 'predictions']

    if not rows:
        abort(404, f"Machine {uuid} has no utilization history.")

    time_window_size = 10
```

```
33      tot_util_sec = machine['tot_util'] * 3600
34      if tot_util_sec >= machine['cycle_dur']:
35          # Old machine
36          df = pd.DataFrame(rows, columns=keys)
37          filename = f'./models/{uuid}_{service_id}.sav'
38          last_cycle_end = df[df['days_to_maint'] == 0].iloc[-1]['date']
39          # Check if the model can be trained again with new data
40          if machine['last_update'] is None or last_cycle_end >
    ↪ machine['last_update'] or not os.path.isfile(filename):
41              df = Model.build_dataset(df, time_window_size)
42              X_train, X_test, y_train, y_test =
    ↪ Model.my_train_test_split(df, last_30_days=True)
43              Model.train_random_forest(filename, X_train, y_train)
44              db_mgr.set_last_update(machine['uuid'],
    ↪ machine['service_id'], last_cycle_end)
45
46      response = {'uuid': machine['uuid'], 'service_id':
    ↪ machine['service_id'], 'updated': True}
47
48      db_mgr.close()
49
50      return jsonify(response)
51
52
53  @app.route('/predict')
54  def predict():
55      uuid = request.args.get('uuid', type=str)
56      service_id = request.args.get('service_id', type=int)
57      last_maint = request.args.get('last_maint', type=str)
58
59      if uuid is None:
60          abort(404, 'Invalid uuid, should be a string.')
61
62      if service_id is None:
63          abort(404, 'Invalid service_id, sould be an integer.')
64
```

```
65      if last_maint is None:
66          abort(404, 'Invalid last_maint, sould be a string.')
67      else:
68          try:
69              last_maint = datetime.strptime(last_maint,
    ↪ '%Y-%m-%d').date()
70          except:
71              abort(404, "Unable to parse last_maint, format should be
    ↪ '%Y-%m-%d'")
72
73      db_mgr = DbManager()
74      db_mgr.connect()
75
76      try:
77          # Get machine's information
78          rows = db_mgr.get_machine(uuid, service_id)
79          keys = ['uuid', 'service_id', 'cycle_dur', 'time_zone',
    ↪ 'tot_util', 'avg_util', 'last_update']
80          machine = dict(zip(keys, rows[0]))
81      except IndexError:
82          abort(404, f"Machine {uuid} with service_id {service_id} not
    ↪ found.")
83
84      # Get table
85      rows = db_mgr.get_table(uuid, service_id)
86      keys = ['uuid', 'service_id', 'date', 'util', 'util_to_maint',
    ↪ 'days_to_maint', 'predictions']
87      df = pd.DataFrame(rows, columns=keys)
88
89      if not rows:
90          abort(404, f"Machine {uuid} has no utilization history.")
91
92      last_util_date = df.iloc[-1]['date']
93
94      # Select the last 'time_window_size' records in the dataframe
95      time_window_size = 10
```

```
96      to_be_predicted = Model.build_prediction_X(df, time_window_size,
        ↪ last_maint, machine['cycle_dur'])
97
98      tot_util_sec = machine['tot_util'] * 3600
99      if tot_util_sec >= machine['cycle_dur']:
100         filename = f'./models/{uuid}_{service_id}.sav'
101         if os.path.isfile(filename):
102             prediction = Model.predict(filename, to_be_predicted)[0]
103             status = 'ok'
104         else:
105             prediction = None
106             status = 'model not found'
107     else:
108         filename = './models/unified.sav'
109         if os.path.isfile(filename):
110             prediction = Model.predict(filename, to_be_predicted)[0]
111             status = 'ok'
112         else:
113             prediction = None
114             status = 'model not found'
115
116     last_daily_util = int(to_be_predicted['util'].iloc[0])
117
118     response = {'uuid': machine['uuid'], 'service_id':
        ↪ machine['service_id'],
119                 'date': last_util_date, 'daily_util': last_daily_util,
120                 'days_to_maint': prediction, 'status': status}
121
122     db_mgr.close()
123
124     return jsonify(response)
125
126
127 if __name__ == '__main__':
128     app.run(host='0.0.0.0')
```

Bibliography

[1] Stephen Ornes. «Core Concept: The Internet of Things and the explosion of interconnectivity». In: *Proceedings of the National Academy of Sciences* 113.40 (2016), pp. 11059–11060. ISSN: 0027-8424. DOI: 10.1073/pnas.1613921113. eprint: https://www.pnas.org/content/113/40/11059.full.pdf. URL: https://www.pnas.org/content/113/40/11059 (cit. on p. 1).

[2] Kevin Ashton. «That 'internet of things' thing». In: *RFID journal* 22.7 (2009), pp. 97–114 (cit. on p. 1).

[3] E. Dave et al. «How the next evolution of the internet is changing everything». In: *The Internet of Things* (2011) (cit. on pp. 1, 2).

[4] Syeda Tahsien, Hadis Karimipour, and P. Spachos. «Machine learning based solutions for security of Internet of Things (IoT): A survey». In: *Journal of Network and Computer Applications* 161 (Apr. 2020), p. 102630. DOI: 10.1016/j.jnca.2020.102630 (cit. on p. 2).

[5] Tanweer Alam. «A Reliable Communication Framework and Its Use in Internet of Things (IoT)». In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 3 (May 2018) (cit. on p. 2).

[6] Huadong Guo, Lizhe Wang, Fang Chen, and Dong Liang. «Scientific big data and Digital Earth». In: *Chinese Science Bulletin (Chinese Version)* 59 (Dec. 2014), p. 1047. DOI: 10.1360/972013-1054 (cit. on p. 2).

[7] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011 (cit. on p. 3).

[8] David J Hand and Niall M Adams. «Data Mining». In: *Wiley StatsRef: Statistics Reference Online* (2014), pp. 1–7 (cit. on p. 3).

[9] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016 (cit. on p. 3).

[10] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020 (cit. on p. 3).

[11] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018 (cit. on p. 3).

[12] Y. Huo, W. Tu, Z. Sheng, and V. C. M. Leung. «A survey of in-vehicle communications: Requirements, solutions and opportunities in IoT». In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 132–137. DOI: 10.1109/WF-IoT.2015.7389040 (cit. on p. 6).

[13] Hashem M Hashemian. «State-of-the-art predictive maintenance techniques». In: *IEEE Transactions on Instrumentation and measurement* 60.1 (2010), pp. 226–236 (cit. on pp. 7–9).

[14] Basit Farooq, Jinsong Bao, Jie Li, Tianyuan Liu, and Shiyong Yin. «Data-driven predictive maintenance approach for spinning cyber-physical production system». In: *Journal of Shanghai Jiaotong University (Science)* 25.4 (2020), pp. 453–462 (cit. on p. 12).

[15] Omkar Motaghare, Anju S Pillai, and KI Ramachandran. «Predictive maintenance architecture». In: *2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. IEEE. 2018, pp. 1–4 (cit. on p. 11).

[16] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. «Machine learning for predictive maintenance: A multiple classifier approach». In: *IEEE transactions on industrial informatics* 11.3 (2014), pp. 812–820 (cit. on p. 12).

[17] Patrick Killeen, Bo Ding, Iluju Kiringa, and Tet Yeap. «IoT-based predictive maintenance for fleet management». In: *Procedia Computer Science* 151 (2019), pp. 607–613 (cit. on p. 13).

[18] Yong Sun, Zhentao Xu, and Tianyu Zhang. *On-Board Predictive Maintenance with Machine Learning*. Tech. rep. SAE Technical Paper, 2019 (cit. on p. 13).

[19] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. Vol. 734. John Wiley & Sons, 2011 (cit. on p. 13).

[20] Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. «A review on time series forecasting techniques for building energy consumption». In: *Renewable and Sustainable Energy Reviews* 74 (2017), pp. 902–924. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2017.02.085. URL: http://www.sciencedirect.com/science/article/pii/S1364032117303155 (cit. on p. 14).

[21] Ratnadip Adhikari and RK Agrawal. «A combination of artificial neural network and random walk models for financial time series forecasting». In: *Neural Computing and Applications* 24.6 (2014), pp. 1441–1449 (cit. on p. 14).

[22] Yukun Bao, Tao Xiong, and Zhongyi Hu. «Multi-step-ahead time series prediction using multiple-output support vector regression». In: *Neurocomputing* 129 (2014), pp. 482–493 (cit. on p. 14).

[23] Jeffrey O Grady. *System integration*. Vol. 5. CRC press, 1994 (cit. on p. 14).

[24] Dmitry Namiot and Manfred Sneps-Sneppe. «On micro-services architecture». In: *International Journal of Open Information Technologies* 2.9 (2014), pp. 24–27 (cit. on p. 15).

[25] Craig Walls. *Spring Boot in action*. Manning Publications, 2016 (cit. on p. 15).

[26] Daniel Berrar. «Bayes' theorem and naive Bayes classifier». In: *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics* 403 (2018) (cit. on p. 21).

[27] Robert E Schapire. «Explaining adaboost». In: *Empirical inference*. Springer, 2013, pp. 37–52 (cit. on p. 23).