UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE
IN COMPUTER ENGINEERING

"Machine Learning-based Approaches for Advanced Monitoring of Smart Glasses"

Relatore: Prof. / Dott. Susto Gian Antonio

Laureando: Corsetti Rocco

ANNO ACCADEMICO 2022 - 2023

27 Febbraio 2023

Abstract

With today's growing demand on productivity, product quality and effectiveness, the importance of Machine Learning-based functionalities and services has dramatically increased. Such paradigm shift can be mainly associated with the increasing availability of Internet of Things (IoT) sensors and devices, the increase of data collected in the IoT scenario and the increasing popularity and availability of machine learning approaches. One of the most appealing applications of ML-based solutions is for sure Predictive Maintenance, which aims at improving maintenance management by exploiting the estimation of the health status of a piece of equipment. One of the main formalizations of the PdM problem is the prediction of the Remaining Useful Life (RUL), that is defined as the time/process iterations remaining for a device component to perform its task before it loses functionality. This work investigates a possible application of predictive maintenance techniques for the monitoring of the battery of Smart Glasses. The work starts with the description of the considered devices, the modalities of data collection and the Exploratory Data Analysis for better understanding the task. The first experimental part consists in the application of an unsupervised anomaly detection technique, useful to initially deal with the partial and unlabeled data. The last part of the work contains the results of the application of both classical machine learning and deep learning approaches for the estimation of the RUL of the devices battery. A section for the interpretation of the machine-learning models is included for both the anomaly detection and RUL estimation approaches.

Contents

1 Introduction

With today's growing demand on productivity, product quality and effectiveness, the importance of Machine Learning-based functionalities and services has dramatically increased. Such a paradigm shift can be mainly associated with the increasing availability of Internet of Things (IoT) sensors and devices, the increase of data collected in the IoT scenario and the increasing popularity and availability of ML approaches. One of the most appealing applications of ML-based solutions is for sure Predictive Maintenance, which aim at improving maintenance management.

Approaches to maintenance management can be grouped into three main categories which, in order of increasing complexity and efficiency, are as follows.

- **Run-to-failure** (R2F) — where maintenance interventions are performed only after the occurrence of failures. This is obviously the simplest approach dealing with maintenance (and for this reason, it is frequently adopted), but it is also the least effective one, as the cost of interventions and associated downtime after failure are usually much more substantial than those associated with planned corrective actions taken in advance.

- **Preventive Maintenance** (PvM) — where maintenance actions are carried out according to a planned schedule based on time or process iterations. With this approach, also known as scheduled maintenance, failures are usually prevented, but unnecessary corrective actions are often taken, leading to inefficient use of resources and increased operating costs.

- **Predictive Maintenance** (PdM) — where maintenance is performed based on an estimate of the health status of a piece of equipment. PdM systems allow for advance detection of pending failures and enable timely pre-failure interventions, thanks to prediction tools based on historical data, ad-hoc defined health factors, statistical inference methods, and engineering approaches.

By enabling PdM technologies, many advantages can be envisioned: (1) PdM minimizes equipment downtime and costs of repairs; (2) PdM can extend the life of the machine's component and defers new purchases; (3) PdM can maximize usage/productivity of a device/system. Moreover, for Original Equipment Manufacturers, new services can be enabled thanks to PdM, leading to a new stream of revenues, more satisfied customers and/or more efficient service operations. Considering the costs of repair, for example, in survey [19] it is reported that: "[...] depending on the industry between 15 and 70 percent of total production costs originate from maintenance activities" and that: "[...] about 33 cents of every dollar spent on maintenance in the US is wasted because of unnecessary maintenance activities". IBM, in its report [34], summarized the advantages of maintenance management systems: "[...] companies that used computerized maintenance management systems exhibited an average of:

- 28.3% increase in the productivity of maintenance
- 20.1% reduction in equipment downtime
- 19.4% savings in the cost of materials
- 17.8% decrease in inventory maintenance and repair
- 14.5 months payback time [...]"

Applications of PdM technologies can be found in very different fields; in report [34] of IBM, for instance, three examples of real implementations are presented:

- "British designer and manufacturer of intelligent lighting and intelligent building solutions, PhotonStar Technology, develops systems that collect facilities and equipment metrics such as energy use and building occupancy, encrypts the information and consolidates it for analysis on the cloud. There, its customers use dashboards to track efficiency, create predictive maintenance plans and remotely monitor real-time status."

- "A Japanese automobile manufacturer uses IoT to model the behavior of their welding process. It wanted to identify causal factors of failures and faults and find top predictors of equipment failure. The system delivers 90 percent prediction of faults with no false positives; 50 percent of the faults are predicted over 2 hours in advance. The company saved 1.5 hours per fault thanks to advanced prediction."

- "A major aircraft manufacturer is using IoT to maintain calibration of precision assembly tools and improve manufacturing quality. Data from shop floor tools along with equipment failure data is used in predictive quality analytics to generate models that identify tools likely to need servicing. Faulty tools are proactively removed from the shop floor to be maintained and recalibrated, leading to significant improvements in manufacturing quality. The solution has enabled a 100 percent payback within one year — avoiding millions of dollars of rework and months of production delays by preventing out-of-alignment tools from remaining in the aircraft production workflow."

One of the main formalization of the PdM problem is the prediction of the **Remaining Useful Life** (RUL). RUL is defined as the time/process iterations remaining for a device component to perform its tasks before it loses functionality. RUL can also be defined as the duration (which can be, for example, in minutes, hours, days of usage or process iterations) from the current time to the end of the useful life of a component. This estimation could be exploited to optimize a trade-off between reducing the risk of unexpected failure risks and increasing the exploited lifetime of a component. For example, in the event of failure, there is a high cost to repair damaged parts. For this reason, companies try to prevent failure events before they happen by performing regular checks on the equipment.

Unfortunately, RUL formalization can be challenging when the available data is unlabeled or when there are few examples of failures. Indeed, labelling is often done manually by human experts and, consequently, requires efforts to obtain the labelled training dataset. Moreover, many devices are, of course, designed to not fail, making the problem, from a machine learning

perspective, difficult to be tackled in some cases. For such reasons, **Unsupervised Anomaly Detection** can in some cases work better than RUL prediction because it does not need labeled data and it can still provide useful information related to the device behavior, by providing a so-called Anomaly Score that is a quantitative index characterizing if the collected data are anomalous or not. In this work, some preliminary solutions are provided to monitor the health of two different smart glasses platforms, with a particular focus on the RUL prediction for the battery. First, the data sets are introduced along the data collection strategies, and an exploratory data analysis is performed. Then some preliminary analyses are conducted, starting with Anomaly Detection and ending with application of classic machine learning models; both these parts include a section on machine learning interpretability. The final part of the work consists of advanced experiments, focusing mainly on deep learning, in which an expanded data set is considered for the RUL prediction task. The thesis ends with a brief comparison with the literature and a conclusion on the experiments performed and their limitations.

2 Dataset

In this work, two different datasets were created and used. The first one was collected from the ISee platform, the second one, obtained in a second moment, is based on a different platform.

2.1 ISee Dataset

Data from the ISee platforms were acquired in real-time, recorded every minute. Table 1 shows the fields taken into account, most of the features are acquired through sensors available on the device. At the beginning of the work, the platforms available for the data recording were five, but unfortunately, during the data collection, three of them ceased to work. After three months of data collection, it was possible to use other devices for the dataset creation, for a total of eight working smart glasses. Table 2 reports the number of cycles collected for each smart glasses: the entries are identified by the MAC address of the device.

2.1.1 Data Recording

ISee sensors are equipped with a Bluetooth Low Energy (BLE) communication module that publishes all sensors readings. To create a dataset that could be used for this task, all the charging and discharging cycles of each battery were recorded: a charging cycle consists of a complete charge of the battery, from 0% to 100%, a discharging cycle is the inverse. To record the data and manage the entire communication between the smart glasses and the computers, a Python class, called ISee_manager, was designed with the following key functionalities:

- Scan and connection: methods for the scan and discovery of BLE devices; it can automatically filter out all the non-ISee devices. It can connect to the first ISee found or to a specific sensor if its address is specified in input

Feature	**Description**
address	address of smart glasses
acc	accelerometer along the x,y and z axes
gyr	gyroscope along the x, y and z axes
mag	magnetometer along the x, y and z axes
roll	longitudinal axis of smart glasses
pitch	transverse axis of smart glasses
yaw	normal axis of smart glasses
rh	relative humidity
temp	temperature
uva	UVA sensor
uvb	UVB sensor
x	horizontal position
y	vertical position
blueg	blue good
blueb	blue bad
pressure	barometer
worn	adherence sensor
battery_lvl	level of battery (%)

Table 1: Data variables and description

Address	Number of battery cycles	Recording strted
D5:B8:15:AD:01:0D	114	July
F0:53:52:26:FD:2F	117	July
F9:F1:B2:25:B1:B0	7	September
E3:90:78:A9:6F:BA	7	September
E5:FC:6A:55:5E:83	4	September
ED:F2:0E:AF:7C:4C	50	December
E8:22:97:31:0D:4A	50	December
E7:D7:43:C8:22:1E	49	December
E3:38:BA:9F:0F:B0	46	December
E5:2A:C4:7E:F2:F6	39	December
EC:3B:5C:4D:BA:B6	40	December

Table 2: Number of battery cycles collected for each smart glasses.

- Notification management: methods that can start and stop the notification flow; they also store the raw data in a buffer
- Data decoding: methods for decoding the raw data, formatting it and saving / returning it

All low-level functionalities are managed with the Python library **Bleak**.

Since there are two groups of sensors with different frequencies, the data readings were aggregated by considering the average, maximum and minimum of each feature and combining them in a single reading. The scripts used for the data recording have a similar structure: they connect to the sensor and every minute read the data for a few seconds, then aggregate the reading in a unique sample and wait for the next reading. The first script is summarized in Algorithm 1

The Algorithm 1 works really well, the connection is stable and the data recording is flawless, but the fact that the connection is established at the beginning of the script and continued until the end results in a slow discharge: the sensors take about 9 hours for a discharging cycle. It was noticed that keeping connecting and disconnecting the sensors results in a battery dura-

Input: $reading\_time \leftarrow 10$, $sample\_time \leftarrow 60$, address: optional
$isee \leftarrow Isee\_manager(address)$
$to\_save \leftarrow List()$
`// Connect to device`
$isee.find\_and\_connect()$
while *isee.is_connected() and isee.is_not_charged()* **do**
    $start\_reading \leftarrow time.time()$
    $isee.start\_notification()$
    $data \leftarrow isee.read\_data(reading\_time)$
    $isee.stop\_notification()$
    `// Aggregate data`
    $agg\_data \leftarrow aggregate\_data(data)$
    $to\_save.append(agg\_data)$
    $sleep(sample\_time - (time.time() - start\_time))$
end
`// If charged, disconnect`
if *isee.is_charged()* **then**
    $isee.disconnect()$
end
return *to_save.as_dataset()*

Algorithm 1: First ISee record script

tion that is almost half with respect the normal one, so, in order to collect data faster, the script was modified accordingly. However, the modification brought great connectivity instability: the sensors wrongly disconnected during data recording and sometimes it was impossible to reconnect back to them. To deal with these issues two scripts were created, one that does a single data reading and one that just calls in loop the first program; in this way unexpected interruptions of connection do not result in the impossibility of connecting again to the devices. Algorithm 3 describes the final version of the scripts.

Input: $address$, $index$, $device\_name$, $reading\_time \leftarrow 10$
$isee \leftarrow Isee\_manager(address)$
`// Connect to device and read data`
$isee.find\_and\_connect()$
$isee.start\_notification()$
$data \leftarrow isee.read\_data(reading\_time)$
$isee.stop\_notifications()$
`// Disconnect and save data`
$isee.disconnect()$
$agg\_data \leftarrow aggregate\_data(data)$
$agg\_data.to\_csv("Partials\_\{device\_name\}/partials\_\{index\}.csv")$

Algorithm 2: Script for single reading

Input: $address$, $device\_name$, $charging \leftarrow False$, $timeout \leftarrow 180$
$index \leftarrow 0$
$timeout\_res \leftarrow False$
while *timeout_res is False* **do**
    `// Call the single recording script`
    $cmd \leftarrow single\_reading(address, index, device\_name)$
    $timeout\_res \leftarrow subprocess(cmd, timeout)$
    `// If charging stop after some cycles`
    $index \leftarrow index + 1$
    if $charging \ \& \ index \geq K$ **then**
        break
    end
end
`// Process partial readings`
$partial\_list \leftarrow get\_partials("Partials\_\{device\_name\}")$
return $aggregate(partial\_list).as\_dataset()$

Algorithm 3: Final script for data recording

2.1.2 Data Preprocessing

The recorded data were organized in a typical machine learning matrix, in which each row is a sample and each column is a feature. The raw features extracted from the sensors, reported in the table 1, have been enriched with additional data:

- **cycle_id** is the ID of the battery charge/discharge cycle.
- **last_3_cycles** is a binary variable that takes a value equal to 1 in the last three cycles before the smart glasses stop sending data, 0 otherwise.
- **profile_time** is the time required in a specific cycle for the charge/discharge of the battery.
- **RUL** is the target variable of the data set, calculated according to some assumptions that are better described later.

The above features have been designed under the hypothesis that, in order to detect a degradation of the battery, sequences of charging/discharging cycles could be particularly informative; in particular, a decrease of the duration of the battery over time was expected.

2.2 Second Platform

In addition to ISee, a second platform was built, designed specifically for the predictive maintenance task applied on the batteries. The differences with respect to the ISee are:

1. A higher battery capacity (80 mAh versus 47 mAh of ISee).
2. The possibility of choosing the discharge load attached to the batteries to simulate different usage conditions.
3. Different sensors, designed for battery monitoring.

The data set is composed of the features reported in Table 3.

Feature	**Description**
V_batt	battery voltage
V_cc	circuit voltage supply
T_soc	temperature of recording board
T_imu	temperature of the IMU
T_ntc	temperature of the battery
Flag	flag that states if the battery is charging or discharging
I_discharge	discharging current
acc	accelerometer along x,y,z axis
LED	to tune the discharge load
Battery ID	Id of each battery

Table 3: Features of the second platform

The data of the second platform have been recorded from December. The data recording is done automatically by dedicated hardware; when the storage of the recording board is full, the data can be read and used. Given the automatic data recording process, it was possible to collect way more battery cycles than the one collected for the ISee in much less time; table 4 contains the recorded cycle for each battery.

Battery ID	Number of battery cycles
E597	311
D4CC	318
D2E9	255
BD7A	286
667D	311
593A	224
511A	245
4A71	310

Table 4: Number of cycles collected for the second platform

The preprocessing step for these data consists of the separation and indexing of the charge and discharge cycles, a step implemented with a custom algorithm, described with Algorithm 4, which identifies the different cycles considering the peaks in battery voltage. After the separation, for each cycle, a couple of additional measures were computed:

- **Profile time**: time for charging or discharging, obtained considering a sample time of 60 seconds.
- **Capacity**: capacity of the battery in the cycle, obtained by integrating the discharging current measured in the specific cycle.

Both variables added were expected to show signs of degradation over time.

Input: $data$, $tolerance \leftarrow 0.1$, $max\_distance \leftarrow 10$,
$min\_cycle\_len \leftarrow 5$

```
// Take points close to the peaks
```

$max\_points \leftarrow List()$

$min\_points \leftarrow List()$

for $d \in data$ **do**

  if $d["V"] > data["V"].max() * (1 - tolerance)$ **then**

    $max\_points.append(d)$

  end

  if $d["V"] < data["V"].min() * (1 + tolerance)$ **then**

    $min\_points.append(d)$

  end

end

```
// Clusterize according to max distance between indices
```

$max\_centers \leftarrow cluster(max\_points, max\_distance)$

$min\_centers \leftarrow cluster(min\_points, max\_distance)$

```
// Divide cycles:  a cycle is between 2 peaks
```

$cycles \leftarrow divide\_cycle(max\_centers, min\_centers)$

```
// Label cycle as charging or discharging
```

for $c \in cycles$ **do**

  if $c["V"].begin() < c["V"].end()$ **then**

    $c.label \leftarrow "charging"$

  end

  else

    $c.label \leftarrow "discharging"$

  end

end

```
// Add index according to cycle label
```

$add\_index(cycles)$

return $cycles.as\_dataset()$

Algorithm 4: Cycle separator

3 Exploratory Data Analysis

This section focuses on presenting the results of the Exploratory Data Analysis (EDA) of the data set, considering, in particular, the relationships between the various device sensors. Simple EDA approaches, such as data visualization and correlation analysis, are typically effective to make simple, but important choices, such as feature selection, and to gain useful insights on the problem at hand.

3.1 ISee Preliminary Analysis

As initial analysis, it is interesting to evaluate the correlations between different sensors; it can be particularly useful to perform an initial selection of features: the correlation highlights those features that behave similarly and are somehow repetitive, and those measures that seem linked to the feature to predict. In Figure 1 the pairwise Pearson's correlations between the average aggregations of different features are reported. It's worth noticing that there is a negative relationship between the battery level and relative humidity, while the coefficient value between temperature and battery level is positive and, consequently, an increase of battery level leads to the increase of temperature.

Another aspect that has been deepened is the fact that the duration of the battery should decrease with time, as is possible to see in Figure 2: the discharge times, in Figure 2a seem to have a stable decreasing tendency, while the charging times, in Figure 2b, seem to have a less defined behavior.

To complete the exploratory analysis, it is usually useful to plot the samples in a 2D environment; this could highlight some patterns of the data and could give an initial idea about the complexity of the task. There are different ways to reduce a data set from R^n to R^2, a common one being Principal Component Analysis (PCA). PCA is a dimensionality reduction technique that can be useful for providing a better overview of a multidimensional dataset; PCA is a way to summarize a large number of features into a smaller set of dimensions, called principal components. The obtained

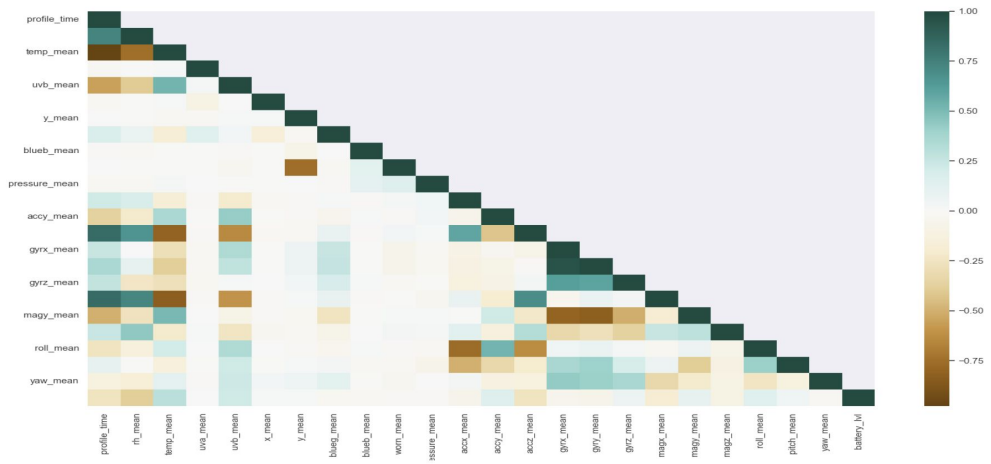


Figure 1: Heat map of Pearson's correlation coefficient between the aggregated features.

principal components are linear combination of the features that should preserve the data variability as much as possible. In Figure 3 an example of 2D PCA applied to a single ISee device is reported. The different colors highlight the distribution of the cycles in the PCA space.

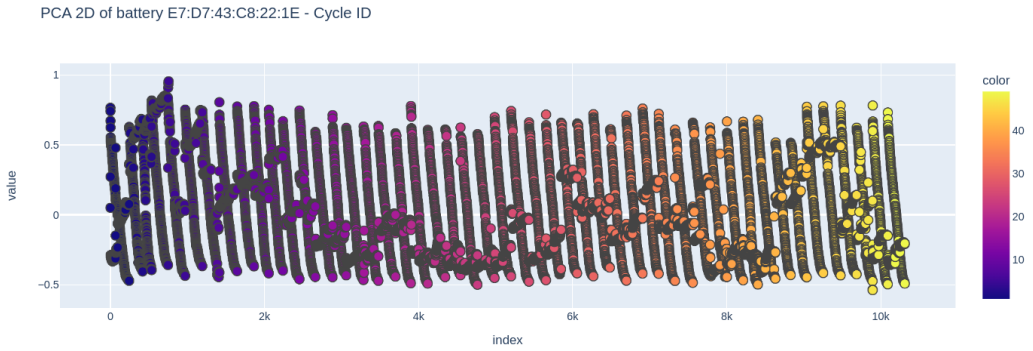
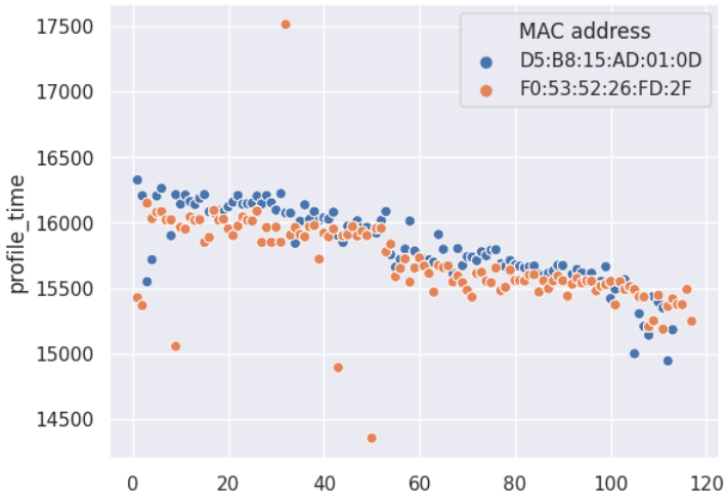
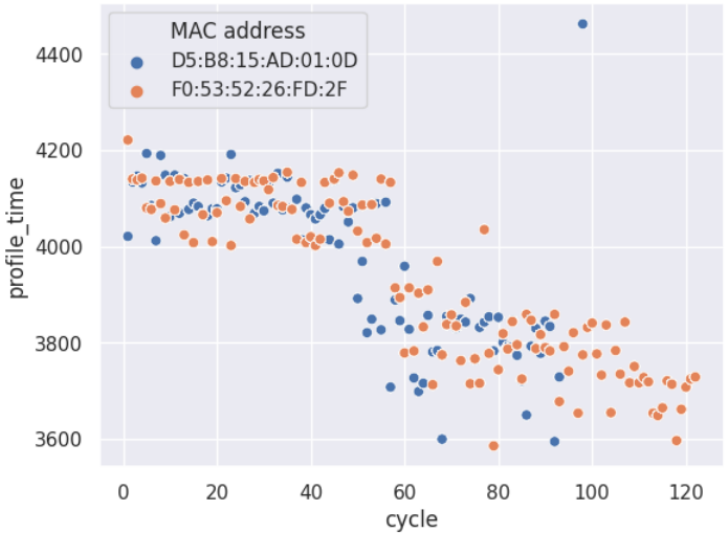


Figure 3: 2D PCA plot of a single ISee device



(a) Duration of discharge cycles within the time



(b) Duration of charge cycles within the time

Figure 2: Analysis of the relationship between duration of battery and time.

3.2 Second Platform Preliminary Analysis

For the second platform, an EDA similar to the one implemented for ISee is performed. The correlation analysis, shown in Figure 4, does not highlight any strong correlation between the sensors, except for the temperatures, which are quite correlated. In Figure 5 it is possible to see a 2D PCA plot of all the batteries of the second platform; it is clear that the batteries behave in similar ways and it can also be noticed that the distribution of the cycles seems to have a well-defined pattern.

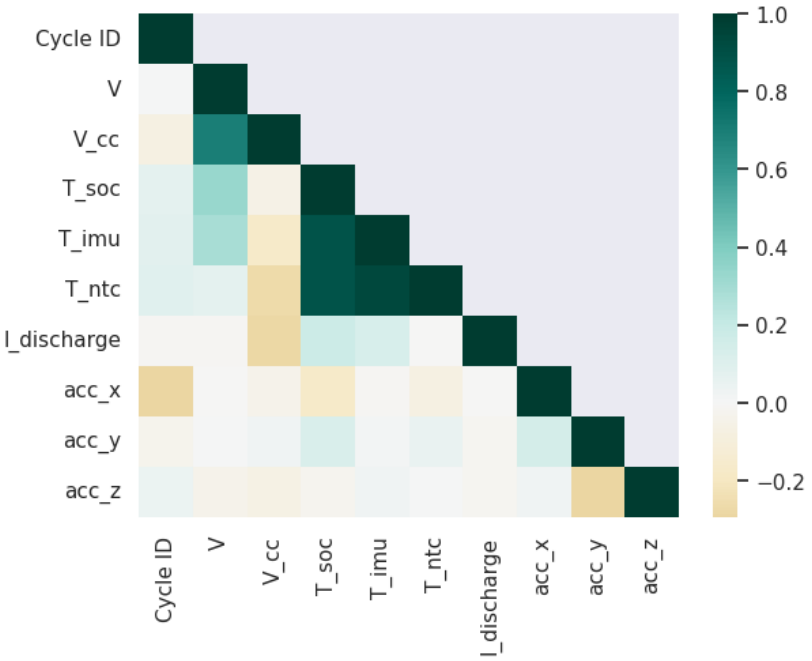
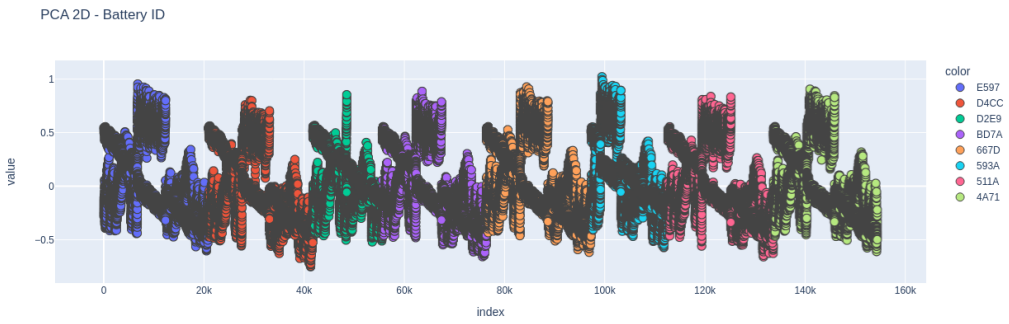
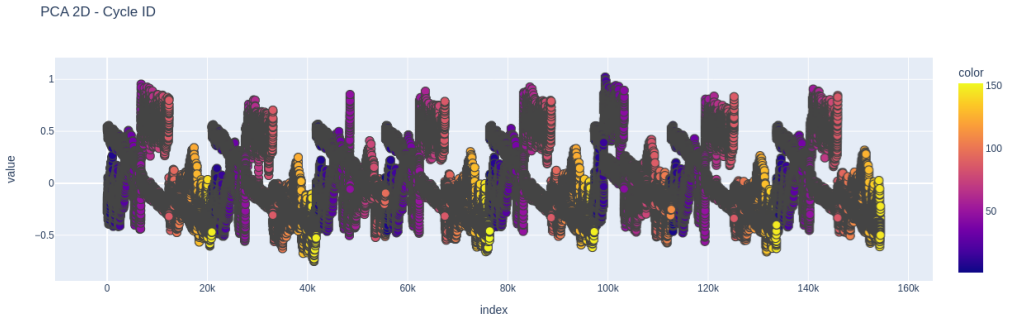


Figure 4: Pearson's correlation between features

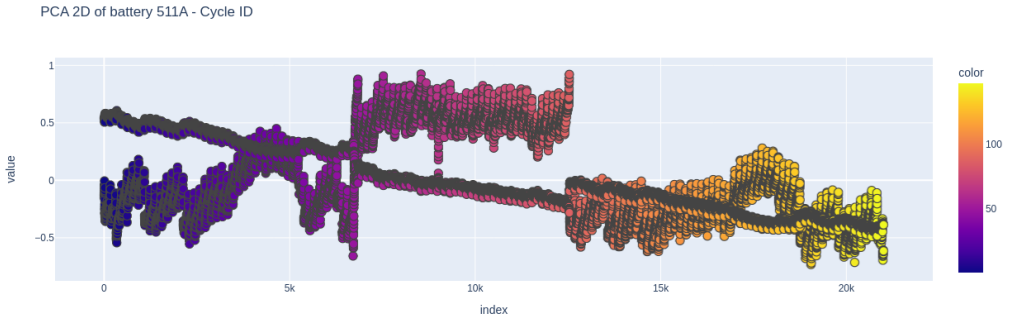
To better evaluate the degradation pattern, battery capacity and discharge time are considered. In Figure 6a the discharge time of a couple of batteries is plotted; it is possible to see that it tends to decrease as the cycles increase. In Figure 6b, instead, the plotted value is the capacity, that clearly decreases over time.



(a) PCA of all batteries, each color is a different battery

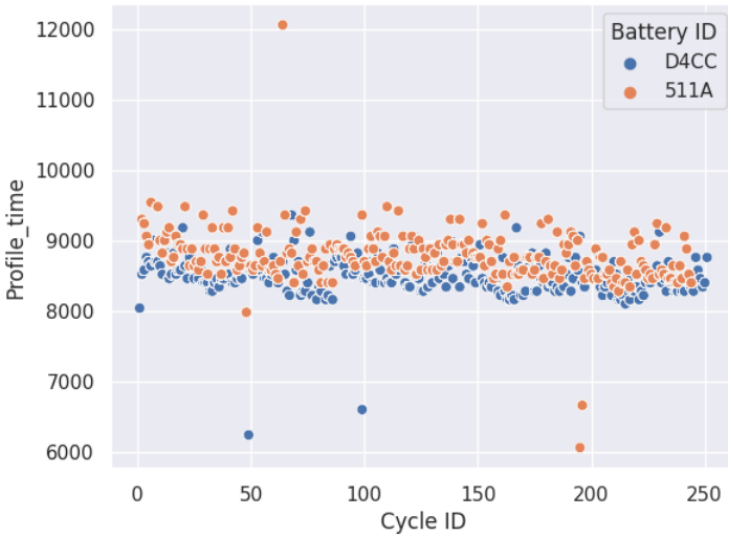


(b) PCA of all batteries, each color is a cycle

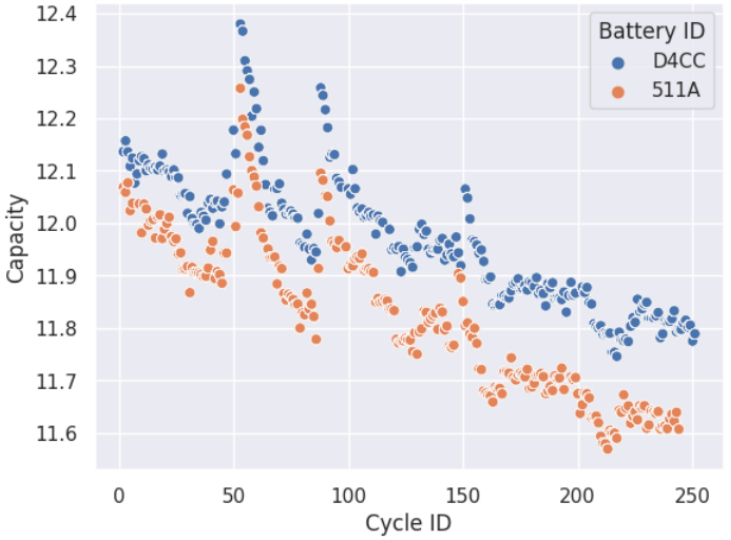


(c) PCA of a single battery, each color is a cycle

Figure 5: PCA plots of second platform



(a) Discharge time of a couple of second platforms



(b) Capacity of a couple of second platforms

Figure 6: Relationship between duration of battery, capacity and time

4 Machine Learning Background

Machine Learning (ML) is the branch of Artificial Intelligence that studies how to teach a computer to learn from its experience. The process of learning consists of the transformation of experience into new knowledge. When considering a machine, the goal of ML is to extract from the input data, representing the experience, some kind of knowledge that can be used to create a program capable of performing a specific task. [1]. Machine learning is very useful in a lot of different scenarios; it is quite effective, in particular, when the task to be solved is very complex or when adaptability is needed:

- **Task too complex to code**:
  - *Tasks performed by humans*: for example driving or walking, so tasks that are quite common for living entities, but still are very difficult to be mathematically described or implemented in a computer program.
  - *Task too complex for humans*: for example, tasks that involve the analysis of large amounts of complex data, in which the capabilities of a machine clearly outperform those of a human one.
- **Adaptability**: one of the main drawbacks of classical coding is that the program, once written and installed, will remain unchanged. In practice, however, there are a lot of fields that evolve over time. Machine learning can be built to adapt to those changes autonomously.

[1] An extensive treatment on Machine Learning theory can be found in the book "Understanding Machine Learning" [6]

4.1 Formal Definitions

In this section, some formal models for learning are presented and briefly analyzed; the discussion starts with simpler and restricted models, then it concludes with a formal definition that most correctly describes the learning task.

4.1.1 Empirical Risk Minimization

An initial definition of a learning task would be the **Statistical Learning model**; it consists of 6 ingredients:

1. **Domain set X**: set of all possible objects to make predictions about. Usually, a point $\vec{x} \in X$ is represented as a vector of features.
2. **Label set Y**: set of all possible labels.
3. **Training data**: $S = \{(\vec{x_1}, y_1), ..., (\vec{x_m}, y_m) | (\vec{x_i}, y_i) \in X \times Y\}$
4. **Learner's output h**: also called predictor, it is the function $h : X \mapsto Y$.
5. **Data-generation model**: all data are generated according to a probability distribution D and labeled according to the function $f : X \mapsto Y$. Both D and f are not known by the learner.
6. **Measure of success**: probability that the predictor does not predict the correct label on a random data point generated by the distribution D. Usually, to measure the performances, a measure of failure, called loss, is used.

Given these ingredients, the Empirical Risk Minimization (ERM) framework aims to find the predictor h^* that minimizes the loss over the entire distribution D, i.e. the predictor that returns the least possible number of bad predictions. Since that D is not known, ERM tries to achieve its goal by minimizing the training loss, that is the measure of the errors of the predictor on the training set S. Even if this approach seems quite effective, in

practice it will probably fail: by minimizing the training loss it is likely that the found predictor will perform incredibly well over the training data, but quite bad on other unseen data. This phenomenon is called **Overfitting**.

A common approach to deal with overfitting is to reduce the possible choices of the predictor h to a specific set H of finite dimension: the fact that the set is finite reduces the possibility of finding a predictor that perfectly represents the training data, but, on the other hand, it also reduces the possibilities of finding the best general predictor. Given the restriction on H, there are now two assumptions that are needed to make learning possible:

- **Realizability**: there exists $h^* \in H$ such that the loss of h^* given D and f is 0.
- **Independently and Identical Distributed** (i.i.d): all samples in the training set are i.i.d according to D.

In the next section it will be explained if it is now possible to learn $h^* \in H$.

4.1.2 Probably Approximated Correct Learning

The Probably Approximated Correct (PAC) learning derives from the following considerations: since the training set derived from D can only capture a part of the distribution, the predictor can only be:

- **approximately correct**
- **probably correct**

A simplified version of the PAC learning theorem is the following:

Theorem (Simplified PAC Learning) *Let H be a finite hypothesis class. Let $\delta \in]0,1[$, $\epsilon \in]0,1[$ and $m \in N$ such that $m \geq \frac{\log(|H|/\delta)}{\epsilon}$. Then for any f and any D for which the realizability assumption holds, with probability $\geq 1-\delta$ we have that for every ERM hypothesis h_s it holds that $L_{D,f}(h_s) \leq \epsilon$, where $L_{D,f}$ is the loss according to D and f.*

This simplified PAC learning theorem states that, if there are enough samples given a hypotheses class H, it is possible to find a predictor that is probably (according to δ) a good approximation (according to ϵ). In practice, however, the assumptions on which this theorem is based are too strong:

- The realizability assumption usually does not hold.
- It is more realistic to consider the labeling function as non-deterministic.

The non-deterministic labeling function can be formalized as follows: consider D a probability distribution over $X \times Y$. D is a joint distribution over the sets of points and labels; it can be seen as a composition of 2 distributions:

- $D_{\vec{x}}$ is the marginal distribution over data points.
- $D((\vec{x}, y)|\vec{x})$ is the conditional distribution over labels for each data point.

Given this relaxation, now the label y of point $\vec{x}$ is obtained according to the conditional probability $P[y|\vec{x}]$.

The second component of the statistical learning that needs to be deepened to better apply the PAC learning in practice is the concept of loss function:

Definition (Loss Definitions) *Given any hypotheses set H and some domain Z, a loss function is any function $l : H \times Z \mapsto R_+$. From this definition, we can formalize:*

- ***Risk function**: expected loss of hypotheses $h \in H$ with respect to D over Z. $L_D(h) = E_{z \sim D}[l(h, z)]$*
- ***Empirical risk**: empirical loss on a given sample $S = \{z_1, ..., z_m\} \in Z^m$. $L_s(h) = \frac{1}{m} \sum_{i=1}^{m} l(h, z_i)$*

Finally, the Agnostic PAC Learning theorem for generalized losses can be defined:

Theorem (Agnostic PAC Learning for Generalized Losses) *A hypothesis class H is agnostic PAC learnable with respect to a set Z and a loss function $l : H \times Z \mapsto R_+$ if there exist a function $m_H : (0,1)^2 \mapsto N$ and a learning algorithm such that for every $\epsilon, \delta \in]0,1[$, for every distribution D over Z, when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d examples generated by D the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ over the choice of the m training examples: $L_D(h) \leq \min_{h' \in H} L_D(h') + \epsilon$, where $L_D(h) = E_{z \sim D}[l(h, z)]$*

It is worth noticing that the formalization of the PAC learning by Leslie Valiant was met with a Touring award in 2010.

The function $m_H : (0,1)^2 \mapsto N$ determines the sample complexity of learning H, i.e., it determines the minimum dimension of the training set needed for a PAC solution. m_H is a function of ϵ and δ, the accuracy and confidence parameters, but it also takes into account H. Given that there are multiple functions that satisfies the definitions of PAC learning, m_H is to be considered the minimum function that satisfies the requirements.

Corollary *Every finite hypothesis class H is learnable with sample complexity $m_H \leq \lceil \frac{log(|H|/\delta)}{\epsilon} \rceil$*

4.1.3 The No Free Launch Theorem

Given the formalization of PAC learning a question might arise: *is it possible to implement an algorithm A that, given the training set S, always returns the best predictor h^* for any possible distribution D?* The No Free Launch Theorem answers this question:

Theorem (No Free Launch Theorem) *Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain X. Let m be any number smaller than $\frac{|X|}{2}$, representing the size of the training set. Then, there exists a distribution D over $X \times \{0,1\}$ such that:*

- *There exists a function $f : X \mapsto \{0,1\}$ with $L_D(f) = 0$.*
- *With probability of at least $\frac{1}{7}$ over the choice of $S \sim D^m$ we have that $L_D(A(S)) \geq \frac{1}{8}$.*

Note that there are similar results for other learning tasks.

The No Free Launch Theorem proves that *it cannot exists a learner that always succeed on every task*, which means that, considering a learner, there will always be a task in which it returns a predictor that performs worse than another predictor found by another learner.

A corollary of the No Free Launch theorem states that:

Corollary *Let X be an infinite domain set and let H be the set of all functions $X \mapsto \{0,1\}$, then H is not PAC learnable.*

This corollary, in combination with PAC learning, highlights a trade-off:

- The cardinality of H should be large enough to include the predictor with the least achievable loss.
- H cannot be the set of all functions in a given domain, otherwise learning will fail.

So it is important to choose a hypothesis set big enough in order to find a good model, but small enough in order to prevent overfitting. A better

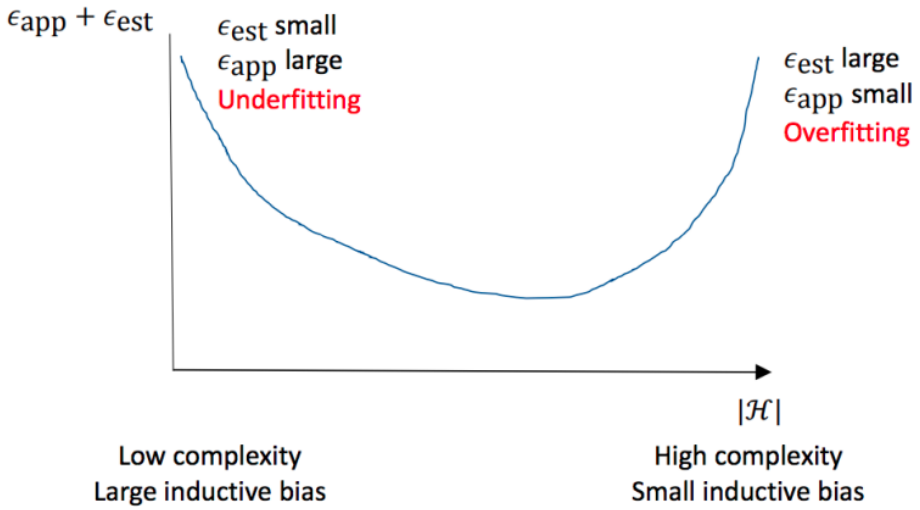


Figure 7: Complexity trade-off

analysis of this trade-off can be performed by decomposing the loss. When considering the predictor h_s and its loss $L_D(h_s)$, it is possible to rewrite the loss as the sum of two terms: $L_d(h_s) = min_{h \in H} L_D(h^*) + L_D(h_s) - min_{h \in H} L_D(h^*) = \epsilon_{app} + \epsilon_{est}$

- $\epsilon_{app} = min_{h \in H} L_D(h^*)$ is the **approximation error**, the best achievable error considering H: it is unavoidable and derives from the choice of H. Increasing H is the only way to reduce it. A large approximation error means that the chosen hypothesis set is not powerful enough to approximate D, this is called *underfitting*.

- $\epsilon_{est} = L_D(h_s) - min_{h \in H} L_D(h^*)$ is the **estimation error**, the error of the current hypothesis considering the best possible predictor in H: it comes from the inability to find the best model in H; decreasing the cardinality of H is the only way for reducing it. A large estimation error means that the model is too powerful, fits the training data too well, but cannot generalize, producing *overfitting*.

A visual representation of this trade-off is reported in Figure 7

4.1.4 Regularization

In practice, control over the choice of H may not be sufficient to correctly manage the overfitting problem, so other learning paradigms, such as **Regularized Loss Minimization** (RLM), were proposed.

The idea of this different learning paradigm is to better manage the trade-off between approximation error and estimation error; to do this, a regularization function is used as penalty for models that are too complex: by setting a good regularization function it is possible to force the learner to choose a simpler predictor with respect to the one that ERM would chose; it will have a worse empirical loss, but it should generalize better.

Definition (Regularized Loss Minimization) *Assume that $h \in H$ can be defined by a vector of parameters $h = (w_1, ...w_d) \in R^d$ and consider a regularization function $R : R^d \mapsto R$, the Regularized Minimization Loss consists in choosing h such that $h = argmin_w(L_s(w) + R(w))$, where $R(w)$ is a measure of the complexity of predictor h.*

Two common regularization functions are:

- l_1 regularization: $R(w) = \lambda||w||_1$, where $||w||_1 = \sum_{i=1}^{d} |w_i|$ is the l_1 norm.
- l_2 regularization: $R(w) = \lambda||w||^2$, where $||w||^2 = \sum_{i=1}^{d} w_i^2$ is the l_2 norm.

In both these functions, it is possible to balance the trade-off between approximation and complexity by choosing a good parameter λ: a huge λ means that the complexity is highly penalized, thus the paradigm will return a quite simple model; on the other hand, a λ close to 0 means that the regularization is "disabled" and the RLM will behave similarly to the ERM.

Figure 8 reports a simple example of regularized vs. non-regularized learning.

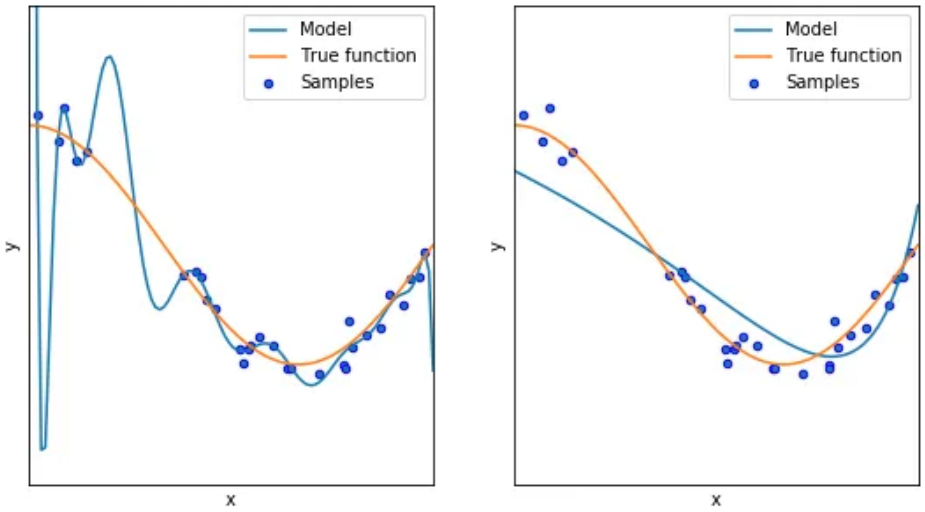


Figure 8: Non-regularized (left) vs. Regularized (right) learning

4.2 Learning Approaches and Tasks

Machine learning tasks can be of different natures and with different goals, but it is possible to divide them into some subfields; the relevant classes for this work are:

- **Supervised Learning**: it is the most classical way of learning, each sample of the training set is correctly labeled. The learning algorithm will return a predictor that approximates the unknown labeling function. Two classical supervised tasks are:

  1. *Regression*: the labels are in R.
  2. *Classification*: the labels are in a discrete set and represent possible classes.

  Different losses are used for the different supervised tasks.

- **Semi-supervised learning**: It is similar to supervised learning, but in this case only a portion of the training set is labeled. In practice, labeling is an expensive operation, so semi-supervised learning can be quite interesting. A way of dealing with this learning approach is to train a model with the available labeled data, then use this model

to pseudo-label the unlabeled training samples. The pseudo-labeling should be done according to a threshold on the confidence of the model; if the threshold is too low the learning will fail.

- **Unsupervised learning**: In this learning approach no label is available, the goal of the model is to capture underlying patterns and connections between the data points. The tasks achievable with unsupervised learning are quite different from the supervised ones, for example:

  1. *Clustering*: group data in clusters such that similar samples are in the same cluster and dissimilar samples are in different clusters.
  2. *Anomaly Detection*: find samples in the training set that do not conform well to other data points.

- **Self-supervised learning**: it is a type of learning that lies between supervised and unsupervised; it aims to learn from a data set without labels (unsupervised learning) while trying to solve a supervised task, such as classification or regression. It is also useful as pre-training step for classical supervised tasks, in which case the self-supervision is used to learn, in an unsupervised way, a good feature representation that can then be used as input of the supervised model.

4.3 A simple task: Linear Regression

Linear Regression is a very good and basic example of machine learning task; it is a supervised learning approach in which the hypothesis class H is limited to the linear models. Mathematically, H is defined as: $H = \{h_{\vec{w},b} | \vec{w} \in R^d, b \in R\}$. A model of the linear class is: $h_{\vec{w},b}(\vec{x}) = (\sum_{i=1}^{d} w_i x_i) + b$, where $\vec{w}$ is the vector of weights, b is the bias and $\vec{x} \in X \subseteq R^d$ is an input sample. Given $\vec{x}$, $\vec{w}$ and b, it is convenient to represent the samples in a homogeneous space:

- $\vec{x}' = (1, x_1, ..., x_d) \in R^{d+1}$
- $\vec{w}' = (b, w_1, ..., w_d) \in R^{d+1}$

To complete the definition of the learning task it is important to define the loss that will be minimized according to the ERM framework; a common choice in regression tasks is the squared loss:

- Loss: $l(h, (\vec{x}, y)) = (h(\vec{x}) - y)^2$
- Empirical risk function: $L_S(h) = \frac{1}{|S|} \sum_{i=1}^{|S|} (h(\vec{x}) - y)^2$

According to ERM, the solution of this task is the learner that better fit the training set, that is: $h^* = argmin_{\vec{w}} L_S(h_{\vec{w}}) = argmin_{\vec{w}} \frac{1}{|S|} \sum_{i=1}^{|S|} (h_{\vec{w}}(\vec{x}) - y)^2$. Given the simplicity of the task, the *Least Squares Algorithm* is able to solve in a closed form the problem: it considers the gradient of the Residual Sum of Squared (RSS) derived from the ERM formulation of the task, then, by forcing the gradient to be equal to 0, the algorithm returns the best possible predictor. It is an efficient and well defined algorithm, whose only drawback is the inversion of a matrix coming from the gradient formulation, but this can be solved by considering the pseudo-inverse or other mathematical tricks.

There also exist algorithms that can solve the regularized version of the Linear Regression, for example the *Ridge Regression Algorithm*. Given the definition of the linear regression task, the ridge regression algorithm goal is to find the predictor that better fit the training data according to the regularization function, that in this case is the l_2 *regularization*: $h^* = argmin_{\vec{w}}(RSS(\vec{w}) + \lambda||\vec{w}||^2)$, where $RSS(\vec{w})$ is the same formulation of the ERM problem. Even in this case, the predictor can be found in a closed-form solution, similarly to the Least Squares algorithm. It is worth noticing that, in this second case, the problem of the inversion of the matrix does not exist, in fact it can be proven that the regularization factor makes the matrix always invertible. There are also algorithms that support the l_1 regularization, like the *LASSO regression*.

4.4 Gradient Based Optimization Techniques

When considering more advanced hypothesis sets and models, the existence of a closed-form solution is not guaranteed. Optimization algorithms implement different methods for minimizing a complex function; they can be based on different mathematical concepts and can be able to find local, and in some case even global, minima for very different functions. The most famous optimizators in machine learning are based on the **gradient**.

4.4.1 Gradient Descent

The Gradient Descent algorithm (GD) is a general optimization technique useful for minimizing differentiable functions. It is a rather simple algorithm, but it works very well and provides some theoretical guarantees that makes it one of the fundamental block of learning. The GD is based on the definition of Gradient:

Definition (Gradient) *Let $f : R^d \mapsto R$ be a differentiable function. The gradient $\nabla f(\vec{w})$ of f at $\vec{w} = (w_1, ..., w_d)$ is defined as the vector of the partial derivatives of f, that is: $\nabla f(\vec{w}) = (\frac{\partial f(\vec{w})}{w_1}, ..., \frac{\partial f(\vec{w})}{w_d})$*

The intuition behind the definition is that the gradient is a vector that points in the direction of the greatest rate of increase of $f(\vec{w})$ according to $\vec{w}$; the GD algorithm is exactly based on this intuition and the idea is to move in the opposite direction of the one pointed by the gradient: the greatest rate of decrease. The algorithm is reported here 5

Input: $\eta \in R_+$, $T \in N$
$\vec{w}^{(0)} \leftarrow \vec{0}$
for $t \leftarrow 0$ *to* $T - 1$ **do**
  $\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta \nabla f(\vec{w}^{(t)})$
end
return $\vec{w}' = \frac{1}{T} \sum_{t=1}^{T} \vec{w}^{(t)}$

Algorithm 5: Gradient Descent

Notes:

- The initialization of $\vec{w}^{(0)}$ is usually a *random initialization*.
- In Machine learning the function $f(\vec{w})$ is the empirical risk.
- The output can be the mean of the vectors $\vec{w}$, useful in case of non-differentiable functions (subgradient), but also the last vector $\vec{w}^{(T)}$ or other options.
- The parameter η is called **learning rate**, it is used to calibrate the rapidity of the convergence of the algorithm: a learning rate that is too small will result in a very slow convergence (the algorithm will require too much time to fit), but a learning rate that is too high will result in no convergence at all (the weights will jump between local minima without been able to reach one). η can vary according to some strategies during iterations.

The main drawback of the GD algorithm is its time complexity: the computation of the gradient considers all samples in the training set, resulting in an inefficient algorithm when dealing with large datasets. In these case it is better to rely on approximations of the gradient.

4.4.2 Gradient Descent Approximations

A famous alternative to the GD algorithm is **Stochastic Gradient Descent**, an algorithm that relies on an approximation of the gradient instead of considering its exact formulation. This allows to obtain a much faster optimizer that can still be applied in machine learning. It is also possible to prove that the approximation considered by SGD allows to directly minimize the true loss L_D without the need to consider its estimation, the empirical loss L_S. To minimize the unknown true loss, all that is needed is an estimate of the gradient, so a vector whose expected value is equal to $\nabla L_D(\vec{w})$: $\nabla L_D(\vec{w}^{(t)}) = \nabla E_{z\sim D}[l(\vec{w}^{(t)}, z)] = E_{z\sim D}[\nabla l(\vec{w}^{(t)}, z)] = E[\vec{v}_t|\vec{w}^{(t)}]$, where $\vec{v}_t$ is defined as the gradient of the loss with respect to $\vec{w}$ in point $\vec{w}^{(t)}$. It turns

out that the gradient of the loss function in $\vec{w}^{(t)}$ is an estimation of the gradient of the true loss $L_D(\vec{w}^{(t)})$ that can be efficiently computed by sampling a random point $z \sim D$ at each iteration t; since the training set S is, by assumption, generated by the same distribution D, the sampling can be performed by picking a random point in the training set S. The Algorithm of SGD is reported here 6

Input: $\eta \in R_+$, $T \in N$
$\vec{w}^{(0)} \leftarrow \vec{0}$
for $t \leftarrow 0$ *to* $T-1$ **do**
  sample a single data point $i \in S$
  `// Compute gradient of the loss in i`
  $\vec{v}_t = \nabla l(\vec{w}^{(t)}, (\vec{x}_i, y_i))$
  `// Update weights`
  $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{v}_t$
end
return $\vec{w}' = \frac{1}{T} \sum_{t=1}^{T} \vec{w}^{(t)}$

Algorithm 6: Stochastic Gradient Descent

The SGD is hence much faster than GD, shares similar theoretical guarantees with it and allows for a direct optimization of the true loss. In practice, though, the approximation of the gradient considering a single data point of the training set often results in a bad and very noisy estimation, making the convergence of the algorithm more difficult and the performances of the SGD not as good as should be in theory. To obtain a better estimation of the gradient, but still in an efficient way, it is possible to use the **Mini-Batch SGD**: instead of sampling only one point from S, sample a subset of specific size, the mini-batch, and compute the gradient of the loss for all the points in the mini-batch; the final gradient is the average of all the computed gradients. This approach leads to a much better estimation, to a faster convergence and to the possibility of using greater learning-rates, while remaining efficient to

be used in practice. The dimension of the mini-batch can be adjusted considering the dimension of the dataset. It is worth noticing that this approach can be easily parallelized, making the algorithm very convenient to be used with GPUs. This algorithm, described in Algorithm 7, is the base for all the optimizers used in Deep Learning.

Input: $\eta \in R_+$, $T \in N$
$\vec{w}^{(0)} \leftarrow \vec{0}$
for $t \leftarrow 0$ *to* $T-1$ **do**
  sample a batch of B data-points from S
  `// Compute gradient of the loss in the batch`
  $\vec{v}_t = \frac{1}{B}\sum_{k=1}^{B} \nabla l(\vec{w}^{(t)}, (\vec{x}_k, y_k))$
  `// Update weights`
  $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{v}_t$
end
return $\vec{w'} = \frac{1}{T}\sum_{t=1}^{T} \vec{w}^{(t)}$

Algorithm 7: Stochastic Gradient Descent with mini-batches

SGD is still a fundamental optimizer, but it is worth noticing that there exist more advanced approaches based on mini-batch SGD that exploit other types of mathematical estimation in order to improve the convergence properties of the algorithms. The most used optimizer in Deep Learning, for example, is Adam, a method that implements a quite robust update rule. Figure 9 shows different optimization techniques.

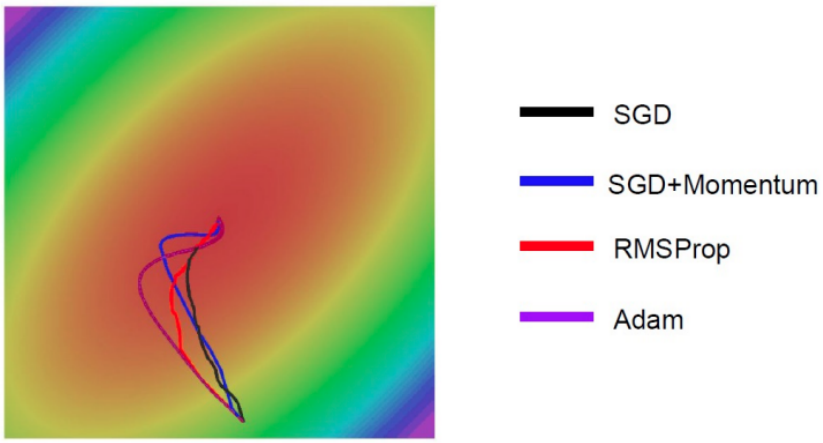


Figure 9: Comparison of different optimization techniques

4.5 Neural Networks and Deep Learning

Neural networks (NNs) are currently the most promising machine learning models in several fields, in particular in Computer Vision and Natural Language Processing. Even if the interest for this type of model is rather recent, neural networks were conceived between 1940 and 1970, but, at that time, technology was not ready for such advanced predictors. There are mainly four reasons why NNs are a very good choice in machine learning:

- **Performances**: NNs can be extremely powerful models that can achieve very low losses in a lot of tasks.
- **End-to-end learning**: the classical ML pipeline expects at least an initial step of feature extraction that should produce informative vectorized inputs for the model; this step is usually engineered and supervised by humans. With NNs, in Deep Learning in particular, it is the model itself that learns how to extract the features it needs, without any type of human supervision. Figure 10.
- **Community interest**: the research community is very active in this field and that brings many new ideas, new libraries and new concepts that keep improving NNs.

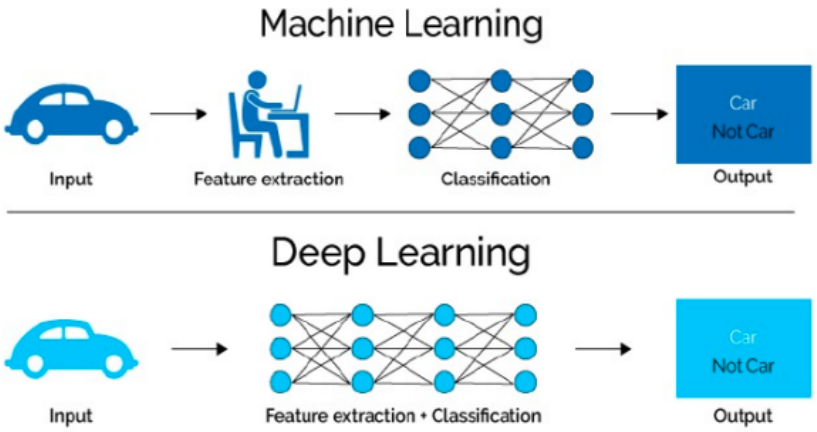


Figure 10: End to end learning

- **Flexibility**: thanks to advanced architectures, especially in deep learning, NNs are very flexible and can easily adapt to very different kinds of data, while remaining very powerful predictors.

Along with all these nice properties, there are also some drawbacks that actually explain why it took so long for NNs to become the famous models that are now:

- **Very complex models**: NNs can be very complex models, difficult to train, difficult to interpret and often not fair.
- **Need of huge datasets**: NNs need a lot of data to be trained (even if transfer-learning reduces this problem), this is one of the reasons why in the past this type of model didn't work well.
- **Need of powerful hardware**: the complexity of the model and the need of huge datasets result in the need for powerful hardware for the train, such as GPUs. This was another bottleneck in the past.

4.5.1 Basics of Neural Networks

The basic block of the NNs is the Neuron, formalized in 1958 by Rosenblatt. Inspired by human neurons, its formulation resembles what was known about the brain: a neuron takes in input different signals, if the signals are strong enough, the neuron fires an output signal to its connections. The mathematical formulation of Rosenblatt, the *Perceptron* is defined as follow:

Definition (Perceptron) $n(\vec{x}) = \sigma(\sum_{i=1}^{n} w_i x_i) = \sigma(z)$, *where* σ, *called activation function, is the threshold function* $\sigma(z) = \{1$ *if* $z >$ *threshold,* 0 *otherwise*$\}$ *and* $\vec{w}$ *is the vector of weights, the learnable parameters that characterize the input connections.*

The problem of the perceptron formulation is the linearity of the activation function: regardless of the complexity of the model, if all its activation functions are linear, it will only be able to produce linear decisions (as shown in the Figure 11). So, in the currently used perceptron formulation, the activation function is usually replaced by nonlinear functions, such as *ReLU, Sigmoid, Tanh...* Figure 12 shows the structure of the artificial neuron.

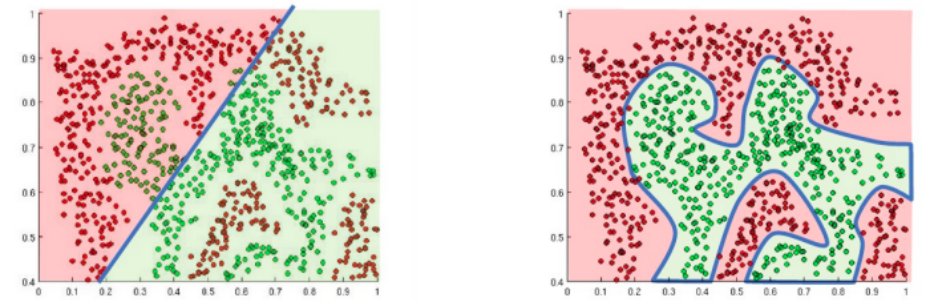


Figure 11: Linearity versus non-linearity

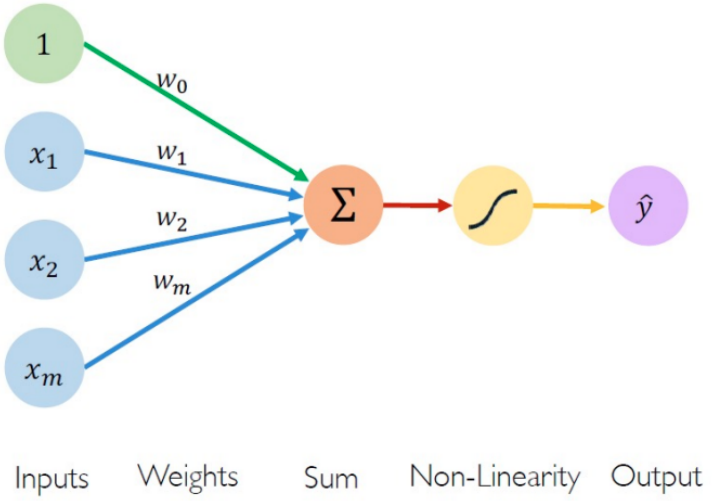


Figure 12: Artificial Neuron

A Neural Network is just a connection of multiple neurons; in its basic form, the feedforward neural network, it is a direct acyclic graph organized in layers: the first layer is the input one, its dimension is equal to the length of the input vector, the last layer is the output one, its dimension depends on the task. The middle layers are called hidden layers and, intuitively, their goal is to perform feature extraction: more hidden layers means more complex features extracted. Figure 13 shows a feedforward neural network with 5 layers.

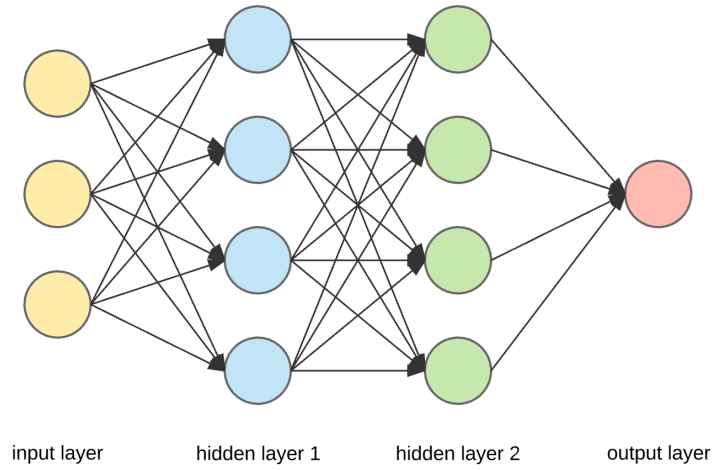


Figure 13: Feedforward neural network

A geometrical interpretation on the hidden layer could be the following: a NN with only the input and output layers (no hidden layers) can express a space that is an intersection of half-spaces; a NN with an hidden layer, instead, can express union of intersections of half-spaces. To train a NN, that is, to learn the weights of its connections, usually GD approximations, like SGD and Adam, are used; the computation of the gradient is done with the Backpropagation algorithm, proposed in 1986. It can be proven that a vanilla NN, one that has only 1 hidden layer, can approximate any continuous function to any desired precision: a NN with 3 layers is an *universal approximators*. The fact that a NN can be an universal approximator, however, does not give any type of guarantee about the feasibility of finding such approximator: to approximate a specific function, an exponential number of neurons of the hidden layer could be required. Luckily it has been observed that the same function can be compactly approximated by a network with multiple hidden layers: this observation leads to the Deep Learning field.

4.5.2 Deep Learning

Deep Learning (DL) not only studies all those networks that have multiple hidden layers, but it also studies particular architectures that implement mathematical operations that are particularly suitable for some tasks. Nowadays DL is almost a synonymous of NNs. Classical deep NNs have usually a large amount of parameters, due to the full connectivity of the neurons, and do not exploit well all the information that an input structured in a certain way could bring in it; for example, when considering images, it is better to consider a pixel alongside its neighbors than consider it alone: the structure of the input provides very good additional information that can be exploited for better predictions. Among the specialized architectures, the most famous are:

- **Convolutional**: very suitable for images; the convolution operation considers a neighborhood of pixel in an image instead of considering one pixel at the time; moreover, it is implemented with fewer parameters than fully-connected layers, but still produces better predictions. Figure 14 shows an example of 2D convolution.
- **Recurrent**: very useful when dealing with sequential data; the recurrent connection allows the layer to create its output considering not only the input in the instant t, but also the previous inputs in the sequence. Figure 15 shows the structure of a recurrent neural network.

In addition to these layers, others were proposed and are currently used, some good examples are:

- **Pooling layer**: Useful to reduce the input size by aggregating the values in a tensor of reduced dimension.
- **Dropout layer**: Useful to reduce overfitting; deactivates different random nodes at each iteration of the training, forcing the model to be not too dependent on a specific feature.
- **Normalization layer**: Useful to normalize the hidden output of the network. It can bring stability during training.

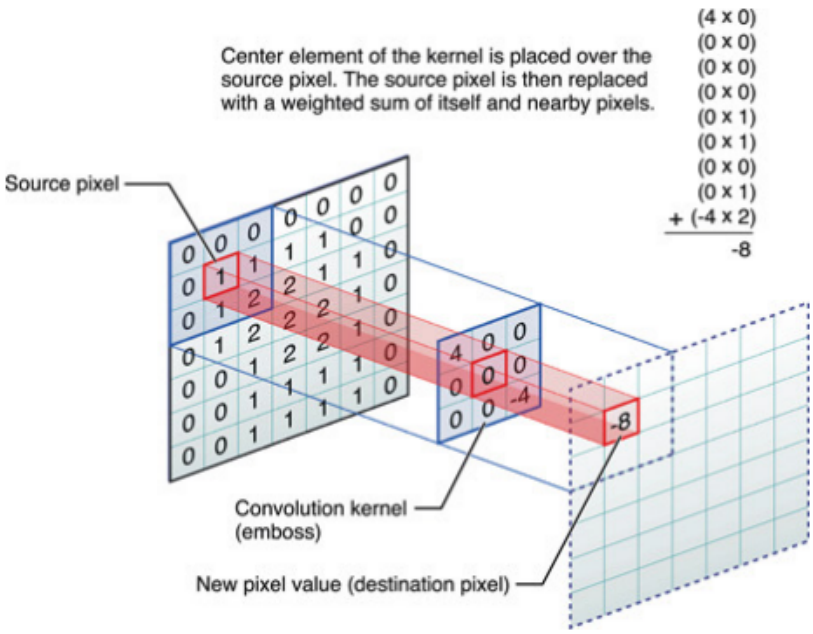


Figure 14: 2D Convolution Operation

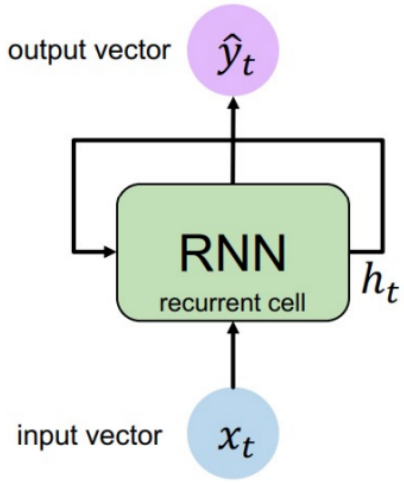
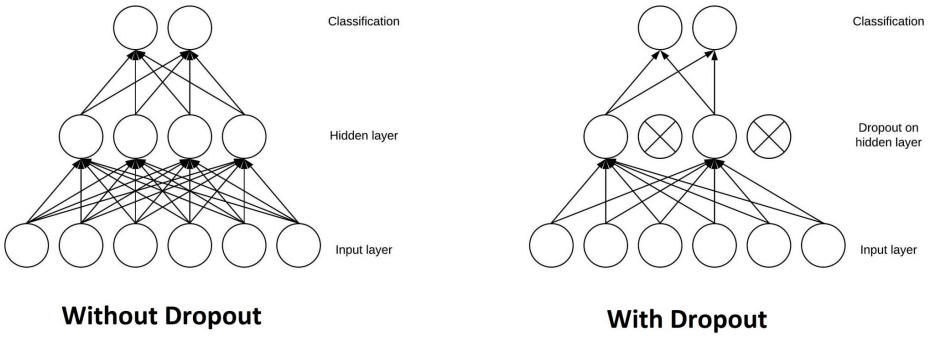
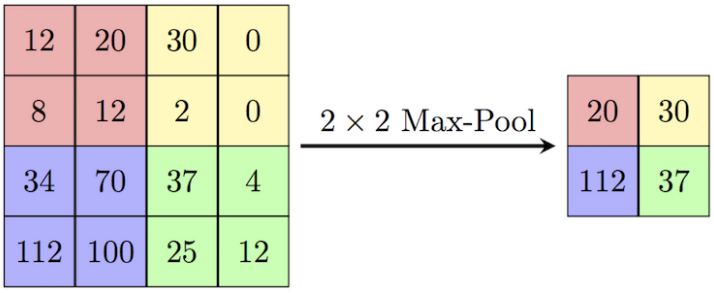


Figure 15: Recurrent structure



(a) Dropout layer



(b) Max pool layer

Figure 16: Advanced deep learning layers

4.6 Sequential Learning

Sequential learning is the subfield of machine learning that studies how to learn from sequential data, that is, from all these types of data that are composed of a series of single data points from a multidimensional space. Sequential data can be found in very different fields, like Natural Language Processing (NLP), finance, biology (for example, in DNA sequence analysis), predictive maintenance and much more. Given the sequential nature of the data, some additional challenges arise when learning:

- **Variable length of sequences**: Sequences can have different sizes, the model must be able to deal with variable length inputs.
- **Long-term dependencies**: To analyze a sequence, it is important to consider the observation at instant t given the previous observations ($t-1$, $t-2$...). The model needs memory.
- **Maintain information about order**: a sequence makes sense only in its order.
- **Parameters sharing**: The parameters of the model must be shared for each sample in the sequence.

Given these additional difficulties, a classical Feed Forward Neural Network (FFNN) is not suitable for the task: the FFNN cannot deal with different length inputs, does not have the possibility of sharing parameters and is not made to deal with long-term dependencies. The macro-architecture that was designed to deal with sequences is called **Recurrent Neural Network** (RNN), the idea is to process time step t of the sequence by considering what has happened before by mean of a recurrent relation. The recurrent relation can be expressed as follows:

Definition (Recurrent Relation) *Consider $\vec{x_t}$ as the input vector at time step t and f_W a learnable function, the recurrent relation is: $h_t = f_W(h_{t-1}, \vec{x_t})$, where h_{t-1} is the old state of the relation and h_t is the current state.*

In modern DL frameworks, an RNN accepts variable-length inputs in multiple ways:

- Process one time step at time. Effective, but very slow.
- Accept sequences that are reshaped to a unique length.
- Define padding and masking for the sequences to virtually shape them as unique length input. The padded and masked input will not be considered by the networks.

Given the recurrent relation, that gives memory to the model, and the sharing of function f_W of the recurrent relation for each sample in the sequence, all the requisites for sequential learning are respected. While RNNs seem promising, there are new problems that arise from the recurrent relation, in particular related to the training of the model:

- **Exploding Gradient**: the recurrent structure could lead to very large values of the gradient; this would compromise learning.
- **Vanishing Gradient**: the opposite of the exploding gradient, but much worse. The gradient becomes close to 0 and the model loses its ability to learn.

The vanishing gradient problem, in particular, makes it really hard for RNNs to be trained; even if it is possible to alleviate this effect by choosing the right weights initialization or good activation functions, in practice there was the need of creating specific architectures that could better manage the long-term dependencies tracking. The most famous are the gated and convolutional units.

4.6.1 Gated Units

The gated units are particular neurons that implement a modified version of the recurrent relation designed explicitly for handling long-term dependencies. The more complex recurrent relation is based on gates, that are logical operators that control how the information flows between the cell states: the gates decide which part of information to track for long time, for a short time or to discard. The most famous gated units are the Gated Recurrent Unit (GRU), represented in Figure 17a, and the Long Short Term Memory (LSTM), represented in Figure 17b. Both types of cells have auxiliary memory that is managed by the gates. Given a sample point of the sequence at instant t, $x^{<t>}$, and the memory of the cell at instant $t-1$, $c^{<t-1>}$, the GRU manages the sequence in the following way:

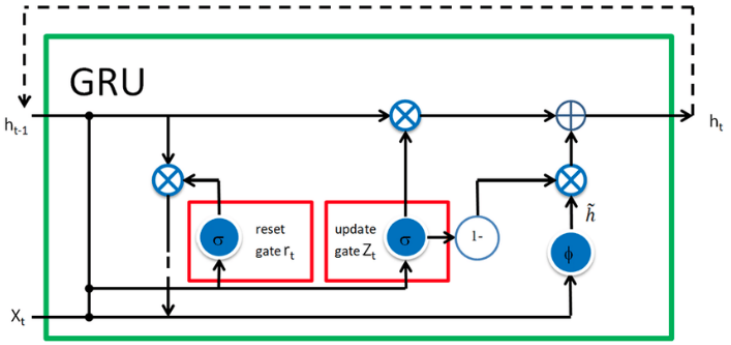
1. Compute the current state of the cell: $\tilde{c}^{<t>} : f(c^{<t-1>}, x^{<t>})$
   - $\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$
   - Γ_r is the *reset gate*: it decides how much of the old memory to forget in the generation of the new state. $\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$
2. Compute the final state at time t, $c^{<t>}$, by combining the current and previous memory
   - $c^{<t>} = \Gamma_u * c^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$
   - Γ_u is the *update gate*: it decides the percentage of old and new memory that will contribute to the final state of the cell. $\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$
3. Finally compute the output of the cell $y^{<t>}$ by using the activation function on $a^{<t>}$; in this case $a^{<t>} = c^{<t>}$

The LSTM cell increases the complexity respect the GRU by using three gates instead of two; this results in an overall more flexible architecture that has been widely used for the past years. Given in input a sample point at

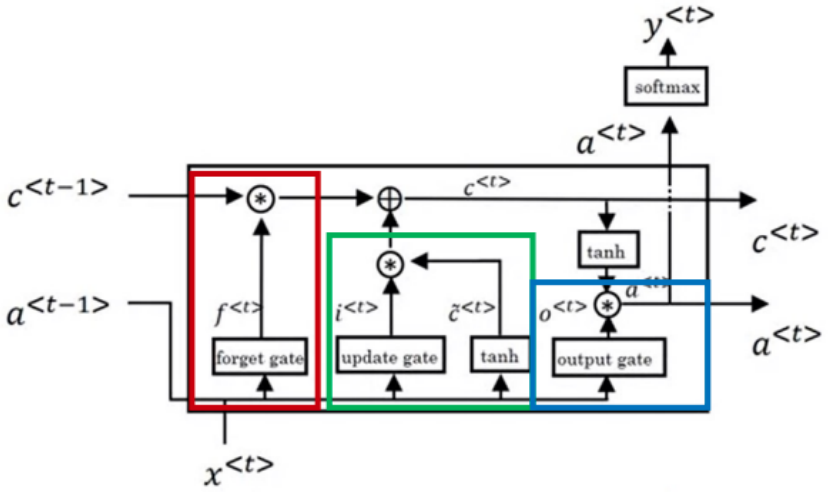
instant t, $x^{<t>}$, the old cell memory, $c^{<t-1>}$, and the old cell output before activation, $a^{<t-1>}$, the LSTM performs in this way:

1. Compute the current state $\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$
2. Set the gates as functions of $a^{<t-1>}$ and $x^{<t>}$
    - Γ_u is the *update gate*: it manages the percentage of current state that will be used to update the memory of the cell.
      $\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$
    - Γ_f is the *forget gate*: it decides the percentage of old memory that will be forgotten. $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$
    - Γ_o is the *output gate*: it tunes the strength of the cell output. $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$
3. Compute the output state of the cell $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
4. Compute the new activation $a^{<t>} = \Gamma_o * tanh(c^{<t>})$
5. Compute the cell output $y^{<t>}$ applying the activation function to $a^{<t>}$

In both layers, the gates output values in range $[0; 1]$, thanks to the Sigmoid function σ, so they behave similarly to logical gates. The idea is that the units, being capable of deciding what to remember and what to forget, will be able to better capture long-term dependencies in the sequence. The possibility of keeping old memory can also contribute to gradient computation, alleviating the problem of the vanishing gradient.



(a) GRU cell



(b) LSTM cell

Figure 17: Gated Cells

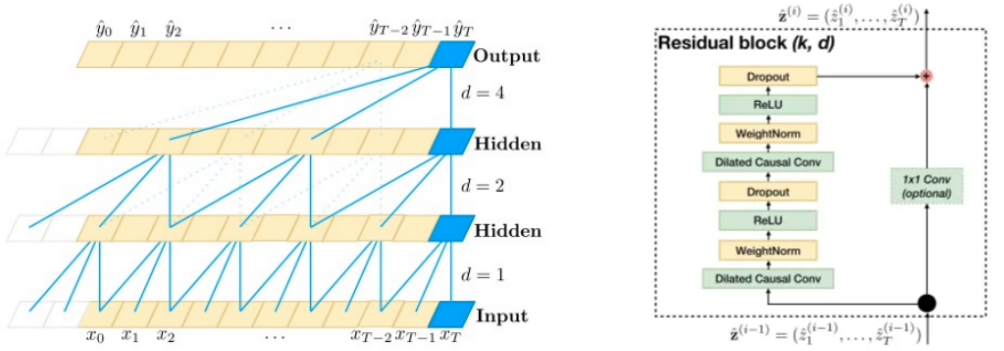
4.6.2 Convolution

Another way of dealing with sequences is to use convolution: the convolution operation naturally considers a point in a sequence alongside its neighbors, resulting in an output that does not depend on the single sample alone. It is a type of layer widely used with images, but its 1-dimensional version can be easily implemented with sequences. Since a greater kernel size of the convolution results in a greater receptive field in the sequence, it is easy to capture long-term dependencies with this type of architecture. A special type of convolution for time series is **Temporal Convolution** (TCN), pictured in

Figure 18, a causal convolution that takes advantage of dilation to perceive a large receptive field without the need to use too many parameters. It is a very good architecture, very useful for dealing with sensors data, particularly interesting for these reasons:

- TCN can manage long-term dependencies easily by increasing the kernel size or dilation, resulting in better memory than even LSTM.
- TCN does not suffer from vanishing gradient.
- TCN can be trained in parallel, better exploiting GPUs, and requires less memory than recurrent units.

Figure 18 shows the building blocks of temporal convolution.



(a) Dilated causal convolution

(b) Temporal convolution block

Figure 18: Temporal Convolution Cell

5 Unsupervised Anomaly Detection

Anomaly Detection, also known as Outlier Detection, is a task in Machine Learning whose goal is to separate a core of regular observations, the so-called *inliers*, from those that do not appear to conform well with the standard behavior. These different data points are called *outliers*.

In the next sections, the results obtained by applying the popular Isolation Forest algorithm [4] to the data set collected from a defective device are presented; the goal was to highlight possible signs of imminent breakage of the glasses considering the values of the equipped sensors.

To better analyze the results obtained, the application of two interpretation techniques used to better understand the correlation between features and anomalies is presented.

5.1 Isolation Forest

Isolation Forest [4] is an unsupervised algorithm for anomaly detection. It is an ensemble approach that aims at isolating each observation by randomly selecting a feature and then randomly splitting those feature values according to their distribution. The partitions obtained by this procedure can be represented by a tree structure; the idea of the algorithm is to classify as anomalies those data points that are easy to isolate, i.e. those points that require few partitions to be separated: considering the tree representation, the anomalies are the observations that appear in the trees at shallower depths.

The algorithm does not try to define a profile for the normal observations before the classification, instead it explicitly identifies the anomalous points, resulting in a time and memory efficient approach.

5.1.1 Isolation Forest Application

To better describe the application of the Isolation Forest algorithm, a single ISee device is considered. Figure 19a shows the 2D plot, obtained with PCA, of the data samples. It is worth noticing that the last cycle shows the highest variability and it has a different behavior compared to the previous ones. The

results of the application of the Isolation Forest are shown in Figure 19b; with the 2D representation, it is possible to visualize both inliers and outliers in a unique plot. The colors of the first plot represent the predicted values of the isolation forest:

- **Blue color** indicates an anomalous observation.
- **Yellow color** indicates a normal observation.

The PCA highlights that the most distant data points compared to the rest of the data are classified as anomalous from Isolation Forest. Moreover, we can observe that the anomalies correspond to the last cycles, as can be seen in the second PCA representation in Figure 19c.

To better understand the distribution of anomalies, an additional binary variable, called *last_3_cycles*, is introduced: the variable is one when considering the last three cycles of a battery, zero otherwise. The confusion matrix, shown in Figure 20, clearly highlights the distribution of the anomalies: 30 over 33 anomalous points are grouped in the last three cycles.

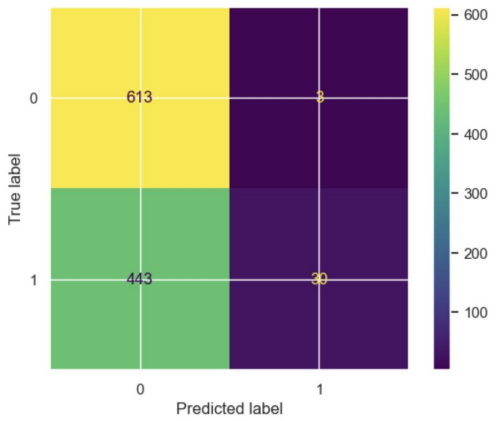
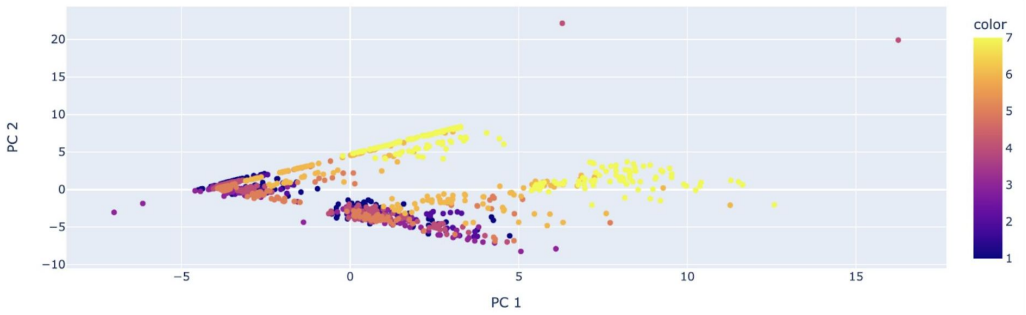
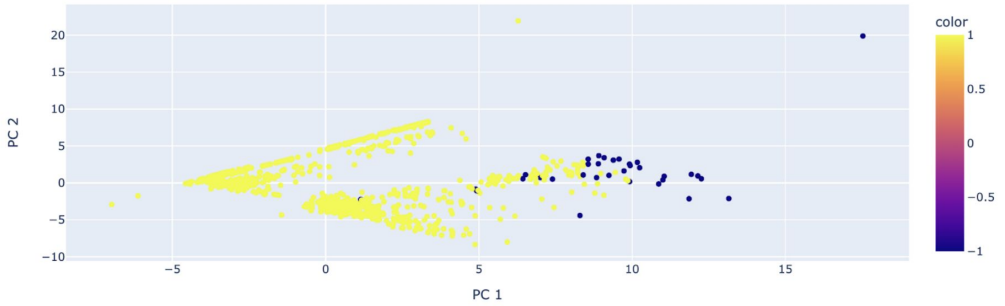


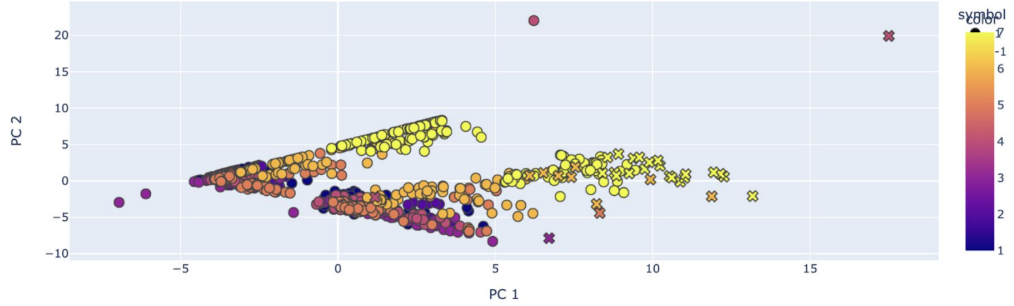
Figure 20: Confusion matrix that compares the last_3_cycles variable with the predictions of isolation forest.



(a) 2D plot obtained with PCA, each color represents a cycle



(b) 2D PCA plot colored by predicted label of isolation forest.



(c) 2D PCA plot colored by cycle id, where the cross markers represent the anomalous cycles predicted by IF.

Figure 19: Isolation Forest results with a single smart glasses.

5.2 Anomaly Interpretation

In Machine Learning, often the more complex predictors are quite difficult to interpret, this means that the model may perform very well, but its decisions cannot be understand by humans.

In the case of Anomaly Detection, for example, it could be possible to find a way to effectively identify anomalies, but humans could not understand why outliers are classified as such.

The interpretation techniques that are presented in the next sections are algorithms explicitly created to make the results of machine learning models easier to understand for a human reader; two approaches will be used to better understand the features that seem the most important in anomaly detection, both *locally*, for a single outlier, and *globally*, considering all observations.[2].

5.2.1 SHAP: SHapley Additive exPlanations

The Shapley value [1] is a solution concept introduced by the economist Lloyd Shapley in 1951; it is based on game theory, the setup is the following: "A group of differently skilled players is cooperating in a game to gain a prize. How to divide the prize fairly among the players?".

Considering a Machine Learning environment, the prize is the outcome of the predictive model, the players are the features: which are the features that contribute the most to the model prediction?

The Shapley value has solid theoretical foundations and it is model-agnostic (can be used with any type of predictive model), but the algorithm has a very high computational complexity, so, in practice, there is the need to rely on approximations.

SHAP is a more recent method based on the Shapley value, the nice things of this method are:

- Efficient implementation.

[2] An extensive treatment on Machine Learning interpretability can be found in the book "Interpretable Machine Learning" [26]

- Efficient model-dependent implementation (for example, for decision trees).
- Addition of aggregation-based methods for global interpretation.

All technical details can be found in the original papers: [8], [10], [21].

For this analysis, the standard method implemented in the SHAP library was applied to the fitted Isolation Forest; an example of local interpretation can be seen in Figure 21.

The limitations encountered using this approach are related to the high time required by the algorithm and, more importantly, to the less expressive global interpretations. To overcome these problems, another approach, called DIFFI, was used, described in the next section.

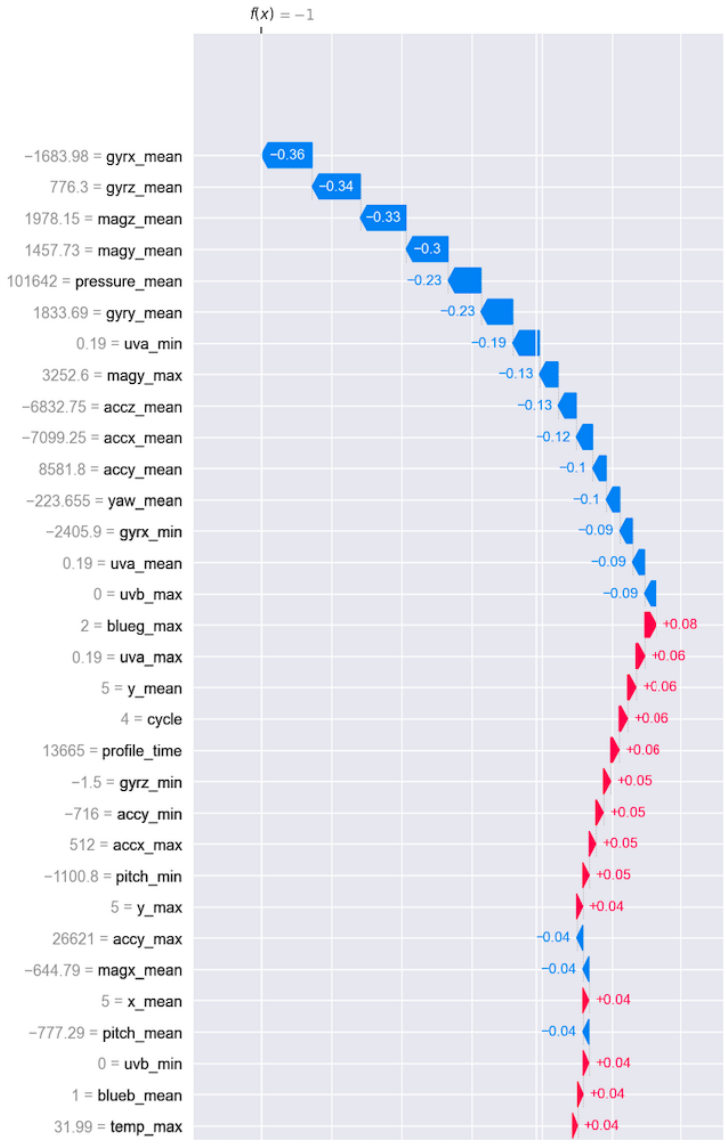


Figure 21: Example of SHAP Local interpretation: the blue arrows "move" the prediction towards -1 (anomaly)

5.2.2 DIFFI: Depth-based Isolation Forest Feature Importance

DIFFI is an interpretation technique proposed in Paper [17]; it is designed specifically for the Isolation Forest algorithm. The main advantages of using this approach are as follows:

- Global interpretation method.
- Local interpretation method (*Local-DIFFI*).
- An unsupervised feature-selection technique based on DIFFI.
- Designed for Isolation Forest.
- Fast and efficient implementation.

DIFFI exploits two hypotheses to assess the importance of the features. A split-test associated to an important feature should:

1. induce the isolation of anomalous observations at small depths of the trees, while relegating the regular data at greater depths.
2. produce higher imbalance on anomalous observations, while being quite useless on the other data points.

The feature importance is based on the combination and normalization of the *Cumulative Feature Importances (CFI)* defined for both inliers and outliers; the *CFIs* are updated following an *update rule* that takes into account the depth of the node that contains the considered observation (first hypothesis) and the *Induced Imbalance Coefficient (IIC)* (second hypothesis). Picture 22 shows the structure of the algorithm.

Using DIFFI, it was possible to have a second local interpretation of the anomalies, as well as a global analysis. Table 5 contains the results for the local interpretations, while Table 6 the global ones.

Given the efficiency of the algorithm, the procedure was applied even to working glasses, with the idea of better understanding which features produce the greatest number of anomalies. In table 7 the results obtained with the global interpretation are presented.

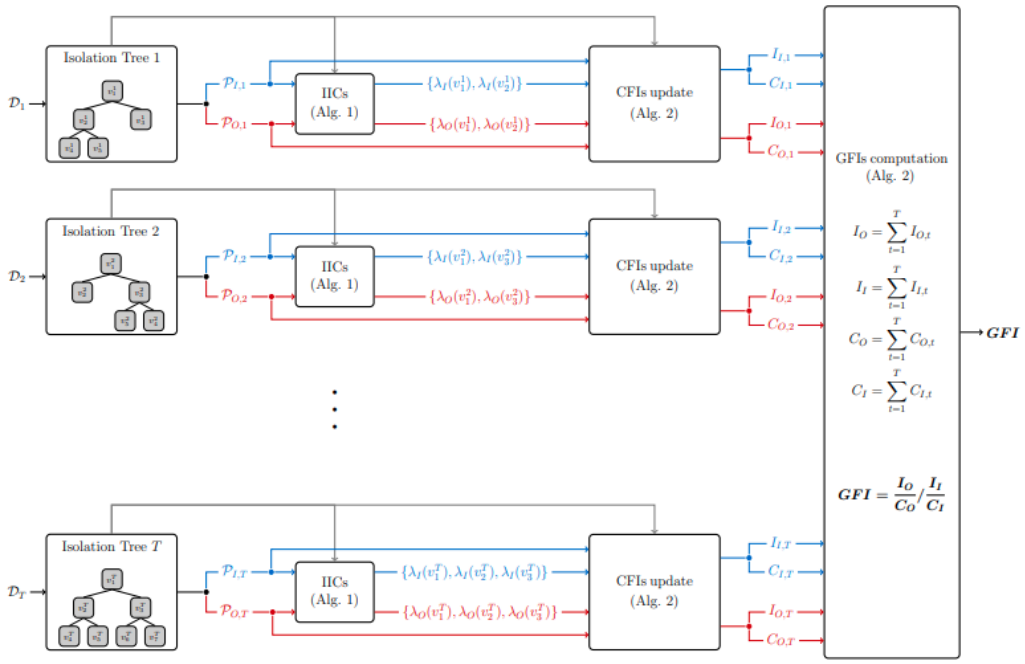


Figure 22: Structure of DIFFI algorithm

Rank 1		Rank 2		Rank 3	
Feature	**Count.**	**Feature**	**Count.**	**Feature**	**Count.**
gyrx_mean	0.15	x_mean	0.09	accx_mean	0.09
accx_mean	0.12	accy_mean	0.09	roll_mean	0.09
uva_mean	0.12	magx_mean	0.06	uva_mean	0.06
roll_mean	0.09	uvb_mean	0.06	x_mean	0.06
blueg_max	0.06	accz_mean	0.06	accz_mean	0.06

Table 5: Local DIFFI interpretation: Count. is a **normalized counter**. For example magx_mean is the second most important feature (rank 2) for 0.06*anomalies, considering all features of rank 2 it's the third that appears the most.

Position	Feature	Feature Importance
1	worn_min	5
2	worn_max	4.18
3	worn_mean	3.7
4	uva_mean	3.31
5	gyry_mean	2.51
6	gyrz_mean	2.46
7	accx_mean	2.46
8	roll_mean	2.39
9	accy_mean	2.39
10	accz_mean	2.22

Table 6: Globlal DIFFI interpretation

Position	Feature	Feature Importance
1	worn_max	3.5
2	worn_mean	3.35
3	gyrx_mean	2.75
4	gyrx_max	2.44
5	y_mean	2.23
6	uva_mean	2.18
7	blueb_mean	2.18
8	blueb_max	2.13
9	gyry_min	2.05
10	gyry_max	1.99

Table 7: Globlal DIFFI interpretation on working glasses

5.3 Conclusions on Anomaly Detection

This section presents the result of the application of Anomaly Detection algorithms to understand if the faults of the defective glasses could have been noticed before the actual break. First, the Isolation Forest algorithm was applied to highlight the recorded outliers; then a qualitative analysis was performed to better understand the correlation between the last discharging cycles before the fault and the anomalous points.

After the qualitative analysis, it was decided to go deeper into the interpretation of the results, with the idea of selecting a subset of features that could be really important for anomaly detection. The identification of such a subset would be very important for the study of the anomalies, for better exploring the problem that caused the break of the smart glasses, but also for a better evaluation of the importance of the outliers: by comparing the environment in which the observations were recorded and the features that contribute the most to the anomalies, it could be possible to better highlight the separation of those outliers that were caused by a modification of the environmental condition, from those anomalies that were actually warning signs for the imminent fault of the devices.

6 Remaining Useful Life Prediction: Machine Learning experiments

In this section, preliminary studies on the Remaining Useful Life (RUL) prediction are presented. The experiments presented in this section were performed considering only the two out of five working smart glasses of the first set; due to the limited availability of data, it was chosen to focus on classic machine learning approaches instead of advanced deep learning technologies; the second, in fact, typically requires large datasets to be trained.

6.1 Introduction to RUL prediction

As said in the introductive section, the goal of Predictive Maintenance is to estimate the best moment in which to perform the maintenance of a product. Predictive Maintenance is not based on a statistical mean estimation of the possible life of the considered entity, instead, it usually relies on machine learning models based on the sensor measurements available, focusing on the actual condition of the subject.

As stated before, one of the most important formalizations of the PdM problem is the RUL estimation: the idea is to quantify, at a given moment, the time remaining for the subject before it loses its operational ability.

The definition of RUL depends on the considered device; in this case, RUL consists of the number of discharge cycles left to the battery before its failure. It is important to distinguish between the *remaining life* and the *remaining useful life*: in some environments the two measures do not coincide; in Paper [22], for example, it is said: *"According to the provisions of the standard, the battery has reached the failure condition, has been aging, and cannot be used when its capacity value drops to 70%–80%."*

6.2 Setup: Software and Limitations

Before presenting the pipeline, it is better to highlight the two main encountered limitations:

1. **Few data**: A sample, in predictive maintenance, is the entire life cycle of a specific subject; only two batteries were available, so only two samples. For this reason, it was not possible to apply *cross validation*, so, for this section, a final assessment of the model with a test set is missing.

2. **RUL choice is arbitrary**: The actual failure of the batteries was not encountered and it was not possible to evaluate their capacity. The RUL was set forcing an hypothetical break in the last available cycle.

Given the limitations presented, the numerical results presented in this section are not to be considered statistically reliable; the interesting part refers to the adopted approach.

6.2.1 CeRULEo

It is the backbone of the pipeline. CeRULEo (*Cool utilitiEs for Remaining Useful Life Estimation methOds*) is an open-source and publicly available library designed specifically to make predictive maintenance tasks both easy and intuitive. It is developed by Luciano Lorenti, a research fellow in the University of Padua.

The library is written in Python 3 and is compatible with the most used machine learning libraries, like Scikit-learn [5], Pandas and Keras [7].

CeRULEo provides support through all the classic PdM pipeline; the main features are the following:

- **Dataset**: it's an abstract class that handles the dataset, it's the accepted input of the predictive models. The constitutive units are run-to-failure cycles, stored as Pandas.DataFrame.

- **Sensor Validation and Analysis**: a set of measurements useful to assess the quality of the collected data, for example: standard deviation,

correlation, autocorrelation, entropy and much more. Very interesting is the "pairwise correlation" function, which was used to get an idea of the feature most correlated to the RUL.

- **Iterators**: support for "rolling window" iterator, very useful when dealing with time-series data.
- **Transformers**: an object that generates the pipelines for the model, very useful for preprocessing, transforming and selecting the data to feed to the model.
- **Models**: wrapper for the well-known libraries for Machine Learning, like Scikit-learn [5] and Keras [7]; it also contains some pre-compiled models that are important in literature.
- **Results**: a set of tools useful for evaluating and plotting trained models.

Here are the links to the source code [29] and the documentation [28].

6.3 Algorithms and Results

The first thing done after implementing the CeRULEo Dataset was to compute the pairwise correlation of the features; the focus was on the correlation between the RUL and the other variables, that is a useful initial guess on the features that could contribute the most to the prediction. In Table 8 a portion of the correlation analysis is reported; the variables are sorted by *absolute mean correlation*.

As first approach, it was decided to keep all numerical features except for *cycle* and *worn_min*, for the first is like keeping the target variable in the train set and the second contained some *NaN* values.

At this point, a Ceruleo Transformer was created to automatically select the train and target features and apply the chosen preprocessing strategy that, in these experiments, is a MinMaxScaler. The devices were divided into a train set that was used to fit the transformer and the models, and a validation set; the division is reported in Table 9.

Feature	Abs Mean Corr	Max Corr	Min Corr
magx_mean	0.66	0.69	0.64
magx_max	0.66	0.68	0.63
magx_min	0.65	0.67	0.62
rh_max	0.49	-0.43	-0.55
rh_mean	0.49	-0.43	-0.55
rh_min	0.47	-0.40	-0.52
magz_max	0.33	0.18	-0.48
magz_mean	0.33	0.17	-0.49
magz_min	0.31	0.14	-0.49
magy_mean	0.26	0.42	-0.11
magy_min	0.25	0.40	-0.11
magy_max	0.25	0.43	-0.07
gyrz_mean	0.22	0.25	0.19
pressure_min	0.19	-0.19	-0.20

Table 8: Pairwise Correlation

Set	MAC address	Number of cycles
Train set	D5:B8:15:AD:01:0D	53
Validation set	F0:53:52:26:FD:2F	54

Table 9: Division of the ISee devices

The main models tested for the experiments are:

- LinearRegression / RidgeRegression / SGDRegression.
- SVM for regression, with all standard kernels.
- RandomForestRegression.
- AdaBoost with linear models.
- GradientBoostingRegressor.
- MLP Neural Networks.

The results were quite bad with the linear models, a little bit better with an SVM with *RBF* kernel and quite bad, due to overfitting, with Neural Networks; the most promising models were **RandomForestRegressor** and **GradientBoostingRegressor**

6.3.1 Evaluation Metrics

To evaluate the models, four different metrics were considered:

- Mean Squared Error (**MSE**): $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(\vec{x_i}))^2$
- Root Mean Squared Error (**RMSE**): $RMSE = \sqrt{MSE}$
- Mean Absolute Error (**MAE**): $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - h(\vec{x_i})|$
- Coefficient of Determination ($\mathbf{R}^2$)

Particularly interesting is the R^2 coefficient, which, instead of giving a direct measure of the error, produces a score that compares the performance of the chosen model with respect to a *Dummy Predictor* that always returns the average of the target variable. This is quite useful because the coefficient value gives a very intuitive measure of the performance of the model (i.e., that does not need additional knowledge on the specific task to be understood), while also taking into account, at least in part, the complexity of the data.

The score is computed as follows:

- Consider an observation $(\vec{x_i}; y_i)$, where y_i is the target variable.
- Considering the chosen model **h**, it is possible to compute the *Residual Sum of Squares* (**RSS**) as follows: $RSS = \sum_i (y_i - h(\vec{x_i}))^2$
- Considering the Dummy Predictor, it always returns the mean of the target variables, which is called $\bar{y}$. The *Total Sum of Squares* **TSS** is defined as follows: $TSS = \sum_i (y_i - \bar{y})^2$

Finally, the R^2 score is defined as:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_i (y_i - h(\vec{x_i}))^2}{\sum_i (y_i - \bar{y})^2}$$

From the definition of the coefficient, it is easy to see that:

- $R^2 < 0$ means that the error of the chosen model is larger than that of the Dummy predictor, so the model performs badly.
- $R^2 = 0$ means that the chosen model and the Dummy one perform equally.
- $0 < R^2 \leq 1$ means that the chosen model performs better than the Dummy predictor; the closer R^2 is to 1 the better the model.

6.3.2 Random Forest

Random Forest [3] is an *ensemble* model based on *Decision Trees*. A Decision Tree is a model that try to predict a value by learning simple decision rules inferred from the data; it can be seen as a piecewise constant approximation of the target function. The nodes of the tree are the decision functions; the observation starts from the root, the result of the decision function identifies the next node to choose among the children of the current one; when the observation reaches a leaf, the prediction is obtained. The greater the depth of the tree, the more complex the model.

A Random Forest is just a set of decision trees; each tree has random components and it is usually trained considering, to identify each decision

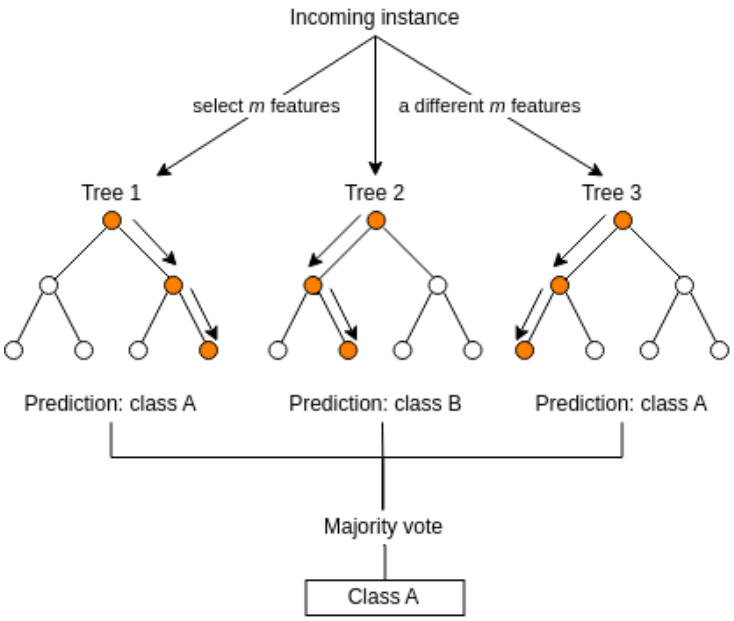


Figure 23: Structure of Random Forest for classification

function (or split), a random subset of the available features: in this way it is possible to obtain a set of models that are not highly correlated. The final prediction of the model is obtained by aggregating the results of all the trees, the idea being that the error that a tree could make will not be relevant due to the output of all the other decision trees. Figure 23 shows the structure of the Regression Forest for classification.

This model was embedded inside a CeRULEo regressor; the dimension of the rolling window was defined and, after some tuning, the model with hyperparameters specified in Table 10 was obtained. Table 11 contains the evaluation of the model.

Hyperparamter	**Value**
window_size	32
n_estimators	300
max_depth	6
max_features	log_2

Table 10: RandomForestRegressor hyperparameters

Set	MSE	RMSE	MAE	R^2
Train	4.23	2.06	1.47	0.98
Validation	130.05	11.40	8.69	0.44

Table 11: Random Forest Evaluation on Validation Battery

The algorithm is implemented in the Scikit-learn library; all the non-mentioned parameters were used with their default values. Here is the link to the algorithm: [33]

6.3.3 Gradient Tree Boosting

Gradient Tree Boosting is a *boosting* method that uses Decision Trees as weak learners. To explain the Boosting idea, the AdaBoost algorithm [2] steps are summarized: to get a prediction, the AdaBoost algorithm try to fit a sequence of weak learners, that are learners that perform slightly better than random guessing, on repeatedly modified versions of the data. The outputs of all weak learners are then aggregated to obtain the final prediction. After the data are passed through a weak learner, they are modified according to some weights (one for each training sample); these weights are then tuned by the model by increasing the weights associated to miss-interpret observations while decreasing the others, in this way the next weak learner should focus more on the wrongly predicted data-point instead of the correctly predicted ones. The Gradient Tree Boosting algorithm uses decision trees as weak learners, but it also generalizes the boosting strategy by allowing the optimization of any differentiable loss function. Figure 24 shows the structure of a Gradient Tree Boosting algorithm.

This model was embedded in the CeRULEo pipeline, achieving, after some tuning, the best predictor found in this section; the hyperparameters are reported in Table 12, while in Table 13 the evaluation metrics are written.

The algorithm is implemented in the Scikit-learn library; all the non-mentioned parameters were used with their default values. Here is the link to the algorithm: [30]

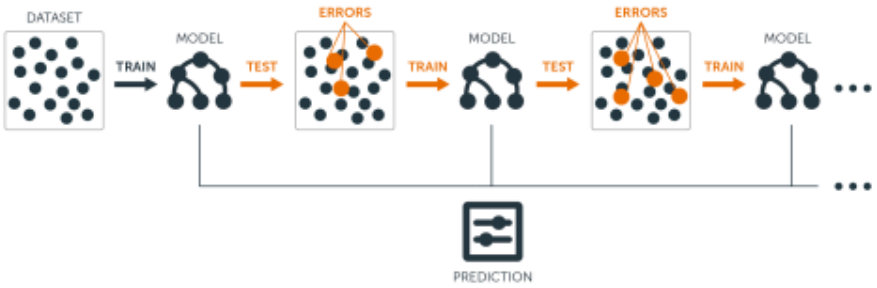


Figure 24: Structure of Gradient Tree Boosting

Hyperparamter	**Value**
window_size	48
loss	huber
n_estimators	1500
subsample	0.8
max_depth	3
max_features	log_2
alpha	0.8

Table 12: GradientTreeBoosting Hyperparameters

Set	**MSE**	**RMSE**	**MAE**	$\mathbf{R^2}$
Train	0.21	0.45	0.28	0.99
Validation	113.8	10.67	8.38	0.51

Table 13: GradientTreeBoosting Evaluation

6.4 Model Interpretation and Feature Selection

After the identification of a couple of promising models, it was important to identify the subset of features that contributed the most to the RUL prediction; this has been proven useful as a metric for a feature selection step, but it is also very important to better understand how smart glasses react to the aging of the battery and which sensors better show this decline.

As briefly introduced in Section 5.2, the interpretation of a model is particularly important when dealing with complex predictors: the expressivity that powerful models can achieve makes it very difficult for a human to understand how the algorithm acts in order to obtain its output. A learner could perform very well on a specific task, but without interpretation all its analysis cannot be used to increase and improve the human knowledge on the subject, resulting in a definitely useful tool, but not completely exploited. As described in the book [26], there are also ethical and legal considerations related to the interpretability of the model.

To better interpret the predictions, it was necessary to define a variation of the GradientTreeBoosting model described in Section 6.3.3, in particular:

- The **window size** is fixed to 1, to have a clearer relation between the feature and its score.
- The **init** model needs to be the default one to have compatibility with SHAP.

All computations were performed on the Validation Set.

For this task, to have a comparison between differently computed importance scores, two different procedures were applied:

- SHAP, described in section 5.2.1.
- Permutation Feature Importance (**PFI**), that will be described later in section 6.4.1.

This time the focus was only on global interpretations. Firstly, SHAP was applied in its *Tree Version*, which is quite fast, to obtain two plots: one

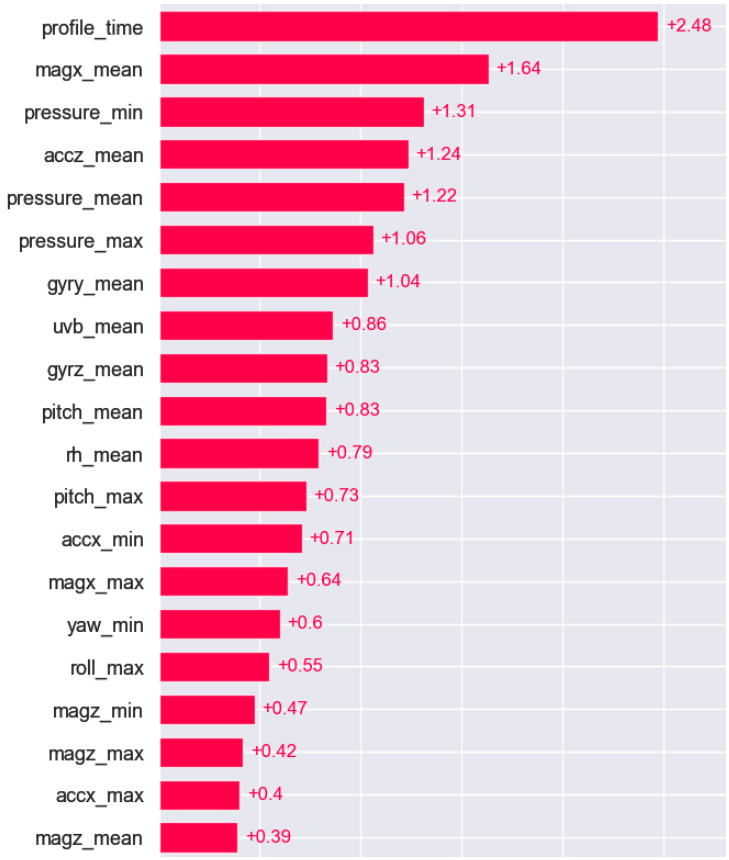


Figure 25: Partial SHAP global interpretation considering all the features

that only represents the most important features 25 and the other that also describes how the value of a feature changes the importance scores 26.

Then Permutation Feature Importance was applied and the plot 27 was obtained.

From the plot of the PFI 27 it is clear that there are some features that are actually important for the model, some that are ignored and even some that mislead the predictions; so it was decided to select only the positive features, discarding the ignored and misleading ones, and retrain the algorithm of section 6.3.3; the results are reported in Table 14.

As can be seen in Table 14, the performance of the algorithm has improved slightly, validating the feature selection approach.

One of the problems of PFI, by the way, is the presence of highly corre-

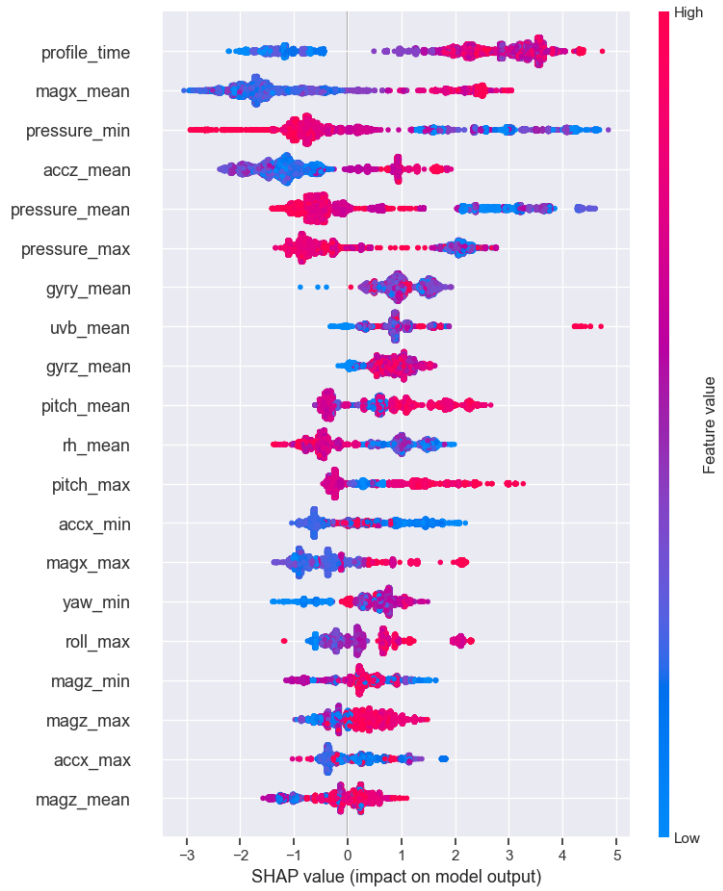


Figure 26: Partial SHAP global interpretation considering all the features

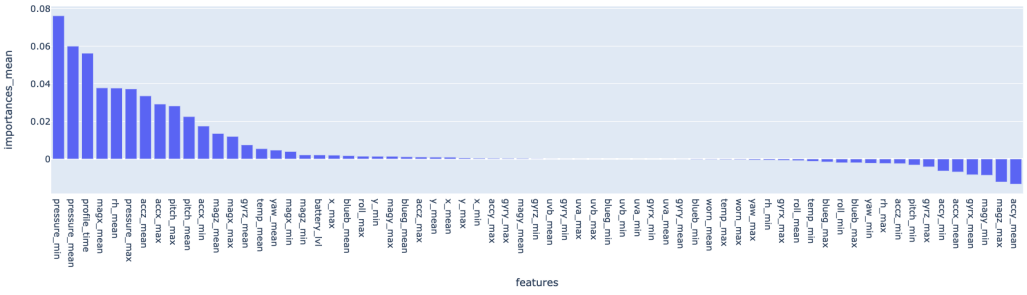


Figure 27: Permutation Feature Importance with all the features

Set	MSE	RMSE	MAE	R^2
Train	0.13	0.36	0.20	0.99
Validation	106.60	10.32	7.56	0.54

Table 14: GradientTreeBoosting Evaluation after feature selection

lated variables in the dataset, as explained in Section 6.4.1. In the dataset there are features that are produced by the same sensor, but aggregated differently according to mean, maximum and minimum, so, excluding strange cases, there is a high correlation between all the aggregated variables. This is the reason why it was decided to propose again the training and interpretation of models 6.3.2 and 6.3.3, but considering only the mean as form of aggregation for the sensors: this should result in a lighter model that should almost maintain its previous expressivity.

The results of the training considering all the "_mean" features are reported in Tables 15 and 16.

Set	MSE	RMSE	MAE	R^2
Train	4.15	2.04	1.44	0.98
Validation	147.59	12.15	9.57	0.37

Table 15: RandomForestRegressor Evaluation with Only Mean Features

Set	MSE	RMSE	MAE	R^2
Train	0.13	0.36	0.20	0.99
Validation	124.36	11.15	8.69	0.47

Table 16: GradientTreeBoosting Evaluation with Only Mean Features

At this point, the same interpretation techniques described above (Figures 28, 29, 30) were applied and, for the second time, the subset of the most important features was selected: not only did the model improve its accuracy (as reported in Table 17), but it also achieved the best results exploiting fewer features.

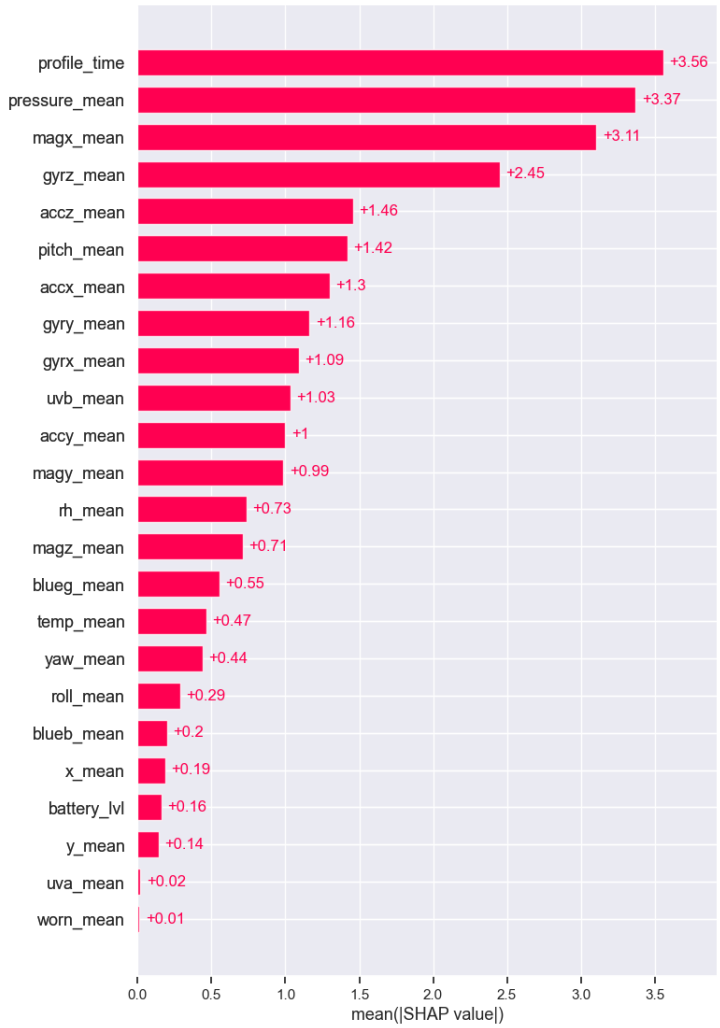


Figure 28: Global SHAP interpretation with only mean features

Set	**MSE**	**RMSE**	**MAE**	**R^2**
Train	0.22	0.47	0.18	0.99
Validation	102.00	10.10	7.57	0.56

Table 17: GradientTreeBoosting Evaluation with only mean features after feature selection

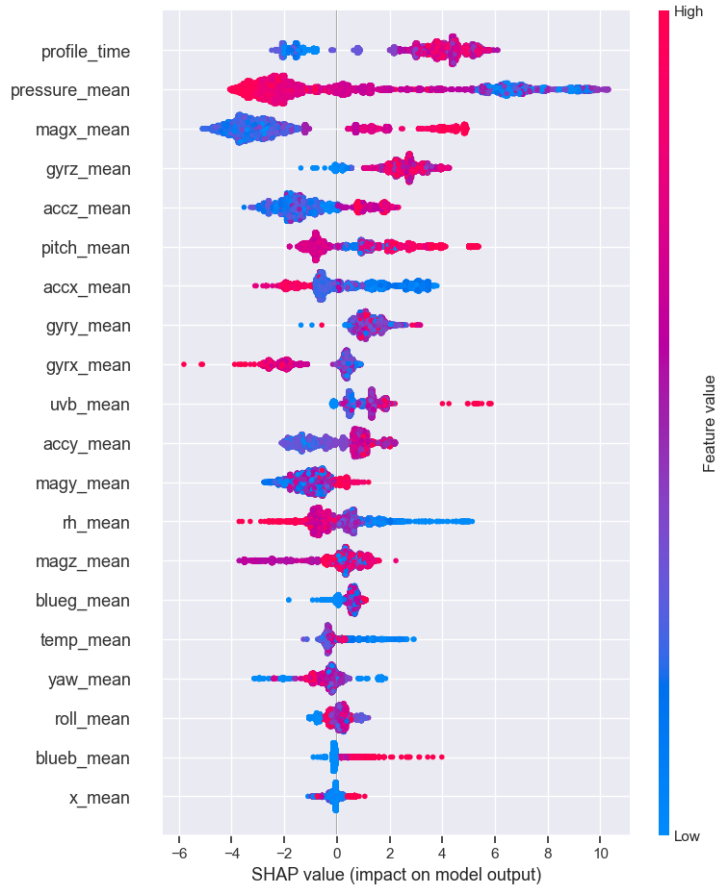


Figure 29: Global SHAP interpretation with only mean features

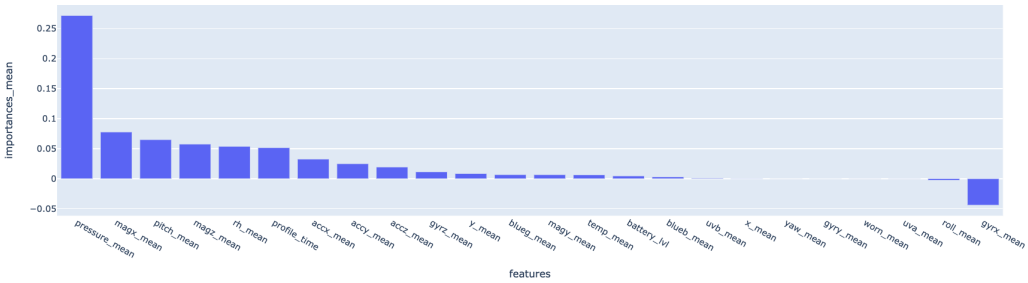


Figure 30: PFI with only mean features

The subset of the most important features (sorted according to PFI) is reported in Table 18.

Rank	**Sensor**	**Rank**	**Sensor**
1	Pressure	10	Gyrz
2	Magx	11	Y
3	Pitch	12	Blueg
4	Magz	13	Magy
5	RH	14	Temperature
6	Profile Time	15	Battery Level
7	Accx	16	Blueb
8	Accy	17	Uvb
9	Accz	18	X
19		Yaw	

Table 18: Most Important Features

6.4.1 Permutation Feature Importance

Permutation Feature Importance is a technique introduced by Breiman [3] and then generalized into a model-agnostic version by Fischer, Rudin and Dominici [9].

The idea of the algorithm consists in evaluating the loss of the model after the permutation of a feature values, breaking the relations with the true outcome: if the error increases, then the feature is important for the model, if the error remains constant, the feature is ignored, and if the error decreases, the feature is actually misleading for the model. The feature importance scores are computed using the difference between the original loss and the loss after the permutation of the feature.

This approach is valid due to its ability to produce a nice and simple interpretation without the need of retraining the model (so it is a fast technique to apply), but it has two main disadvantages:

- PFI is linked to the error of the model: a feature considered not important by a bad model could be very important for a good predictor.
- Random approach: it needs more than one permutation to obtain stable importance scores.
- It is not reliable in the presence of highly correlated features: the model, even after the permutation, could retrieve the information using the correlated variable, resulting in a decrease of the importance of both features; on the other hand, it can be possible to evaluate the model over data that cannot be find in reality (for example, when considering the height and weight of a person, after a permutation it would be possible to find a person 2 meters tall that weighs 30kg, a very unrealistic tuple).

The algorithm used in the experiments is the one implemented in Scikit-learn [32], additional information can be found in this book [26].

6.5 Final Considerations

In this section the RUL prediction task, applied to a couple of smart glasses, was explored; as stated at the beginning of the section, there are some limitations that prevent one from considering the numerical results stable, but on the other hand, the approaches and the software used remain quite interesting given the considered information. In particular, the identification of a couple of models that perform way better than others is quite promising.

Particularly relevant is also Section 6.4, dedicated to the interpretation of the model and feature selection, in which was identified the subset of sensors that contribute the most to the predictions, hence the subset of sensors that seems to be quite informative about the decline of the battery. The comparison between these results and the pairwise correlation matrix 8 highlights that some sensors that seemed to be quite relevant a priori are indeed important even for the model, but there are also some features that did not seem to be too correlated with the RUL that are still considered important by the predictor.

7 Advanced Experiments

During the second period of the work, it was possible to collect data from more devices, hence, given the expanded dataset, it was also possible to apply Deep Learning models for solving the RUL prediction task.

7.1 ISee and Deep Learning

During these experiments, eight ISee smart glasses were available and used for data collection. All available discharge cycles were considered, as reported in Table 2. Even in this part of the work, Ceruleo was the backbone of all experiments: it was used to split the batteries in train (5 batteries), validation (2 batteries) and test (1 battery) sets, as reported in Table 19, to select the features and the target variable and to create the windowed samples. A windowed sample corresponds to a sequence of data points, each point is a reading. The selected window size was 64, the selected features, at least at the beginning, were 24, resulting in a dataset of shape: $(n\_samples, 64, 24)$. All values were normalized in a range $(-1; 1)$ using a Min-Max scaler robust to outliers.

Set	**MAC Address**	**Number of cycles**
Train	E3:38:BA:9F:0F:B0	46
	ED:F2:0E:AF:7C:4C	50
	E7:D7:43:C8:22:1E	49
	E8:22:97:31:0D:4A	50
	D5:B8:15:AD:01:0D	114
Validation	E5:2A:C4:7E:F2:F6	39
	F0:53:52:26:FD:2F	117
Test	EC:3B:5C:4D:BA:B6	40

Table 19: Division of the ISee devices

7.1.1 Custom Architectures

The first experiments were carried out using the classical Keras layers, such as GRU, LSTM and Conv1D. The structure of the networks remained quite similar:

- A feature extractor composed of multiple layers, convolutive, gated or both, interspersed with dropout or normalization layers. Sometimes, pooling layers were added.
- A layer to flatten the data, usually a global pooling one.
- A regression head composed of a sequence of fully connected layers interspersed with dropout or normalization layers. The last layer is composed of a unique neuron with a linear activation function.

The Rectified Linear Unit (ReLU) activation function was used for all layers but the gated one, in which the Hyperbolic Tangent (tanh) was chosen in order to better exploit GPU computation. Table 20 contains some examples of results obtained with the custom model; the performance of all networks is not as good as expected, the models tend to overfit and have difficulty generalizing on the validation set. To try to improve the results, it was decided to test some advanced architectures found in the literature.

Base Layers	**RMSE**	**MAE**	**% Error**	$\mathbf{R^2}$
GRU	17.20	13.72	18%	0.75
LSTM	21.75	17.24	23%	0.60
Conv1D	24.38	20.16	26%	0.49

Table 20: Custom architectures results (Validation set only)

7.1.2 Well-Known Architectures

Ceruelo implements a few famous architectures described in literature to solve the RUL prediction problem; here a brief summary of the tested networks:

- **CNLSTM**: Proposed in paper [13], it is a network that aims to learn both long and short temporal dependencies for RUL estimation by combining Temporal Convolution and LSTM layers. Figure 31 represents the structure of the proposed network.
- **MSWRLRCN**: Proposed in paper [27], it is a network that combines convolutional blocks, LSTM units and Attention mechanism for the task of RUL prediction applied to Rolling Bearings. The architecture is summarized in the picture 32.
- **XiangQiangJianQiaoModel**: Proposed in paper [11], it is a deep convolutional network that operates with convolution in both one- and two dimensions. It should require very basic data preprocessing, which makes the architecture very convenient for real applications. The structure of the network is summarized in Figure 33.
- **XCM**: A modified version of the network proposed in [18], in which the classification head was replaced with a regression head. It is an architecture for general classification of multivariate time-series built to be interpretable. It is a convolutional structure, as can be seen in Figure 34.

Even if some of these architectures worked better than others, the results obtained, reported in Table 21, are still not so good: the models had difficulties in generalization and the overall error on the validation set remains quite high.

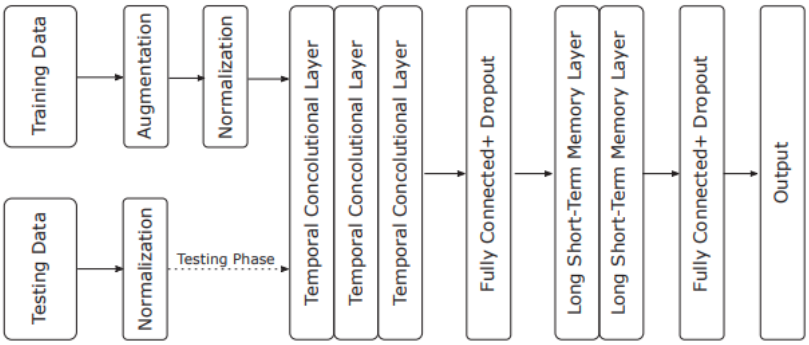


Figure 31: CNLSTM

Network	RMSE	MAE	% Error	R^2
CNLSTM	27.16	21.68	28%	0.37
MSWRLRCN	21.12	15.22	20%	0.62
XiangQiangJianQiaoModel	16.85	13.50	18%	0.76
XCM	19.52	16.53	22%	0.67

Table 21: Well-Known Architectures

7.1.3 Final Considerations Isee

After all the experiments performed, both with custom and well-known architectures, it is clear that the problem is not related to the choice of the model, but more likely it depends on the dataset. No model can generalize well and the performance in the test set is much worse than the one on validation. This can be caused by the feature of the dataset, which sure does not represent directly any measure about the battery, but also by the following considerations:

1. The initial part of the degradation curve is typically flat. It was possible to gather only a few cycles for each battery, so it is likely that the data represent that flat part and the models struggle to distinguish the samples.

2. The labels are assigned by considering the last available cycle and not the real breaking point, that was not encountered. Furthermore, 6 of

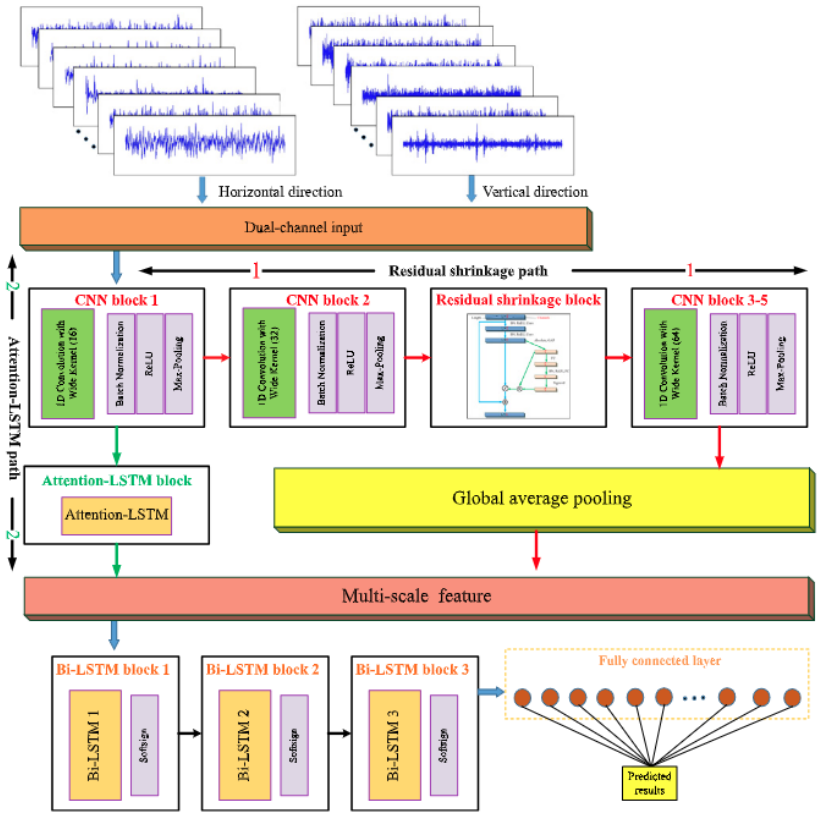


Figure 32: MSWRLRCN

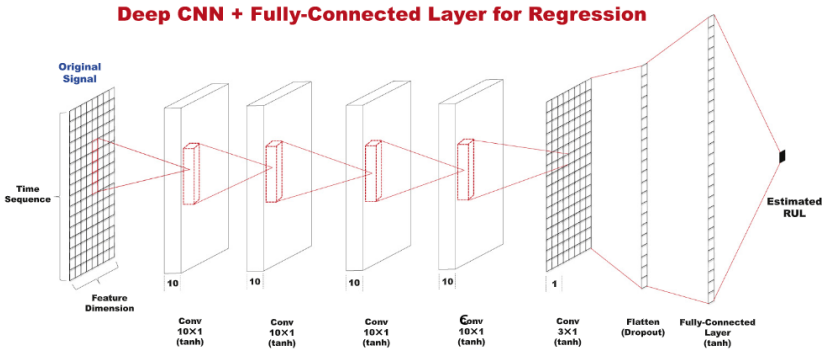


Figure 33: XiangQiangJianQiaoModel

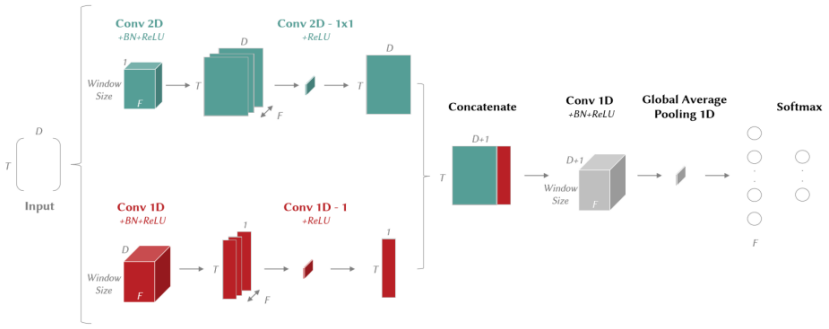


Figure 34: XCM

the 8 batteries had an unknown history before data recording, so the labels are based only on their known history, without considering what happened before. Labels can be misleading.

To improve the results, it was first decided to try to augment the sets by using a GAN to generate additional synthetic data. Two architectures were tested, DoppelGANger [20] and TimeGAN [16]; the latter seemed to work quite well, but the contribution for the RUL prediction task was not enough to really improve the results. Moreover, GAN training is very difficult, slow and requires a lot of hardware resources. On the other hand, to try to reduce the labeling problem, it was decided to test an additional self-supervised pre-training step inspired by article [25]. An autoencoder network was designed and trained in an unsupervised way to learn a good feature extractor, the encoder; the encoder was then used in a transfer-learning and fine-tuning pipeline in order to be adapted for the regression task. Unfortunately, even with this approach, the results did not seem to improve. It is likely that, to really find a good model for the task, a better data collection is needed.

7.2 Second Platform Advanced Experiments

The division of the second platform data set is similar to the one done for the ISee devices, as shown in Table 22. The initial experiments are based on more classical machine learning approaches to better understand the complexity of the task and the data set. Then, the experiments focused on deep learning models, which could be used given the automatic data recording table that allowed for a more proficient data collection.

Set	Battery ID	Number of cycles
Train	511A	245
	4A71	310
	D2E9	255
	D4CC	318
	593A	224
Validation	E597	311
	667D	311
Test	BD7A	286

Table 22: Division of the second platform devices

7.2.1 Polynomial-based Predictor

For these experiments, instead of considering a windowed dataset as done for the ISee, each cycle was considered as unique sample. The idea was to use directly the measures relative to the cycle to predict the RUL. It was noticed that only some signals had a significant shape considering a single cycle, so, it was decided to perform a feature selection step and include only those meaningful features. The selected features are reported in Table 23.

To create an input feature vector that was descriptive of the whole cycle, it was decided to preprocess the raw data: instead of considering the values of the selected features directly, each signal was represented considering the coefficient of the polynomial fitted to the signal itself, as shown in Figure 35. The coefficients obtained from different signals were then concatenated

Type	Feature
Signals	V V_cc T_imu I_discharge
Cycle-descriptive	Capacity Profile_time

Table 23: Selected Features

in a unique feature vector. Some descriptive measures of the cycles, such as profile_time and capacity, were also concatenated in the input vector. This approach has some advantages:

1. Each cycle is considered as a whole, without overlapping windowed samples.
2. The polynomial fit results in:
   - A fixed-length feature vector.
   - A less noisy signal.
3. The possibility of including cycle-descriptive features just once.
4. A much smaller dataset with respect to the windowed one.

The approach is briefly described in algorithm 8

Since the data set obtained after the preprocessing is rather small, only classic machine learning approaches were tested. The results obtained with a simple and light-weight model, such as Linear Regression, are actually interesting: the model seems to generalize, at least on validation set, and the prediction accuracy is promising. Sadly, when using more complex regressors, the models can reach very high accuracy on the training set, but cannot generalize.

Input: set, $signals$, $attributes$, $target$, $poly\_degree \leftarrow 10$
$X \leftarrow []$
$Y \leftarrow []$
$poly\_regressor \leftarrow PolynomialRegressor(degree \leftarrow poly\_degree)$
// Consider each battery in the set
for $battery \in set$ **do**
    // Consider each cycle
    for $cycle \in battery$ **do**
        // Fit each signal
        $temp\_x = []$
        for $feature \in signals$ **do**
            $poly\_regressor.fit(cycle[feature])$
            $temp\_x.append(poly\_regressor.coefficients)$
        end
        $temp\_x \leftarrow horizontal\_concatenation(temp\_x)$
        // Add cycle-descriptive features
        for $feature \in attributes$ **do**
            $temp\_x.append(cycle[feature])$
        end
        // Update dataset
        $X.append(temp\_x)$
        $Y.append(cycle[target])$
    end
end
// Shape dataset as matrix
$X \leftarrow vertical\_concatenation(X)$
$Y \leftarrow vertical\_concatenation(Y)$
return X, Y

Algorithm 8: Polynomial Fit Preprocessing

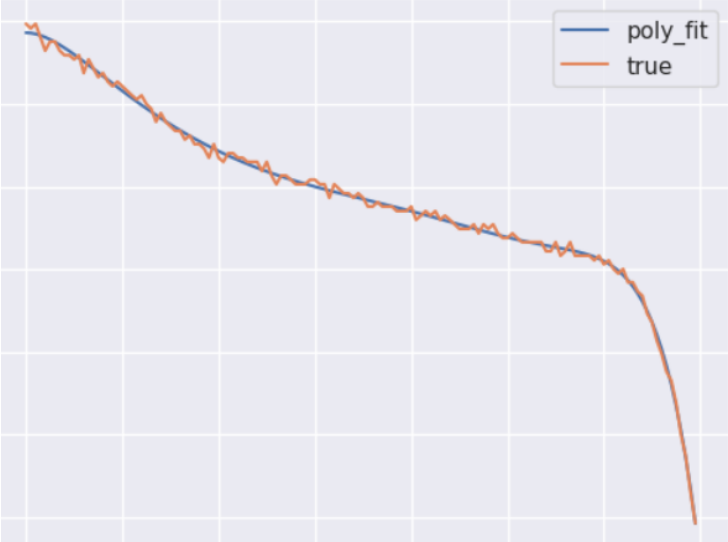


Figure 35: Example of polynomial fit

Set	RMSE	MAE	% Error	R^2
Train	48.90	39.40	15%	0.64
Validation	48.74	40.57	13%	0.70
Test	66.64	52.58	19%	0.34

Table 24: Polynomial Fit Results

Table 24 shows the results obtained with this approach; unfortunately, the performance obtained on the test set is not as good as that obtained in the validation.

7.2.2 Deep Learning Experiments

For deep learning-based experiments, all cycles were divided in windowed time series of length 64 and the data points were normalized with a robust Min-Max scaler. The features considered are those reported in Table 23, the same as those considered for the previous approach. As for the ISee, both custom and well-known architectures were tested. At the beginning of the experiments, it was noticed that there were two batteries that did not well conform with the others, probably because those devices had a problematic

Set	Battery ID	Number of Cycles
Train	D2E9	255
	4A71	310
	D4CC	318
Validation	511A	245
	E597	311
Test	667D	311

Table 25: Devices Considered for Deep Learning

data collection or because they behave differently with respect to the other devices. Therefore, the data set used for these experiments was modified as shown in Table 25.

Table 26 contains some example of the results obtained with the best models; even if there are some predictors that show promising results, it is clear that, even with this second platform, the models have difficulties in generalization: the networks tend to overfit too much. An interesting characteristic of this platform is the fact that the models actually work very well with only a few cycles, but the performance drops if the number of battery cycles increases. This behavior is probably related to the fact that, typically, degradation is not visible at early stages: it is likely that, given the limited dataset, the available points are actually quite similar, so the models struggle to distinguish between data samples, and the addition of other similar data is only an obstacle. With more battery cycles, the points will probably start to show the degradation pattern better. Table 27 shows the results of the best model, the LSTM-based one, on the test set; the network consists of:

- Three LSTM layers with 64 units each and Tanh as activation function.
- Two dropout layers between the LSTM with probability 0.25.
- A global average pooling layer to flatten the data.
- Two fully-connected layers with 64 and 32 units, respectively, and

ReLU as activation function.

- A dropout layer between the fully-connected ones, with probability 0.25.
- An output layer of dimension 1 and linear activation function.

Network	**RMSE**	**MAE**	**% Error**	$\mathbf{R^2}$
GRU-based	49.44	35.49	13%	0.69
LSTM-based	39.88	30.25	11%	0.80
Conv1D-based	48.33	37.93	14%	0.70
CNLSTM	42.88	31.65	12%	0.77
MSWRLRCN	47.78	35.60	13%	0.71
XCM	47.36	35.11	13%	0.72

Table 26: Architectures tested with the second platform (Validation set only)

Model	**RMSE**	**MAE**	**% Error**	$\mathbf{R^2}$
LSTM-based	54.73	41.08	14%	0.66

Table 27: Results of the best models with the test set

8 Related Works

There are a lot of works in literature regarding the monitoring of batteries; from more classical model-based approaches to advanced data-based methods.

In reports such as [15] or [31], the PdM approaches are classified into different classes:

- **Knowledge-Based**: an approach that leverages prior knowledge of the system. It requires the use of expert knowledge as a basis for the task. It can be divided into 3 subclasses:
  - *Ontology-based*: it formally describes the context-knowledge of a specific domain and produces a model that is easy to use and integrate, but it usually needs to be integrated with other types of models.
  - *Rule-based*: based on rules defined by experts; it can be used when there is enough expertise, but the model is difficult to be well defined. It can well leverage human expertise, but usually it cannot deal with new types of fault and its difficult to implement.
  - *Model-based*: it is based on the physical model that describes the application field. It can be very accurate and effective, but it is difficult to design, implement and adapt to real systems that are often too complex and too variable to be well captured by a physical model. Furthermore, accuracy is often compromised by small changes in the experimental setup.

  Usually, when dealing with batteries, knowledge-based approaches focus on the physical or chemical model of the battery.

- **AI-based**: given the diffusion of IoT, there is an availability of new datasets that can be widely used for this task. Data-based approaches are convincing because they require less knowledge of the field and can automatically extract good predictors. They can be very effective,

accurate and simple to obtain, but require a lot of aging data to be trained. It is possible to distinguish two macro-area of ML applications to PdM:

  - *Classic machine learning models*: they can achieve good accuracy even with limited datasets, but data preprocessing still requires some kind of human expertise in order to obtain a good feature representation.
  - *Deep learning based*: approaches based on DL to implement automatic feature engineering and extract very accurate models. From the classical DL architectures, like GRU, LSTM and Convolution, to more advanced techniques, like GAN, Autoencoders, Transfer-learning, and more. The main limitation of DL is the need for large data sets: the models are very powerful and tend to overfit if not enough data is provided. Usually, it is difficult to gather data for PdM.

- **Statistical-based**: used to analyze and perform observational studies on the collected data. Generally, the prediction is based on the previous data recorded on the same system. It is suitable for non-linear systems and supports uncertainty representation, but it is difficult to compute.

- **Hybrid models**: combines different kinds of approach with the goal of overcoming the drawbacks of the single ones. They can be very accurate, effective and stable, but are difficult to obtain and require some high-level expertise to be well defined.

Usually, in the literature, the proposed approaches are validated on public datasets; the most famous ones are reported in Table 28:

There are many papers related to the application of PdM to batteries, spanning all of the techniques reported before. Some examples of articles that were relevant for this work are briefly described:

- "Long Short-Term Memory Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-Ion Batteries" [12]. In this article,

Provider	**Description**	**Source**
TOYOTA research institute	124 commercial batteries cycled down to breaking point	Experimental
Mendeley	Panasonic 18650PF Li-ion battery data	Experimental
IEEE Data port	Automotive Li-ion battery data	Simulated
US goverment's open data	Data of commercially usable Li-ion batteries	Experimental
Science Direct	2 datasets of Li-ion batteries degradation	Experimental
CALCE battery research group	Data from different batteries	Experimental and Simulating
NASA Data repository	Two datasets of different batteries	Experimental

Table 28: Most common datasets for PdM on batteries. Source paper: [31]

the authors study the direct application of a LSTM-based recurrent neural network for RUL prediction. They show how to train the network, how to use Dropout layers to control overfitting and finally they compare their results with more classical prediction methods, like SVM, proving the superiority of LSTM-RNN for the task, in particular for what concern the long-term dependencies tracking. Figure 36 shows the high-level structure of the proposed approach.

- "A Neural-Network-Based Method for RUL Prediction and SOH Monitoring of Lithium-Ion Battery" [14]. In this article an approach based on LSTM-RNN combined with an Attention layer is proposed for the RUL prediction based on the SOH monitoring. The architecture is optimized with a Particle Swarm optimization and the data are preprocessed using a CEEMDAN. Finally, PA-LSTM, the proposed architecture, has been shown to perform better both in SOH monitoring and RUL prediction compared to other RNN models. Figure 37 shows the PA-LSTM framework.

- "Combining empirical mode decomposition and deep recurrent neural networks for predictive maintenance of lithium-ion battery" [23]. In this article, a hybrid approach is developed and tested. The hybrid model uses Empirical Mode Decomposition and Grey Relational Analysis for data preprocessing and feature extraction, then some RNNs are compared for the SOH estimation, metric that is then used for prediction of the RUL. They show that the proposed method can achieve a high accuracy.

- "Improving Semi-Supervised Learning for Remaining Useful Lifetime Estimation Through Self-Supervision" [24]. In this article, the authors propose a novel self-supervised pre-training step for RUL prediction when the labeled dataset is limited. It is an important aspect to consider in the PdM field because of the difficulties of collecting complete degrading trajectories. The authors compare existing semi-supervised and self-supervised methods, obtaining an approach that can outper-

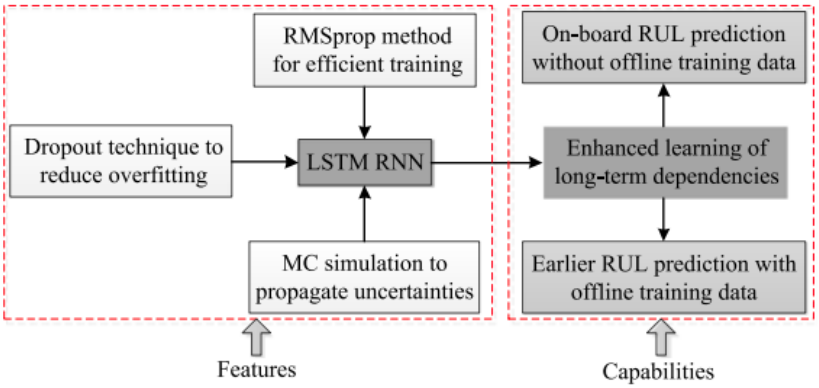


Figure 36: Method proposed in paper [12]

form existing technologies even when considering more realistic conditions with respect to the assumption made in other works.

- "Masked Self-Supervision for Remaining Useful Lifetime Prediction in Machine Tools" [25]. In this article, the self-supervised pre-training for RUL prediction is deepened. The proposed method is based on a masked transformer-based autoencoder, which is trained both to learn a good feature representation of a sequence and to reconstruct the masked patch of it. The encoder is then used as feature extractor for the regression step. The authors implemented and compared the proposed approach considering different masking ratios, concluding that the proposed architecture performs better than fully supervised methods. Figure 38 shows the proposed approach.

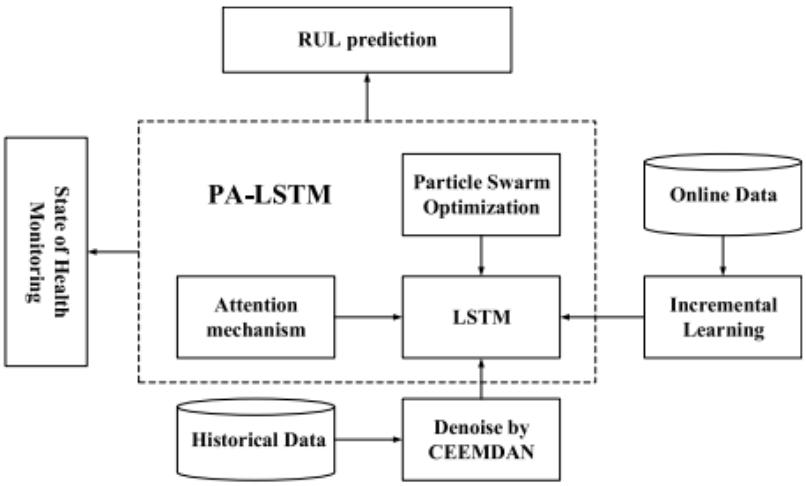


Figure 37: Method proposed in paper [14]

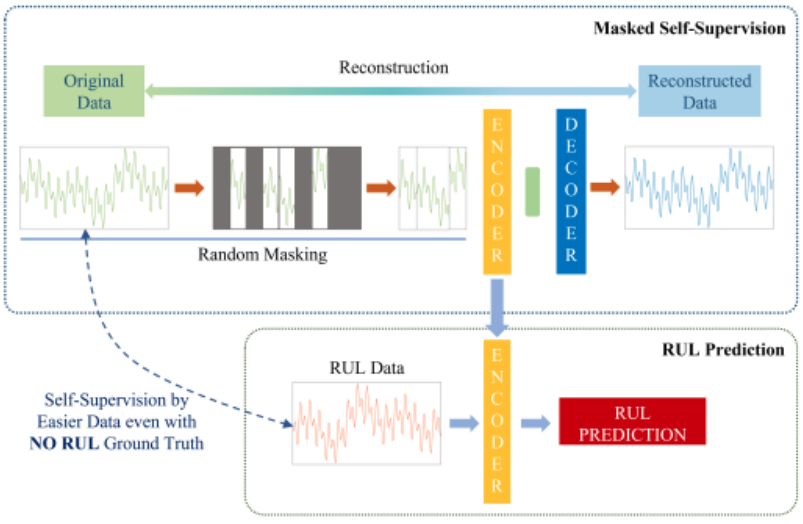


Figure 38: Method proposed in paper [25]

9 Conclusions

The smart monitoring of a device is a very active field of research: being able to predict a fault of a machinery before the actual break allows to avoid critical situation in which the failure can be both expensive and dangerous. For this task machine learning technologies have been proven very effective, since they allow one to define very accurate predictor without requiring too much expertise on the field, making ML one of the most promising approaches for predictive maintenance. This work mainly focused on applications of machine learning techniques for the monitoring of smart glasses battery; a real-world problem that was faced, along with all its challenges, from the data collection to the actual experiments. Data collection, in particular, was the main bottleneck of all the work, considering that it took a long time to be done, it was impossible to collect a complete dataset and there were also difficulties with the connection between devices and computers. The initial part of the thesis describes the application of unsupervised anomaly detection and its interpretation to defective devices. It was interesting the presence of a lot of anomalies in the last battery cycles preceding the break, but it was also interesting the interpretation part, which highlighted the sensors that could be better exploited for monitoring. In the second part the application of classic ML approaches to the two working device available was deepened; the preliminary analysis on RUL prediction performed, along its interpretation, better highlighted the challenges related to this work, in particular regarding the dataset. The interpretation of the models and the feature selection based on such interpretation is an approach that showed promising results. Even if some decent results were obtained, the inability of the models to well generalize made clear that the lack of data and the lack of features that directly describe the battery were really a problem for this task. A second platform was built to collect samples with more descriptive features of the battery. In addition, six other Isee devices were made available for data collection. In the last part of the work, advanced experiments were performed, with a particular focus on deep learning. Even if a lot of different approaches were tested, it seems that, with the available datasets, the performance of the

predictive models cannot improve, highlighting once again the limitations imposed by the available data. The not recorded break of the battery, the discontinue data collection and the partial samples collected are the main bottlenecks of the work and the obtained results, while valid as preliminary analysis for the task, could not be improved more. Still, from what was possible to be seen from these experiments and even considering the literature, future works that address the data collection problem may find an effective and numerically stable predictor that could be actually applied to the task.

Acknowledgments

I would like to thank my supervisor Professor Gian Antonio Susto, who assisted and guided me through this project. I would also like to thank Professor Susto's team for the help they gave me in finalizing this project, in particular Dr. Eugenia Anello, Dr. Jacopo Andreoli and Dr. Davide Dalle Pezze. Lastly, I would like to express my gratitude to Luxottica and its staff, especially to Dr. Paolo Giavarini, Dr. Dino Michelon and Dr. Massimo Reineri.

References

[1] *The Shapley Value: Essays in Honor of Lloyd S. Shapley*. Cambridge University Press, 1988. DOI: 10.1017/CBO9780511528446.

[2] Yoav Freund and Robert E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *Computational Learning Theory*. Ed. by Paul Vitányi. Springer Berlin Heidelberg, 1995.

[3] Leo Breiman. "Random Forests". In: *Machine Learning* (2001). URL: http://dx.doi.org/10.1023/A%5C%3A1010933404324.

[4] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. "Isolation Forest". In: (2009). DOI: 10.1109/ICDM.2008.17.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* (2011).

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107298019.

[7] Francois Chollet et al. *Keras*. https://keras.io. 2015.

[8] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[9] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. "All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously". In: (2018). URL: https://arxiv.org/abs/1801.01489.

[10] Scott M Lundberg et al. "Explainable machine-learning predictions for the prevention of hypoxaemia during surgery". In: *Nature Biomedical Engineering* (2018).

[11] "Remaining useful life estimation in prognostics using deep convolution neural networks". In: *Reliability Engineering and System Safety* (2018). DOI: https://doi.org/10.1016/j.ress.2017.11.021.

[12] Yongzhi Zhang et al. "Long Short-Term Memory Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-Ion Batteries". In: *IEEE Transactions on Vehicular Technology* (2018). DOI: 10.1109/TVT.2018.2805189.

[13] Lahiru Jayasinghe et al. "Temporal Convolutional Memory Networks for Remaining Useful Life Estimation of Industrial Machinery". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. 2019. DOI: 10.1109/ICIT.2019.8754956.

[14] Jiantao Qu et al. "A Neural-Network-Based Method for RUL Prediction and SOH Monitoring of Lithium-Ion Battery". In: *IEEE Access* (2019). DOI: 10.1109/ACCESS.2019.2925468.

[15] Yongyi Ran et al. *A Survey of Predictive Maintenance: Systems, Purposes and Approaches*. 2019. DOI: 10.48550/ARXIV.1912.07383.

[16] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[17] Mattia Carletti, M. Terzi, and Gian Antonio Susto. "Interpretable Anomaly Detection with DIFFI: Depth-based Feature Importance for the Isolation Forest". In: *ArXiv* abs/2007.11117 (2020).

[18] Kevin Fauvel et al. "XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification". In: *CoRR* (2020). URL: https://arxiv.org/abs/2009.04796.

[19] Christian Krupitzer et al. "A Survey on Predictive Maintenance for Industry 4.0". In: *CoRR* (2020). URL: https://arxiv.org/abs/2002.08224.

[20] Zinan Lin et al. “Using GANs for Sharing Networked Time Series Data”. In: *Proceedings of the ACM Internet Measurement Conference*. 2020. DOI: 10.1145/3419394.3423643.

[21] Scott M. Lundberg et al. “From local explanations to global understanding with explainable AI for trees”. In: *Nature Machine Intelligence* (2020).

[22] James C. Chen et al. “Combining empirical mode decomposition and deep recurrent neural networks for predictive maintenance of lithium-ion battery”. In: *Advanced Engineering Informatics* (2021). DOI: https://doi.org/10.1016/j.aei.2021.101405.

[23] James C. Chen et al. “Combining empirical mode decomposition and deep recurrent neural networks for predictive maintenance of lithium-ion battery”. In: *Advanced Engineering Informatics* (2021). DOI: https://doi.org/10.1016/j.aei.2021.101405.

[24] Tilman Krokotsch, Mirko Knaak, and Clemens Gühmann. “Improving Semi-Supervised Learning for Remaining Useful Lifetime Estimation Through Self-Supervision”. In: *CoRR* (2021). URL: https://arxiv.org/abs/2108.08721.

[25] Haoren Guo et al. *Masked Self-Supervision for Remaining Useful Lifetime Prediction in Machine Tools*. 2022. DOI: 10.48550/ARXIV.2207.01219.

[26] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book.

[27] “MSWR-LRCN: A new deep learning approach to remaining useful life estimation of bearings”. In: *Control Engineering Practice* (2022). DOI: https://doi.org/10.1016/j.conengprac.2021.104969.

[28] *Ceruleo Documentation*. https://lucianolorenti.github.io/ceruleo/.

[29] *Ceruleo Github*. https://github.com/lucianolorenti/ceruleo.

[30] *GradientBoostingRegressor*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor.

[31] Shahid A. Hasib et al. "A Comprehensive Review of Available Battery Datasets, RUL Prediction Approaches, and Advanced Battery Management". In: *IEEE Access* (). DOI: 10.1109/ACCESS.2021.3089032.

[32] *permutation_importance*. https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html#sklearn.inspection.permutation_importance.

[33] *RandomForestRegressor*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor.

[34] *What is predictive maintenance?* https://www.ibm.com/uk-en/services/technology-support/multivendor-it/predictive-maintenance.