UCLouvain

École polytechnique de Louvain

Predictive maintenance: predict upcoming failures via machine learning

Author: **Félix DE PATOUL**
Supervisor: **François GLINEUR**
Readers: **Valentin HAMAIDE, Raphaël JUNGERS**
Academic year 2021–2022
Master [120] in Mathematical Engineering

Preface

This master thesis has been written to fulfill the graduation requirements of the Mathematical Engineering program at UCLouvain.

Firstly, I would like to thank the supervisor of this master thesis, Professor François Glineur, for his constructive comments, guidance and availability.
In the second place, I am grateful to Valentin Hamaide for his precious help and useful feedback. He was always ready to answer my questions and allow me to advance when I remained blocked on specific details. Without his explanations, the final results of this thesis would not have been possible.
Finally, I would like to thank the last jury member, Professor Raphaël Jungers, for taking the time to read this manuscript and being present at the final presentation.

I wish all readers a constructive reading and hope this thesis will bring you clarifications on predictive maintenance of hard drive disks.

Félix de Patoul
Brussels, June 2022

Abstract

Predictive maintenance is an increasingly important research area in accordance with the emergence of Industry 4.0. Since Hard Drive Disks (HDD) are among the most commonly replaced hardware components in today's IT environments, their replacements are perfect to illustrate the efficiency of such predictive tools for maintenance.
Although this subject is already well-referenced and many studies can be found in the literature, most of them propose a machine learning approach where the decision to replace a disk is made by machine learning methods.

In this thesis, we propose to follow the two-level framework developed by Valentin Hamaide and his team [11], where the decision-making is handled separately from the machine learning methods. For this approach, in the first level, we build a health indicator which is a value that is mapped from pertinent SMART features [18] and represents the health status of a disk. Depending on this value, an alarm would be triggered in the decision-making part of the second level.
If we restrict our predictor to the work of Valentin Hamaide and his team, when we pull the trigger, it is a definitive decision. In this document, we propose to add a probability measure. This measure allows limiting prediction failures when the health indicator exceeds the threshold at one moment but shows better values the time after. Furthermore, the probability measure highlight failing disks which have recurrent small health variations.
Using this two-level approach and the new probability based decision allows us to build a predictor with a *F1_score* of 0.5965 for a multi-class SVM classification method. This predictor performs well compared to literature and this score is improved by 0.0202 due to the use of probability. The data set used for this thesis come from Backblaze [1] and contains 7058 runs of 91 days.

Contents

Chapter 1

Introduction

This thesis will present a global and efficient way to apply practical solutions for the maintenance of an industrial product. Specifically, we will focus on the predictive maintenance of hard drive disks in a data center.

Predictive maintenance is a key element of industry 4.0. It consists of anticipating future failures in equipment, objects or systems thanks to the accumulation of a data set of information. It allows several advantages, compared to corrective maintenance. Indeed, while corrective maintenance applies after failures occur, predictive maintenance prevents a production system from being immobilized when an element breaks down. Compared to preventive maintenance, where we replace the equipment before the failure without considering data that represent the individual status of each machine, predictive maintenance allows us to keep track of the life cycle of each piece of equipment individually and replace the material when it is really necessary.
This technique makes it possible not to stop the industrial process during a breakdown while allowing to reduce costs by avoiding changing parts when it is unnecessary.

As mentioned in the first paragraph, for this thesis, we will focus on the failures of hard drive disks from Backblaze [1]. The causes of their failures could be media damage (physical damage from impact to the drive itself), printed circuit board failure, stiction (when disks stop operating after a long period unused) and motor failure [6]. A disk failure could lead to data alteration or loss. Such an event is intolerable for the professional or industrial environment.
This domain is subject to various studies and prediction methods. Most of them propose prediction methods based on machine learning classification algorithms such as artificial neural networks, search trees or k-nearest neighbors. It mainly gives satisfying prediction scores. Most of the main studies propose to add new

elements and ideas to the existing works and will be detailed in the state of the art section.

In this thesis, we propose to follow two main goals which would differentiate our study from works already available in the literature and enhance the results of our predictors.

In the first instance, we will apply a two-level framework for predicting failures. The goal of this part is to apply the predictive methods employed by Valentin Hamaide's team in their paper: "A two-level machine learning framework for predictive maintenance: comparison of learning formulations" [11]. The first objective is to observe if this framework would enable us to obtain the same satisfying results, for another predictive maintenance problem with a different data set.
At the first level, machine learning models would be used to map a set of features into one value representing the health of the disk at a specific moment. Then, at the second level, based on this health indicator, a decision would be made about raising an alarm or not, depending on an optimized threshold. Deeper explanations are provided throughout the body of this document.

For a second purpose, this thesis proposes to add a probability measure in the second level of the predictor to improve the predictions. Indeed, in the second level process, we look at the health value of the disk in time and if it exceeds a threshold, we consider the disk as failed and raise an alarm. This decision is deterministic and definitive in time. On the contrary, using probability, if the health value exceeds a threshold, we consider a growing probability of pulling an alarm. This probability will depend on how far the value is beyond the threshold. Consequently, this new measure should limit the influence of a unique exceeding health value in a run (isolated event) and allow small recurrent health variations in time to accumulate in a bigger probability of being considered responsible for a failure.

Through the rest of the document, we will explain the data and the choices made to build the predictors in chapter 2. We will browse the main literature articles in section 3, before explaining our methodology in section 4. Then, in chapter 5, we will present our results, main interpretations, and comparison to the same literature. Finally, we will finish with possible future works and a general conclusion.

Chapter 2

Data Preprocessing

In this chapter, we will present the data we used for our study and the choices we made to apply machine learning models on these data. Since the goal of this thesis is to predict failures on an industrial machine, we got interested in the well-known big data center containing thousands and thousands of hard drives subject to possible problematic failures.

2.1 My data

In order to perform our study on machines for which there would be failures, we use the Backblaze[1] hard disk data collected once a day during the first 3 months of 2016. This period corresponds to 91 days between 01-01-2016 and 31-03-2016. These data are available on the Backblaze website and are part of a Kaggle contest regularly studied [2].

In this document, we will call a run, the data collected for a machine in this three months time lapse. A run has therefore 91 samples when it is complete and could be smaller if a failure occurred in the observed period. Indeed, after a failure, the data are not collected anymore. One collection of data will be called a sample while one specific information for these samples will be called a feature.

Each sample provided by Backblaze contains 95 features:

- The first feature is *date*: the day on which the data was collected. By default, we set the collection to midnight but it is possible that the collections are variable during the day. This difference will not be considered important.
- The second and third features are *serial_number* and *model*. The serial

number allows us to identify exactly a hard disk drive while the model should give us diversity in the fabrication of the disk. A run is therefore defined by a unique serial number.

- Then we have the feature called *capacity_bytes* which is the memory capacity of the hard disk. This feature varies from model.

- The fifth column gives information about the *failure*: for each sample, it says if it is the last day of collection before the failure of the hard disk occurs. A run for which there would be a failure after 36 days of observation, would thus have a length of 36 samples and a value of 1 only in the *failure* feature of the 36th sample.

The following 90 features are the S.M.A.R.T (Self-Monitoring, Analysis and Reporting Technology) attributes of the hard drive [18]. On the Wikipedia page, we can read *"It is a monitoring system included in computer hard disk drives (HDDs), solid-state drives (SSDs) and eMMC drives. Its primary function is to detect and report various indicators of drive reliability with the intent of anticipating imminent hardware failures."*
According to this description, these attributes seem to be the perfect features on which we will base our study to prevent failures. For example, the feature can be the temperature, the reallocated sectors count or the the spin-up time. They are either the absolute value of the attribute called the "raw" value. Raw values are collected directly from sensors in the disk and their interpretation can vary from one manufacturer to another. Either the value is normalized between 1 and 253, with 1 representing the worst value possible for the attribute and 253 representing the best. The initial default value of the attributes is 100. These “normalized” features actually are not related to any statistical normalization. They are instead a specific manufacturer interpretation indicating the health of a drive.

2.2 Choices

The database available on the website of Backblaze (2016 Data, Q1)[1] contain 3179295 samples for 95 features.

Since this database is very large, the first decision was to focus on a single hard drive model. Among the 69 models available, we will focus on the model "ST4000DM000". This is the most represented model in the database (53%). By selecting only one model for our study, we want to make sure that the causes of failures do not differ for a reason of manufacture or brand. According to [15], the

SMART collecting implementations still differ from one manufacturer to another and the SMART attributes may be missing depending on the models. Nevertheless, limiting our study to one model will not really be heterogeneous in regards to the disk population. Besides, because of this choice, the *model* and *capacity_bytes* columns have no more use, as they are similar for the whole new dataset.

As explained in the previous section, attributes are split in two categories: "raw" and "normalized". These are in fact different scale possibilities for a SMART attribute and both of these scales could be used. In this paper, we choose the normalized scaling to allow the attributes to be considered with the same importance during future transformations. Some papers such as [9], use raw and normalized features even from the same attributes. For our study, we will prefer to avoid this, since it could be redundant to use the same information, scaled differently. The number of attributes is thus divided by two and 45 features remains available.

2.2.1 Feature selection

This amount of features seems still too heavy for a predictive study. We now have to proceed with a feature selection and try to base our choices on the relevancy of each attribute for our goal to predict failures. The main reason why we select features is to build better-performing models and to reduce the computational cost of modeling.
After browsing the literature, some attributes seem to be more relevant to failure prediction. In the paper [10], the research use: *smart_5*, *smart_187*, *smart_188*, *smart_196*, *smart_197* and *smart_198*. Among these 6 features, 5 are considered by BackBlaze and Google as being the most correlated to failures [8]. The study Google and BackBlaze made is in the univariate case i.e when we look only at the feature itself, without combining it with other attributes. Finally, in the article [9], they add *smart_3*, *smart_7* and *smart_190* to their multivariate analysis.

In our data set, the *smart_196* attribute is filled with null values and the *smart_198* is identical point to point to the *smart_197*. These two features are therefore useless. For this study, we decided to follow the work made by the articles cited in the previous paragraph and keep the attributes *smart_5*, *smart_187*, *smart_188* and *smart_197* for the univariate study while adding attributes *smart_3*, *smart_7* and *smart_190* when predicting in multivariate. More information about these features is available on Wikipedia [18] and given in table 2.1.

SMART	Attribute name	Description
3	Spin-Up Time	Stores data related to the rate of hardware read errors that occurred when reading data from a disk surface.For some drives, this number may increase during normal operation without necessarily signifying errors
5	Reallocated Sectors Count	Count of reallocated sectors. The raw value represents a count of the bad sectors that have been found and remapped. Thus, the higher the attribute value, the more sectors the drive has had to reallocate. This value is primarily used as a metric of the life expectancy of the drive. A drive which has had any reallocations at all is significantly more likely to fail in the immediate months.
7	Seek Error Rate	Rate of seek errors of the magnetic heads. If there is a partial failure in the mechanical positioning system, then seek errors will arise. Such a failure may be due to numerous factors, such as damage to a servo, or thermal widening of the hard disk. For some drives, this number may increase during normal operation without necessarily signifying errors.
187	Reported Uncorrectable Errors	The count of errors that could not be recovered using hardware ECC
188	Command Timeout	The count of aborted operations due to HDD timeout. Normally this attribute raw value should be equal to zero.T
190	Temperature Difference	Value is equal to (100-temp. °C), allowing manufacturer to set a minimum threshold which corresponds to a maximum temperature. This also follows the convention of 100 being a best-case value and lower values being undesirable. However, some older drives may instead report raw Temperature or Temperature minus 50 here.
197	Current Pending Sector Count	Count of "unstable" sectors (waiting to be remapped, because of unrecoverable read errors). If an unstable sector is subsequently read successfully, the sector is remapped and this value is decreased. Read errors on a sector will not remap the sector immediately; instead, the drive firmware remembers that the sector needs to be remapped, and will remap it the next time it has been successfully read. However, some drives will not immediately remap such sectors when successfully read; instead the drive will first attempt to write to the problem sector, and if the write operation is successful the sector will then be marked as good.

Table 2.1: Description of the relevant SMART features for predictive machine learning

The normalized values of the SMART attributes we keep for our study are included between 1 and 100. At an initial healthy phase, the values are 100. When it gets worst, possibly leading to a failure, the value should get smaller. During our study, all these attributes should ideally be as high as possible to have a healthier disk.

Among the data set, there were very few missing values in the features (23 in total). We replaced them by taking the value of the previous sample to replace the null value.

2.2.2 Train, validation and test

After pruning our data set, there now are 34988 runs left with 200 of them leading to a failure. This amount represents 0.574% of failure. Among the disks that do not fail, we keep 28792 that are complete, i.e. they have 91 samples. Among the 200 disks that will fail, 40 have a failure that happens in the first 15 days. We decide not to consider them for our study, because in this case, we don't have enough information before the failure and enough time to predict it.

In order to increase the proportion of disks that end in a failure, we keep only a quarter of the 28792 disks, so 7198 disks. Keeping too many healthy runs would have been redundant information and would have hidden the information of failing run besides leading to bigger computational cost. The proportion of failure raises thus to 2.22%.

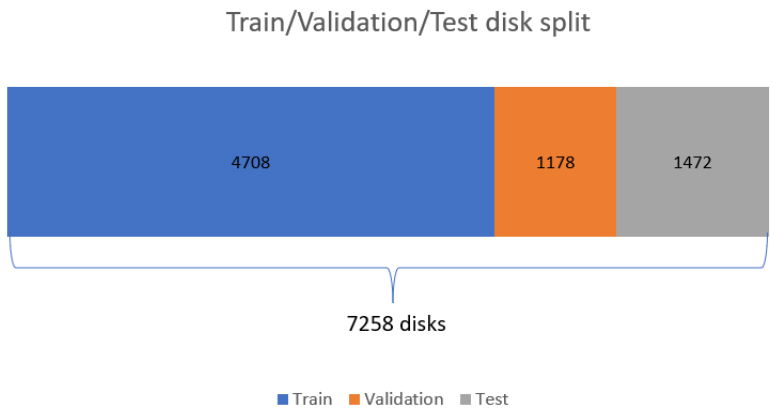


Figure 2.1: Train/Validation/Test repartition of hard drive disks

While keeping this failure proportion, we then divide these 7258 disks into 5886 train disks which represent 80% of the total, and 1472 test disks (20%). In the group of training runs, we finally split it into 80% of train disks (4708 runs) and 20% of validation disks (1178 runs). This split strategy, shown in figure 2.1, will be useful during the machine learning methods we explain in section 4.1 but also for tuning the hyperparameters in section 4.4.

Chapter 3

State of the art

In this chapter, we gather a good amount of what the literature can already conclude about the predictive maintenance applied to the Blackbaze data set, from a selection of articles.

The document [11] is the main work on which we will base our studies. It contains practically the same methodology and scoring function we will use. In the article [7], the authors use the same data set that we use in our study and select the same unique hard drive model, over a bigger period. They use a gradient boosting algorithm and propose to incorporate historical context into each sample. The study [15] proposes a cost-effective intelligent backup system for the same hard drives as ours, whereas another work [9] uses the Backblaze data on a hard drive model different than ours. This article uses Optimal classification trees for long-term and short-term prediction. In the document [10], they explore the study using several hard drive models and decision trees as their machine learning method. Finally, in [13], the authors add two new categories of features to the SMART attributes and propose one of the largest known database studies.

Later in this thesis, it will be interesting to compare the results obtained from those articles to what we obtain in section 5.

In another article, we can find a global literature review of the recent advancements in predictive maintenance approaches for industrial equipment in "the fourth Industrial Revolution" [19]. The information they gather is more global and less specialized in the disk failure domain.

3.1 Work inspiration

The main goal of this thesis is to apply the method developed by Valentin Hamaide and his co-authors on predictive maintenance [11], to another data set.
This paper proposes a general formulation of the predictive maintenance problem. In their study, Hamaide's team aims to focus more on the modelization of the predictive maintenance problem rather than on the methods used to predict failures. For this reason, they choose to use traditional machine learning methods rather than deep learning. As explained by the authors, what differs their work from other papers in the literature, is their desire to tackle the decision-making part of the problem. In order to achieve this goal, they "propose a framework divided into two levels, the first level consisting of a machine learning model mapping a set of features into a health indicator and the second level being responsible for the actual decision making, where an alarm is raised if the health indicator of the first level crosses a threshold whose value can be optimized".

In their study, they exploit data collected in form of time series, which is also the case for the Backblaze data we use for our study. Their data are collected from a high-speed rotating machine. The main difference between our two data sets can be summarized as follows: time-frequency and number of runs. Indeed, they mainly collect data each second and aggregate these into one hour time-windows whereas our collection of data is made once a day for 3 months. Secondly, while they perform a predictive maintenance algorithm over 11 runs running to failure among a total of 39 runs, we have access to 7258 runs with 160 leading to failures. Since one objective of this thesis is to apply the work of this paper to another industrial problem, it is interesting to test whether or not the same framework applies to a different data set.

For their first level, they compare three main categories of methods to build a unique health indicator. As we can see in figure 3.1, the first category is simply following a single feature. We will call this category "univariate". If we simultaneously use multiple features, it would be "multivariate". The second category is anomaly detection. For these multivariate methods, we train the model on healthy runs. Therefore, the model learns what it should consider healthy. Then, it would identify data points in the test data that don't fit the normal patterns. Finally, the last category is supervised machine learning where the multivariate methods are trained on healthy and unhealthy runs. All these algorithms are explained in further detail in section 4.1.

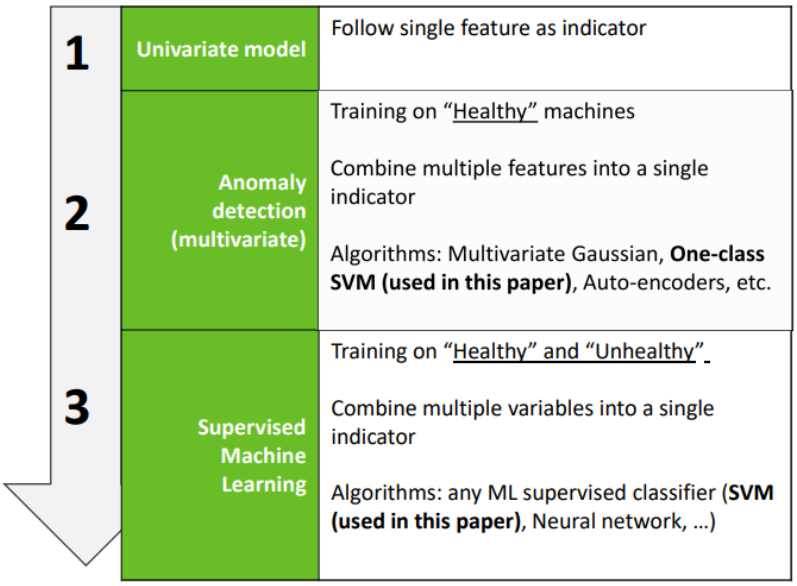
In their second-level section, the authors introduce a decision-making step on the health indicator built in the first level. During this process, the decision maker will trigger an alarm if the health indicator exceeds a previously optimized

threshold, as illustrated in figure 3.2. The values they use in this level can be the health indicator values themselves or aggregation of them, like their moving average or exponential moving average.

To assess the predictive performance of the models, the authors build scoring measures to evaluate the models. First, they define a scoring metric they call *F_score*, which is a trade-off between false alarms and failure detection. Then, they create a business score that encodes the best timing for detecting a failure. These metrics are explained in section 4.2. In order to validate their models and tune their hyperparameters, they then use cross-validation. In fact, because of the scarcity of their database, they perform what they call double cross-validation. Since, in our case, we have enough failure in our database, we will not use the double cross-validation but we encourage you to look for more details in their paper [11].

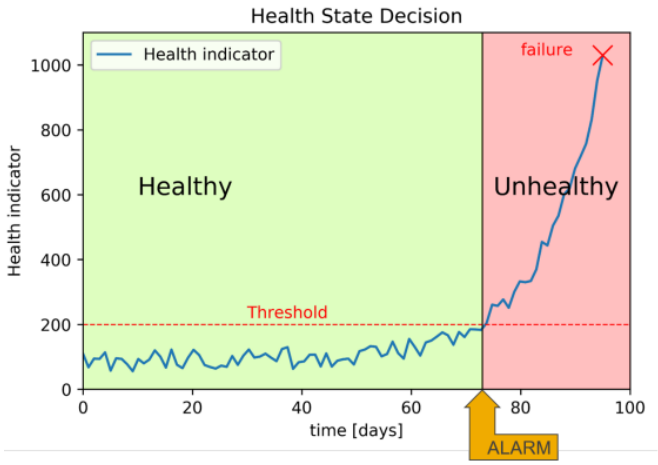
After explaining their methodology, the authors present convincing results for several models. Their best model is a binary classification model with a horizon of 10 days. They obtain a *F_score* of 0.919. Meanwhile, what surprised them is that the simplest model they build, the univariate case, gives them very good results *F_score*=0.894. "It can already be effective provided that a health indicator with high predictive power has been found." They also concluded that "a supervised learning algorithm should lead to better performance than a one-class classification or anomaly detection algorithm. The binary classification formulation yields the best results when the time-window splitting healthy and faulty data is carefully selected."

While we will not compare their results to ours because of differences in our respective data sets, it would be interesting to see if their conclusions apply to our study.



(a) First level: Three refinement of the ML model

Figure 3.1: First level: 3 categories of methods for translation of a set of features into a single indicator [11]



(b) Second level: Decision maker

Figure 3.2: Second level: decision making to raise an alarm when the health indicator exceed an optimized threshold [11]

3.2 Adding historical context

This article [7] is meant to explain the basics of predictive maintenance using the same database, we used. Indeed, they use the database of Backblaze with SMART features. Compared to us, they collected all samples from 2013 to 2019 and use BigQuery, a tool available on Google Cloud, to deal with Big Data. However, to reduce the amount of data, they will also select the same unique manufacturer model: "ST4000DM000".

In their article, they first proceed to data visualization to illustrate the differences between failing runs and healthy runs. They gather the 30 last days of all their failing runs and compare them to the same amount of healthy runs. In the article, they choose to compare the mean of *smart_5*, *smart_187*, *smart_188*, *smart_197* and *smart_198*. From these univariate statistical results, they could conclude a simple predictive maintenance procedure where they should investigate the drive when the "raw" value for one of these five SMART stats is greater than zero. This procedure was already highlighted by the Google study [8].

After analyzing the features, they proceed to machine learning methods. They aim to do four binary classifications over a horizon of 1, 2, 7 and 30 days. the machine learning model they use is the "XGBoost", a gradient boosting algorithm [5].

What is interesting about this paper, is its attempt to incorporate historical context by doing some aggregations of the samples (mean, variance, min, max and last value) over a 10-days window. They analyze whether or not it could improve the prediction.

In their results, we can see that adding historical context leads to better results and that their machine learning methods perform better than the univariate case they presented in their statistical study. The problem is that they did naive downsampling to avoid a heavy class imbalance. Indeed, they kept 50% of failure proportion in the data they trained, while we kept 2.22% in this thesis. It leads to particularly good results [7] for their predictor but doesn't represent the actual hard drive maintenance situation.

Indeed, later in their paper, they actually test their predictor on the 2019 data and obtain other metrics which are less promising. Adding the historical context lead them to a consequent increase in false alarms. They conclude by saying their study does not go deep into the predictive possibilities but gives a good start in tackling the Backblaze maintenance problem.

3.3 Backup system

This paper [15], as the previous one and our study, will use SMART attributes to train their machine learning models. They use the same database from the same Kaggle post we were inspired for our data acquisition. The main part of this paper explains statistical methods such as t-test, F-test or attribute vs. attribute plot, to proceed with a feature selection and reduce the dimensionality of the problem.
They then present partial results for four main ML methods: AdaBoost, Random Forest, Logistic Regression, and Naive Bayes. The results they show are not explained in detail and besides the AUC (Area Under the Curve) of the ROC curve, they didn't share any other metrics.
Finally, they want their model to be implemented to achieve a cost-effective intelligent backup system. Currently, they claim that for each drive, a backup disk exists to duplicate the data. They study the possibility to accept a little bit of risk to save place for backup data in the current redundancy system used to avoid loss of data. They suggest that reducing the number of backup disks by half will lead to a probability of losing data of 8.6% in a year. This is the probability that during the current year, at least one drive not backed up will fail. This backup system shows a good example of limitations and possible improvements of corrective maintenance.

Although the results they supply may seem encouraging to look deeper, the authors did not prove their numbers rigorously and give no more information about how they got their results.

3.4 Different disk model

In this article [9], the authors described the machine learning methods they used to predict failures of Backblaze hard drive disks.

They did their study on the Backblaze data set from Q1 2019 to Q1 2020 which has the same SMART features as those used in our case. However, they decide to keep both the raw and normalized values. According to them, this could allow their models to have more predictive power. As in our case, they also decide to base their research on a single hard drive model. However, while our choice of disk model is based on the frequency of the disk model in the database, they choose the model "ST12000NM0007" which is the model with the highest proportion of failure. As you can see in the figure 3.3, our studied model "ST4000DM000" is still the model with the second-highest proportion of failures among all the runs available in their database.

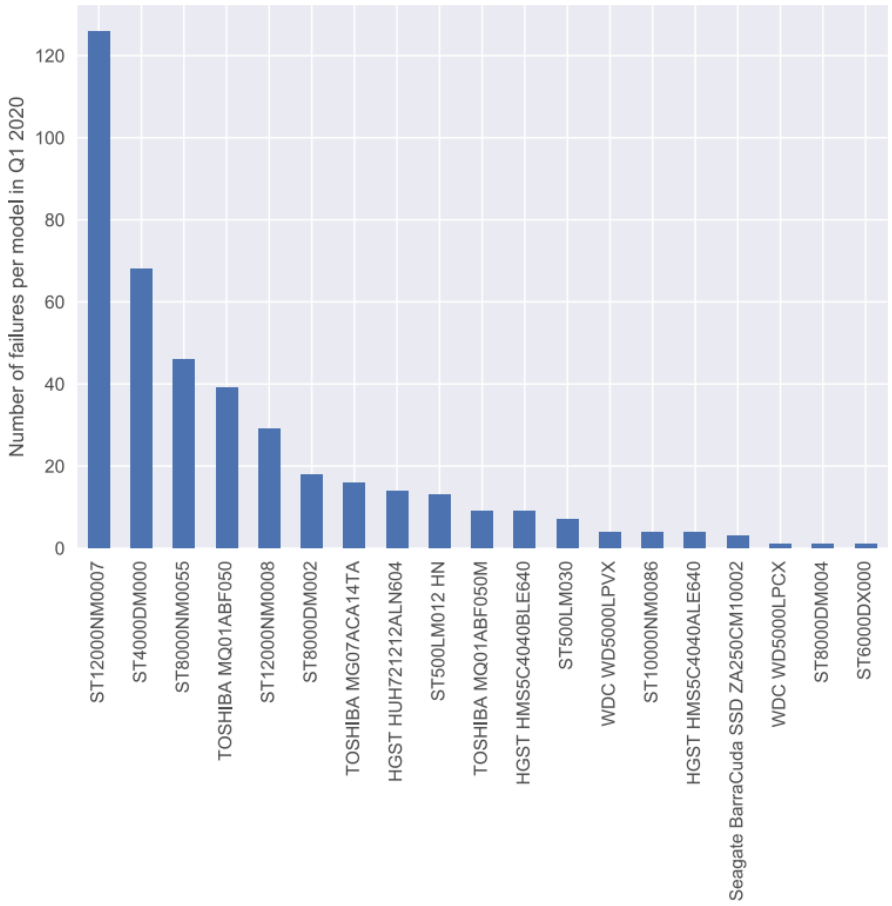


Figure 3.3: Number of failures per disk model in Q1 2020 [9]

In this document, the authors of this paper perform their research in two different approaches: long-term health and short-term health.

In the long-term approach, their purpose is to predict the overall health of the disk over its lifespan. They observe the 3 years of SMART features that precede the records leading to a failure in Q1 of 2019 and 2020. These three years represent in most cases, the entire lifespan of the failing disks. They calculate for each drive the remaining useful life (rul) which is the number of days remaining before the day of failure. This *rul* value will be used as a target value in the training of their Optimal Survival Tree (OST) model. The goal of this model is to maximize the expected survival times.

The second approach is the one that interests us the most in our case, because it deals with the study of predictive maintenance over a 90-day window. As in our study, they look at both the disks that fail and those that are healthy during the 90-day window. Again, they use an Optimal survival Tree (OST) that displays the expected survival times for failing disks. In this way, they measure the health of the drive over the entire time window of 90 days. In order to be more accurate in their prediction, they perform a binary classification with a healthy class and another class that corresponds to the last 30 days before the failure. To perform the classification, they build an Optimal Classification Tree (OCT) which, according to them, permits to construct a single decision tree that has similar performance to the classical black-box methods. The big advantage of their model compared to these black-box methods would be that it delivers interpretability without sacrificing performance. Indeed, as they mention in their conclusion: "They detect interpretable paths to failure. Compared to Random Forest and other black-box models, Optimal Trees can automatically identify both paths leading to accelerated failure as well as paths indicating healthy behaviors".

We will not enter into details about the functioning of the OST and OCT. But I encourage you to read the mathematical explanations in the papers [9]. What matter to us in this document is the metrics of their best predictors.

We look a the results they presented of the OCT model for binary classification of 30 days. This model is the most fitted with our goal to predict if a failure could occur in a run of 91 days. Moreover, this model gives them their best results and therefore would be the most interesting to compare in section 5.

Their predictor has a precision of 0.4432, an accuracy of 0.8614, a true positive rate of 0.5468 and a false positive rate of 0.1185. The main metric we consider to evaluate the quality of a classification predictor is the *F1_score* which is valued at 0.4896. All these metrics are explained in section 4.2 and will be metrics used to compare with our predictors.

3.5 Heterogeneous disk population

In the paper [10], the authors do global research for predictive maintenance with machine learning predictors which are trained on the Backblaze data set, for the whole year 2015 and tested on 2016 data.

The main difference compared to this thesis, is the fact they consider the focus on one model as a limitation. Indeed, they want their results to be applicable over a heterogeneous disk population. However, this choice leads them to observe a lack

of standardization of SMART attributes between makes and models.

The SMART feature selection they proceed is based on multiple choices. First of all, they chose to keep only the raw value of the features to limit the differences in the normalization interpretation of the SMART features. Then they apply statistical analysis and trend test to keep 6 features: *smart_5*, *smart_187*, *smart_188*, *smart_196*, *smart_197* and *smart_198*. These trend tests allow analyzing if the failure is correlated to a trend of the specific feature. As mentioned in section 2.2.1, we based a part of our feature selection on this paper.

Then they proceed to a data set sampling to overcome the problem of unbalanced data. It results in a proportion of 22% of failures in their database which, compared with our work, seem not to be representative enough of the actual situation.

They try three usual machine learning models: Decision Trees (DT), Neural Networks (NN), and Logistic Regression (LR). In their results, it is observed that NN and LR are predicting all samples as negative leading to bad results. They then investigate the decision tree and obtain a true positive rate (TPR) of 52%, a precision of 0.5755 and a *F1_score* of 0.5475. When they analyze the timing of their true positive prediction, 72.56% are predicted in the month before the actual failure but 70% of this proportion is predicted in the week before the failure and thus may be a bit too late to have the time to proceed with the maintenance.

3.6 Performance and location features

In this article [13], the authors claim to present results for one of the largest disk failure prediction studies using three different types of data to reach a very reliable predictor.

Considering other studies of the literature as limited in sample size, they gathered a total of 380 000 hard drive disks across 64 different places, 5 manufacturers and over roughly 70 days.

They also claimed: "For the first time, this paper demonstrates that disk failure predictions can be made highly accurate by combining disk performance and disk location data with disk monitoring data (Self-Monitoring, Analysis, and Reporting Technology SMART data)."
Indeed, most of the literature, including our thesis, settles for a study using the SMART attributes as predictive features. In order to increase predictive power,

they add performance metrics which according to their study, show more variations much before the actual drive failure. By combining both approaches, they try to reach more powerful predictors.
Their improvement does not stop with these two approaches. They add another feature to improve their models: the location information of disk drives in the data center. "Disks in close spatial neighborhoods are more likely to be affected by the same environmental factors, such as relative humidity and temperature, which are responsible for accelerating disk component failures".

Figure 3.4: neighbourhoods of disks can influence their performances

In their paper, they study 14 different SMART features and five of them are in common with ours. They look at 12 disk-level performance metrics and 18 server-level performance metrics. For more details about the performance features see section 2.3 in [13]. For the unique location marker, each disk is attached to a server which is placed on a rack in a room on a geographic site. They are 64 sites for 199 rooms, 10440 racks and 120 000 servers. This location feature has a large window of possible markers.

In their paper, the authors try different machine learning models including naive Bayes, random forest and gradient boosted decision trees. More importantly, they build a "convolutional neural network long short-term memory (CNN-LSTM)" which combines CNN and LSTM. The CNN part allows them to select better features while the LSTM part would be effective for sequential data. They proceed to a binary classification with a time window of 10 days.

In their result section, they highlight the improvement obtained after adding

performance and location. Location improves however less than the performance measures and doesn't improve without the performance implied in the model. They found that their CNN-LSTM model performs the best with a *F1_score* of 0.95 for the model trained with the three categories of features. When they use only SMART features, as we will do in this thesis, the best *F1_score* is 0.58. It will be interesting to compare later with our results. Moreover, they observe that their method CNN-LSTM is better than the other model as long as it contains performance features. If it's not the case, the metrics are quite similar and the authors advise us to use the classic machine learning models which ask for less training time. They also observe that models based on SMART attributes have a very high false-negative rate. They miss a lot of failures compared to the model which includes the two other categories. Finally, they found that the location features have to be trained on several sites in order to test a new unseen location.

Although they build a new highly efficient model "convolutional neural network long short-term memory (CNN-LSTM)", they don't aim to focus on improving the machine learning models themselves but more on the "actionable insights, trade-off lessons learned in applying these models, and assessment of model robustness". Their paper seems to be the only one to include three types of data for predictive maintenance: SMART, performance and location. Above all, they demonstrate the predictive power of these new features.

Chapter 4

Methodology

In this chapter, we will explain how we build our predictive models. The choices and the methods were inspired by the work of Valentin Hamaide [11], the work on which this study is based.

In order to build a failure predictor with the data context explained in section 2, we can separate the work into two levels: the first level build a health indicator from the available features in the data set, this indicator will be used in the second level where we determine if it exceeds an optimized threshold. If it exceeds the threshold, we will trigger an alarm, meaning we have to replace the hard disk drive.

The first level will be fed with the training runs while we use the test runs, in the second level, to obtain a significant score depending on the threshold. This is a general procedure in the machine learning domain. By doing so, we make sure the score we use to compare the different models, would be independent of the data used to build these same models.

4.1 First Level

In level 1, the goal is to gather the information of each sample in the dataset to create a value that represents the health of the disk on this specific day. We consider three main methods to gather the information on several features: univariate, anomaly detection in multivariate and supervised machine learning in multivariate.

4.1.1 Univariate

In the univariate case, the feature itself is already the health indicator. We don't have to apply any methods to transform this value. We will analyze the data by

taking a single feature and deciding to raise an alarm when that feature exceeds a threshold. We will see further than in the two multivariate cases, we build our health indicator with machine learning functions from several features. As explained in the section 2.2.1, the features that would be relevant and on which we will base our analyse are *smart_5*, *smart_187*, *smart_188* and *smart_197*. Since the attributes we kept during the feature selection should ideally be as low as possible to reflect the good health of the hard disk, the optimal normalized value is 100.

4.1.2 Multivariate

The purpose of switching to multivariate is to combine the information that several features can give us, into a single health indicator. As explained in the section 2.2.1, the features we will use for these methods are the following: *Smart*03/05/07/187/188/190/197.

Multivariate methods can be separated into two main categories: supervised and unsupervised methods [12]. Supervised learning use labeled data sets. "These data sets are designed to train or “supervise” algorithms into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time."

On the other hand, unsupervised learning aims to detect anomalies in a healthy data set: "Unsupervised learning uses machine learning algorithms to analyze and cluster unlabelled data sets. These algorithms discover hidden patterns in data without the need for human intervention".

Several machine learning methods can be used to determine our health indicators. Following the comparison with the methods used in Valentin Hamaide's analysis [11], we will use in this study the Support Vector Machine (SVM) model for Binary Classification, Multi-Class Classification and OneClass SVM anomaly detection.

For the theoretical explanation of these methods, I refer you to the excellent explanations of Valentin Hamaide [11].

Binary classification

The goal of binary classification is to classify a set of data into two different classes. During this method, we fit a model with a *X_train* data set containing the features we want to combine and *Y_train* the corresponding label that the sample should have. Once the model is fitted, it can predict the value of the test data *X_test* and compare it to the label of these samples *Y_test* to determine the score of the

predictor.

In our case, we consider a healthy class, the class in which are the samples of the healthy runs and the data before x days preceding a failure. Then the second class is a failure class, where we find the data in the x days before the failure. Let's call these x days, the horizon. We will explore 3 horizons: 91 days which represents the whole run studied, 15 days and 7 days which are reasonable values for our objective of replacing the hard disk sufficiently early before failures.

So we take the data from the 7 features, mentioned previously, which are already separated into training and testing. We call them respectively X_train and X_test. Then we create a $target\_value$ column which contains the label that should be assigned to each sample. These labels would be the integer between 0 and 1, depending on the class within each sample should be. This gives us Y_train and Y_test.

Using the functions available in scikit-learn [4], we can create a support vector machine for classification model (SVC) using the default hyperparameters C and γ and a kernel rbf. It will be explained in section 4.4, how to tune these hyperparameters to have the best predictor. Once the model is created, we fit the model with the training data: it trains the model on X_train so that it has a maximum similarity with Y_train . This training part takes approximately 1300 seconds for the CPU of an "Asus ZenBook 14" to execute. After being trained, the model would predict the values that it returns with X_test. For a classical classification method, we have to compare this prediction with the desired Y_test values, to determine the different metrics of the model. In our case, our goal is to obtain a health indicator. So what interests us is the decision function of the model. Indeed, this function is positive or negative depending on whether we are in one class or the other. Furthermore, the value returned in the decision function corresponds to the distance of the sample features from the hyperplane that separates the two classes.

In our case, for the three horizons, the healthy class is positive and therefore the greater the distance, the further away from the failure class the feature is. The value of the decision function can therefore be considered as our health indicator which will have higher values for a healthier sample. It is this value that we give to level 2 and on which we base ourselves in order to determine if we raise the alarm or not.

Multi-class classification

For multi-class Support-vector machine (SVM), the principle remains the same. Except that this time, each sample is labeled in several classes. In this study, we choose to divide the data into 3 classes and 6 classes.
When dividing our data into 3 classes, we consider a healthy class as in binary classification, a class where the sample is between 7 and 15 days before the failure of the run and finally a class where the sample is within 7 days before the failure. Let's note this labeling 0/7/15. Equivalently, we proceed a labeling in 3 classes 0/15/30 and a labeling in 6 classes 0/5/10/15/20/25. For each possible time horizon where a sample should be classified, we give a number as a label and create a column, which we call *target_value*.

Once our *target_value* column is created according to the labeling explained above, we can again apply the classic machine learning methods of scikit-learn. We apply the model with a kernel "rbf", the default gamma and C. Then for multi-class, we had to transform this model into a *OnevsRest* model. This means that the model compares each time a class against all the others. To perform a multi-class classification, the actual SVM methods propose two main categories: *OnevsRest* and *OnevsOne*. In the first category, we split the multi-class problem into several binary classifications. We compare each class to the group of all the other classes. In this case, n_class classes would lead to n_class binary comparisons, one for each classes. In the second category, each class is compared to all other classes independently. If I have n_class classes, I would have $\frac{n\_class*(n\_class-1)}{2}$ binary comparisons. This would be more computationally heavy.

In the *OnevsRest* model, the decision function returns an array of n_class columns. The only positive column is the prediction of the model. Each value of each column represents the distance of the sample from the hyperplane that separates the respective class from all other classes. In order to have a unique measure of health indicator, we will proceed with a particular method for these different distances of the hyperplanes. In Valentin Hamaide's paper [11], the authors look at when one of the classes considered faulty has a positive decision value, thus when it is greater than the value of the healthy class. If this is the case, he raises the alarm anyway. In our case, if the model predicts a faulty class, we will give an artificially lower value for the health indicator, implying that it will be among the last values analyzed to raise the alarm. In other words, the optimized threshold should almost always be higher and therefore imply that if a sample is predicted faulty by the multi-class model in level 1, the alarm will be always raised when reading its health indicator in level 2.
Practically, to determine the unique health indicator, we look at which class the

value of the decision function is the largest. If it is a faulty class, we subtract a certain amount to this value in order to have a value below the healthy decision value in our health indicator. For example, for a multi-classification 0/7/15, if the decision function returns $[-0.8, -0.7, 0.9]$ for the class $[0-7, 7-15, healty]$, the lagest value is 0.9 in the third class, healthy. The health indicator would be $0.9 + (pos - 1) - (n\_class - 1) = 0.9 + 2 - 2 = 0.9$. If the decision function had returned $[0.8, -0.7, -0.9]$, it is a prediction in the 7 days before failure and the health indicator would be $0.8+(pos-1)-(n\_class-1) = 0.8+0-2 = -1.2$

In the three different multi-class cases we tried, it turns out that all values are predicted to be healthy. The distances of the healthy hyperplane compared to the rest of the classes are roughly between 0.5 and 1.1. This means that if we were only using the first level for our predictor, all runs would be predicted as healthy. The decision function allows us to have a nuance about how healthy the model considers the sample. Therefore, it would be a good health indicator.

One-Class SVM

This model belongs to the unsupervised category. This means that we will train it, on unlabelled data. During one class SVM, we train the model over only healthy samples. The training part takes approximately 2700 seconds for the CPU of an "Asus ZenBook 14" to execute. From these healthy samples, instead of using a hyperplane to separate two classes, the model will try to determine a hypersphere that encompasses all of the instances. It would try to fit the smallest hypersphere which can gather the healthy data and every data outside this hypersphere would be considered an anomaly.

Again, we build our model and then fit it with a healthy X_train. Once the model is fitted on healthy data, we predict X_test which contains failures. In our case, with our data set, approximately half the samples are predicted as an anomaly. It should be non-sense knowing we only have a proportion of 2.2% of failures. However, the goal of the first level is to build a health indicator. So we take the decision function of the one-class SVM model as this indicator. We will see later, in chapter 5, if it allows us to predict failure well in level 2.

4.2 Second Level

The second level is the process where the decision of triggering an alarm is made based on the different health indicators gathered from the first level. During level two, if a value in a run is below a chosen threshold, we would have to raise an

alarm to replace the disk. Depending on the day the alarm is fired and the day the failure occurs, we can consider the failure as well predicted.

While we fitted the models to the training data, the health indicator was built from the decision function of the test data. These are the indicators that are fed into level 2.

Concretely, the first thing we do in this process is to consider each value of the indicator as a possible threshold. By testing all the possible thresholds, we will determine each time a score for the predictor. Then the threshold which gives us the best score would be the optimal threshold chosen to be part of our predictor.

The phase when we try each threshold is the part of our predictor building process where the time variable has importance. Indeed, we look at the samples in the chronological order of the run. During this part of the level 2 process, we look for each run when a sample has a health indicator lower than the threshold as well as when the failure occurs. It is the difference between these two dates which will determine the status of the run. Several specific cases may happen:

- If the alarm is triggered but no failure occurs, the run will be counted as a false positive (FP).
- Inversely when a failure occurs without any health indicator below the threshold, the alarm has not been triggered so we count the run as a false negative (FN).
- If we both raise the alarm and have a failure in the same run, we call it a true positive (TP).
- If nothing happened when we look at the run, we didn't raise any alarm during the healthy run. It is a true negative (TN).

Level 2 variables

Predicting a true positive run can be considered, in our case, as the main goal of the predictor. However, predicting the failure 60 days before or 2 days before has not the same consequences. Indeed, since all the disks are devoted to failure at one time or another, predicting the failure today could be considered a true positive for all disks. Timing of prediction matters. To give importance to the date when we predict our failure, we add a variable $DayBefore$. An alarm triggered in the x $DayBefore$ would be considered now as the true positive. In opposite, if for a faulty run we trigger the alarm sooner than these x $DayBefore$, the run would be considered as

a false positive even if the run would lead later to a failure. In this study, we will consider 3 values for the variable $DayBefore$. The first case is 91 days representing the whole run. With this value, we consider that if we trigger an alarm on a faulty run, it is sufficient to consider the run as a true positive. Then the next two cases are 15 and 7 days before the failure. These periods can be considered as the maximum time window where the replacement of a disk is not considered to be too early.

Notice that the variable $DayBefore$ active in the second level is different from the horizon, we described in the subsection 4.1.2 for the first level. Indeed the first variable is implied in the metrics of the process while the latter variable will change the machine learning models which gives our health indicator.

Another nuance we can add during the second level is a moving average of the health indicator over a run. The goal is to smooth the value of the health indicator over time, before looking at it. In our case, we will study a rolling average over a period of 7 days. This means that for each sample in a run, the health indicator would be the average of the health indicators over the 7 days before. During the first 7 days of the run, the average is taken on the available samples of the days before. When we will be looking for the results, the variable *rolling* will be put at $True$ when we applied this modification.

Main metrics

When we looked at each run available, we count the number of true positive, false positive, true negative and false negative to build the main metrics of our models for a given threshold. From these values, we build several metrics that allow us to evaluate the quality of the predictor with the threshold studied. It is comparing these metrics which would give us the optimal threshold.

- The precision is a measure that evaluates how many of our alarm triggered was indeed correct: $precision = \frac{TP}{TP+FP}$

- The recall or the true positive rate is a measure that evaluates among all the failures, how many were well predicted: $TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$.

- The fall-out or the false positive rate is a measure that evaluates how many triggering errors we did, compared to the total of healthy runs: $FPR = \frac{FP}{N} = \frac{FP}{TN+FP}$. Compared to the previous metrics, we ideally want this value to be as low as possible.

- A usual metrics that data scientist use to compare models is the *F1_score* which is the harmonic average of precision and recall: $F1\_score = 2 * \frac{precision * TPR}{precision + TPR}$.

We will consider this latter metric as the decision measure for our best threshold and our best health indicator.

Finally, we build a business score that has an incentive to give subjective information about the quality over the timing of the prediction. As we said, predicting too soon is not enviable since it represents a shortfall of resources and money. On the other hand, predicting too late before the failure does allow not enough time for the maintenance team to physically replace the disk.
We build the business score according to the function Valentin Hamaide explained in his paper [11]. The ideal delay would be 7 days. Between 5 and 10 days, the score decreases slowly compared to 7 days. The score decreases more strongly between 10 and 15, as well as between 0 and 5. Finally, predicting a failure before 15 days would give a zero business score.

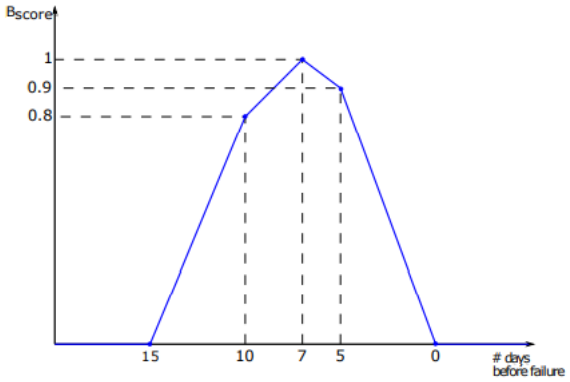


Figure 4.1: Business score function

The $Bscore$ as defined is useful only on true positive runs. To compute the business score of a predictor, we thus take the average $Bscore$ over all true positive runs. The business score will, in this study, be only informative in regards to the choice of the best predictor. It would only give information about the quality of the prediction timing of the final best predictor.

4.3 Adding probability

This section will explain the second principal goal of this thesis: adding to the classical predictor, an evaluation function based on probability. The idea is to stop considering the alarm as triggered or not, but instead consider that from a threshold, we have a probability of triggering this alarm. This change of evaluation is applicable on the second level of our prediction process.

In the deterministic case, we looked at all the possible values of the health indicator build in level 1. We compare each of these values to a threshold k and raise the alarm if the health indicator of the sample was below the threshold k. As soon as we had a value lower than k, we considered the run faulty and do not consider the next samples of the run. Let's recall that we agreed to consider the health indicator to be better when it has a bigger value and to be weaker when the value is lower.

In the probabilistic case, the critical moment for the alarm will not be the value k alone anymore but an area defined by this k and a width l. These two parameters defined two other values $k_1 = k - \frac{l}{2}$ and $k_2 = k + \frac{l}{2}$.

As you can see in figure 4.2, if the value of the health indicator is lower than k_1, the alarm is triggered. Between k_1 and k_2, the probability of sounding the alarm decreases linearly, whereas if our health indicator has a value greater than k_2, the sample is definitively considered healthy and there is zero probability of raising an alarm.

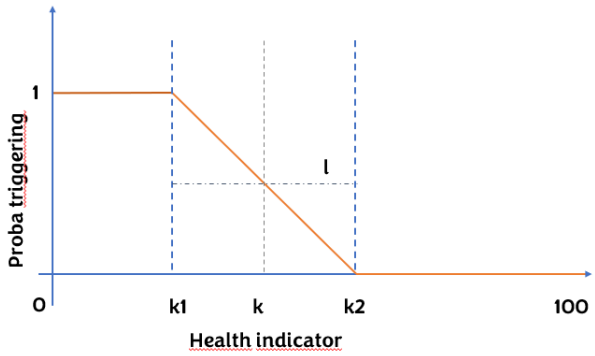


Figure 4.2: Probability thresholds

In order to apply these probabilities in our predictors, when we encounter a value lower than k_2 and higher than k_1, we launch a random phenomenon. This phenomenon uses the probability of the function in the graph above, for the health indicator value encountered.
Practically, to take the decision of triggering the alarm from a given probability, we take a random number between 0 and 1 and look if it is below the probability number. If the alarm has been fired by this "roll of the dice", we no longer consider the data of the current run and analyze the next run. If the alarm has not been fired, we look at the next value in the run which is smaller than k_2 and we restart a random phenomenon with the complementary probability of the previous phenomenon. This complementary probability will lead the decision of triggering to be more and more likely.
Consequently, if a health value corresponding to a small probability of raising the alarm appears many times in the run, there will be a high probability that the alarm will be raised at the end. It comes from the effect of always continuing the

level 2 process with the complementary probability.

Adding the probability measure should be efficient to detect a new sort of failure. It will probably consider repetitive small variations of health values as a deterioration behavior. On the other hand, it may tolerate an isolated low value of health indicator. For instance, let the optimized threshold be $k = 90$ and use a probability width of $l = 10$. If we have a recurrent health value of $hi = 94$, after 5 repetitions, for example, the run would have a probability of 0.40951 to be considered faulty while it would never be the case in deterministic.

$$\begin{aligned} p &= 0.1\left(1 + (1-0.1) + (1-0.1)^2 + (1-0.1)^3 + (1-0.1)^4\right) \\ &= 0.1 + 0.09 + 0.081 + 0.0729 + 0.06561 \\ &= 0.40951 \end{aligned}$$

Moreover, if we observe once a value of $hi = 89$ but values of 100 the days after, we will have a 0.6 probability to consider the run failing and 0.4 to consider the run as a false alarm with an isolated critic health value.

In the next sections of the thesis, when we refer to this probability measure, we always specify the probability range, $ProbaRange$, as the width l from figure 4.2. Indeed, with a threshold k and the $ProbaRange$, we can determine the probability function for the health indicators.

4.4 Tuning Hyperparameters

An important step that has to be done once we will get our first results, is tuning the machine learning hyperparameters. These hyperparameters are parameters which are used to control the learning process of the models, in the first level.

In order to proceed to this step, we will have to compare the metrics from each level 1 method, with a different set of hyperparameters.
For binary and multi-class classification, the hyperparameters we tune are the regularization parameter (C) which is set by default to one, the kernel type to be used in the algorithm (kernel) and the kernel coefficient (γ). For the unsupervised One-class model, we consider the hyperparameter (ν), which is a lower bound of the fraction of support vectors and an upper bound on the fraction of training errors.

Usually, to obtain our metrics we fit the machine learning method on the train set and then test our model with the test data. In this context, for each combination, we fit the model with the train set and then compare the *F1_score*

obtained from the validation set. This set provides an unbiased evaluation of the model while tuning our hyperparameters. Finally, we keep the test set for the final prediction we will make from our tuned model. This data split is explained more specifically in section 2.2.2.

Formulation	**Possible hyperparameters**	**Horizons**	**Tuned hyperparameters**
Binary Classification	C= [0.1,1, 10, 100] $\gamma = [1, 0.1, 0.01, 0.001]$ kernel: ['rbf', 'linear']	0-91	C= 0.1 , $\gamma = 0.01, kernel : rbf$
		0-15	C= 1 , $\gamma = 0.1, kernel : rbf$
		0-7	C= 0.1 , $\gamma = 1, kernel : rbf$
Multi-class Classification	C= [0.1,1, 10, 100] $\gamma = [1, 0.1, 0.01, 0.001]$ kernel: ['rbf', 'linear']	0-7-15	C= 1 , $\gamma = 0.1, kernel : rbf$
		0-15-30	C= 0.1 , $\gamma = 0.1, kernel : rbf$
		0-5-10-15-20-25	C= 1 , $\gamma = 0.1, kernel : rbf$
One-class SVM	$\nu = [0.5, 0.1, 0.01, 0.001]$ $\gamma = [1, 0.1, 0.01, 0.001]$ kernel: ['rbf', 'linear']		$\nu = 0.01$ $\gamma = 1$ kernel: rbf

Table 4.1: Tuned hyperparameters for each first level machine learning methods

All the possible hyperparameters and their tuned values for each category of machine learning models are available in the table 4.1. To train several times the models with different combinations of hyperparameters, the CPU of an "Asus ZenBook 14" takes approximately 20 hours to execute.

Chapter 5

Results

In this chapter, we will discuss the different results we get from each category of predictive models, which we described in chapter 4. For each category, we will determine if adding a probability measure during the second level would improve the *F1_score* or the business score.

We are first going to look at the predictive maintenance results for the univariate case, looking at a single feature. Then, in the multivariate case, we will analyze the results for binary classifications, multi-class classifications and one-class anomaly detection.

In the different tables which will include the main results of this study, each results number will be presented in the following format: (*F1_score*, Precision, TPR, FPR and *B_score*). The *F1_score* would be the main metric we will value to compare our different methods. By doing so, we rely our comparisons on a mix between the precision and the TPR. The FPR would be a good indicator of the false alarm we concede, while the business score would be a good indicator of the timing of our true predictions. If we have comparable *F1_score*, we will compare the business score in second place. All result tables are better displayed and available in the appendix.

Each method's results will be divided into three timing objectives. Indeed, considering a disk healthy when until $DayBefore$ days before failure, is a goal we decide before the prediction. Since this variable represents the time window in which we consider a disk faulty, a small value is much more restrictive for the predictor. Consequently, we expect to get worse results when using a smaller value of the $DayBefore$ parameter. The difference in scores between these three timing objectives doesn't matter because each $DayBefore$ is another prediction purpose and thus should be analyzed independently. As mentioned in 4.2, we will consider

the values: 7, 15, and 91 as the possible values of the parameter $DayBefore$.

5.1 Univariate

As explained in section 4.1.1, to capture the predictive power of single feature in this dataset, we will analyse 4 different features: *smart_5*, *smart_187*, *smart_188* and *smart_197*.
For each of these features, we look for the optimal threshold which would allow the predictor to maximize his *F1_score*.
These SMART features are normalized according to the manufacturers but the four features should lie between 1 and 100. As already explained, 100 represents a healthy feature while a smaller value should represent a degradation of the disk. When a sample in a run has a value below the optimal threshold, the alarm would be triggered.

Feature repartition

To determine the optimal threshold, we can observe in table 5.1 each value available in the train data set for a feature.

Attributes	**Possible values**
smart_5_normalized	100, 99, 98, 97, 96
smart_187_normalized	100, 99, 98, 97, 96, 95, 94, 93, 92, 90, 89, 88, 86, 84, 82, 72, 43
smart_188_normalized	100, 99, 98
smart_197_normalized	100, 98, 97, 96

Table 5.1: Available values for each smart feature in the train data set

For these four features, it would be interesting now to look at the occurrence of their values in healthy samples and failing samples. As already mentioned, a sample could be considered as failing if it is in the $DayBefore$ day window before the actual failure. There are three parameters $DayBefore$ considered in this paper.

In the figures 5.1, 5.3 and 5.4, we observe only a few possible integer values for the threshold. For feature 197, we observe that the values below 98 include 3 failing samples. We could imagine the threshold would be 98. For attribute 188, all failing data have a value of 100 and don't highlight a failing behavior. We suppose the predictor would have to choose for feature 5 whether to put the threshold at 99 or not. If it is the case, some predictions would lead to false alarms.

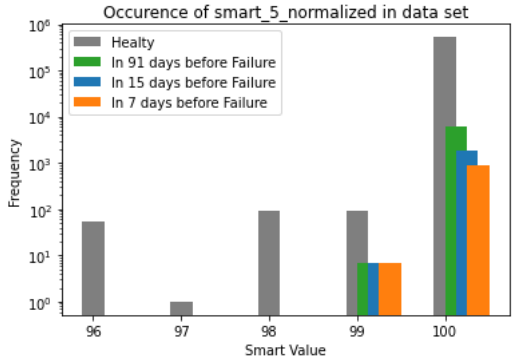


Figure 5.1: repartition of the train data samples of feature SMART 5

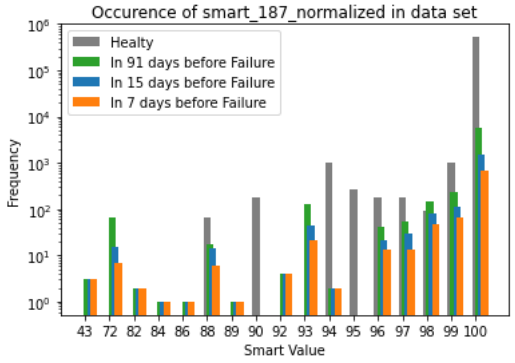


Figure 5.2: repartition of the train data samples of feature SMART 187

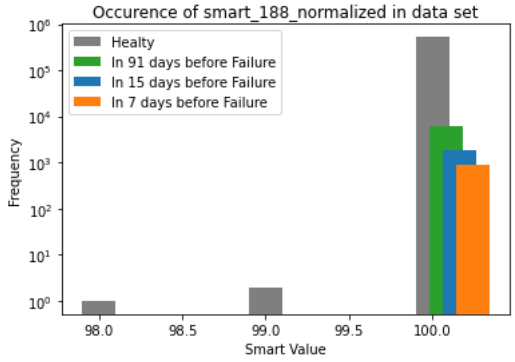


Figure 5.3: repartition of the train data samples of feature SMART 188

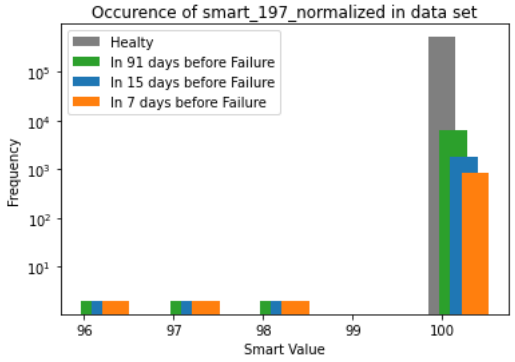


Figure 5.4: repartition of the train data samples of feature SMART 197

On the figure 5.2, we observe a bigger diversity in the values of the samples and it seems complicated from the figure, to imagine which threshold could be optimal. Notice the frequency axis is on a logarithmic scale.

Metrics

In the result tables below, we can observe metrics for each relevant feature, for each studied value of the $DayBefore$ parameter. A reminder of the metrics format is placed on the inferior left corner of each table. The best metrics for each $DayBefore$ parameter is written in red. For each feature, we observe how probability can contribute to the scores. We characterize this probability with a "$ProbaRange$" parameter which is the window width around the threshold k, where the probability of triggering the alarm is non-null nor full. If the $ProbaRange$ parameter is zero,

then it is the deterministic measure. These concepts are explained in section 4.3. Then, per $DayBefore$ parameter, there are still two columns representing whether or not we use a rolling average over the 7 last days of the health indicator value.

UNIVARIATE:		ProbaRange	DayBefore 91	
			Not Rolling average	Rolling average
	Smart05:	0	0.2222 / 1 / 0.125 / 0 /0	0.2222 / 1 / 0.125 / 0 /0
		5	0.2222 / 1 / 0.125 / 0 /0	0.2222 / 1 / 0.125 / 0 /0
	Smart187:	0	0.4444 / 0.5455 / 0.375 / 0.0069 / 0.045	0.25 / 0.375 / 0.1875 / 0.0069 / 0.06
		2	0.38 / 0.5 / 0.3125 / 0.0069 / 0.054	0.25 / 0.375 / 0.1875 / 0.0069 / 0.06
		5	0.32 / 0.4444 / 0.25 / 0.0069 / 0.0675	0.25 / 0.375 / 0.1875 / 0.0069 / 0.09
	Smart188:	0	0.0423 / 0.0217 / 1 / 1 /0.0291	0.0423 / 0.0217 / 1 / 1 /0.0291
*(F1/Prec/TPR/FPR/BScore)	Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0

Table 5.2: Metrics for univariate methods considering an horizon of 91 days

UNIVARIATE:		ProbaRange	DayBefore 15	
			Not Rolling average	Rolling average
	Smart05:	0	0.1111 / 0.5 / 0.0625 / 0.0014 /0.09	0.0606 / 1 / 0.0313 / 0 /0.36
		5	0.1111 / 0.5 / 0.0625 / 0.0014 /0.09	0.0571 / 0.3333 / 0.0313 / 0.0014 /0.8
	Smart187:	0	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.1395 / 0.2727 / 0.0938 / 0.0056 / 0.12
		2	0.1961 / 0.2632 / 0.1563 / 0.0097 / 0.11	0.1395 / 0.2727 / 0.0938 / 0.0056 / 0.17
		5	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.077	0.1 / 0.25 / 0.0625 / 0.0042 / 0.18
	Smart188:	0	0.0013 / 0.0007 / 0.03125 / 1 / 0.93	0.0013 / 0.0007 / 0.03125 / 1 / 0.93
*(F1/Prec/TPR/FPR/BScore)	Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0

Table 5.3: Metrics for univariate methods considering an horizon of 15 days

UNIVARIATE:		ProbaRange	DayBefore 7	
			Not Rolling average	Rolling average
	Smart05:	0	0.0606 / 1 / 0.0313 / 0 /0.95	0.0606 / 1 / 0.0313 / 0 /0.36
		5	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0013 / 0.0007 / 0.0313 / 1 / 1
	Smart187:	0	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.0930 / 0.1818 / 0.0625 / 0.0063 / 0.18
		2	0.2308 / 0.3 / 0.1875 / 0.0097 / 0.09	0.0930 / 0.1818 / 0.0625 / 0.0063 / 0.18
		5	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.1364 / 0.25 / 0.0938 / 0.0063 / 0.12
	Smart188:	0	0 / 0 / 0 / 1 / 0	0 / 0 / 0 / 1 / 0
*(F1/Prec/TPR/FPR/BScore)	Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0

Table 5.4: Metrics for univariate methods considering an horizon of 7 days

In univariate, there is no improvement when adding probability in the second level. The main reason seems to be that only a very few integer values are encountered thus the threshold can only be chosen among a few candidates. There is not enough place for improvement when changing the threshold. We tried a probability range of 5 and 2, but in the best scenario it only gives the same results as we obtained in the deterministic case.
Moreover, adding a rolling average in the second level will lead us to worse results. It could have been expected since rolling averages smooth the values and therefore hide abrupt changes of behavior of the disks.

For the univariate case, we can observe that only the Feature *smart_187* seems to have predictive power. Indeed it has a *F1_score* of 0.4444 when predicting

in 91 days, without probability. This score is already significant and shows the importance of this feature. In the paper [9], they mentioned a Google study where "They find that any change in metric 187 is highly predictive of failure: after their first scan error (i.e. when a positive value for 187 is observed for the first time), drives are 39 times more likely to fail within 60 days than drives with no such errors". Since in our prediction, the threshold is 99, we will also raise the trigger if the *smart_*187 changes from the initial value of 100. This emphasized the importance of this SMART feature. Although the *F1_score* is already promising, the business score is low and gives us an indication of bad timing for predictions in the last 15 days.
As expected, SMART attributes 188 and 197 get bad results for prediction while SMART 5 has a *F1_score* of 0.2222 in the best case. These three features alone would not be useful to predict failures in a hard drive disk population.

5.2 Multivariate

In this multivariate section, we use several features to create a unique health indicator which is going to be fed in the second level of our predictors. As explained in section 2.2.1, the features we will use are: *Smart*03/05/07/187/188/190/197.

As explained in our methodology, in the multivariate case, each health indicator fed in the second level, results from the decision function of different machine learning methods and represents a combination of the features. When testing the effect of applying different probability ranges in the second level, we will normalize the health indicator to ease the comparison between $ProbaRange$ parameters. Each health indicator is therefore included between 0 and 1.

In general for each of the three categories of multivariate methods, due to our unbalanced data set, the machine learning models will predict (quasi)only healthy class in level 1. Indeed, predicting zero failure should give an accuracy close to 97.8%, since the failure proportion is 2.2%. Therefore, the health indicator is built from the decision function. Since all predictions are healthy, the possible values for the threshold are furthermore positive and we can interpret a small decision function value as a point closer to being predicted in a failing class.

5.2.1 Binary

For this specific classification method, we consider only two possible classes where our data samples could be classified. One class is the healthy one and the other

is faulty within a certain time horizon. The three horizons studied are: 7, 15 and 91 days. Again, the horizons involved in level 1, are not to be confused with $DayBefore$ parameters which are applicable in level 2.

Feature occurrence

For the three horizons, we can observe different health indicator features repartitions. In the figures 5.5, 5.6 and 5.7, we can see the distribution for the three horizons with a $DayBefore$ of 91. For each figure, the state of the samples (healthy or in the 91 days before the failure) is shown depending on their health indicator. The repartitions for other $DayBefore$ are shown in the appendix.

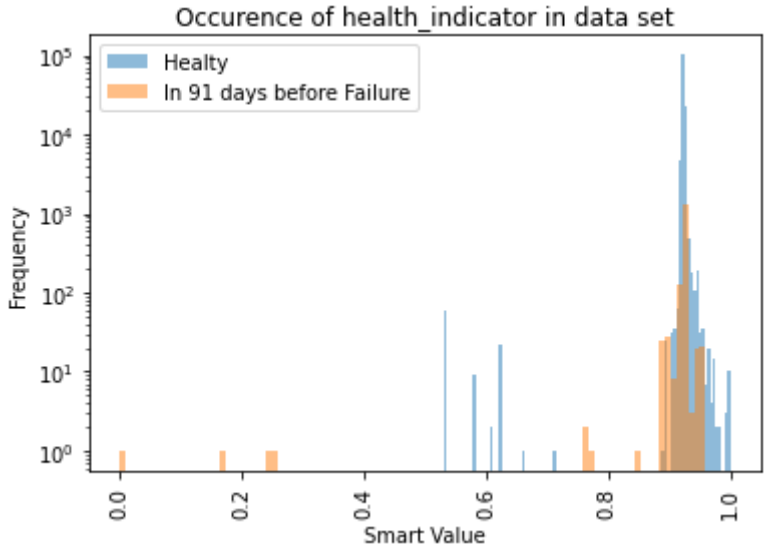


Figure 5.5: repartition of the train data samples of the health indicator created by a binary classification 0-91 with $DayBefore$ 91

From these histograms, we can observe that for neither of the two first binary classification, an obvious split could be made. Indeed, most of the failing samples seem to superpose the healthy values. For figure 5.7, we can imagine a separation around 0.91.
On each of these three figures, we observe a blue peak that contains approximately a hundred thousand healthy samples. When we compared with the orange biggest amount of failing samples which reach a thousand units, we can easily guess our threshold would be below these blue peaks.
Actually, we observe that the threshold for the method binary 0-91, binary 0-15 and binary 0-7 which give us the best results are respectively $k = 0.9158$, $k = 0.6641$

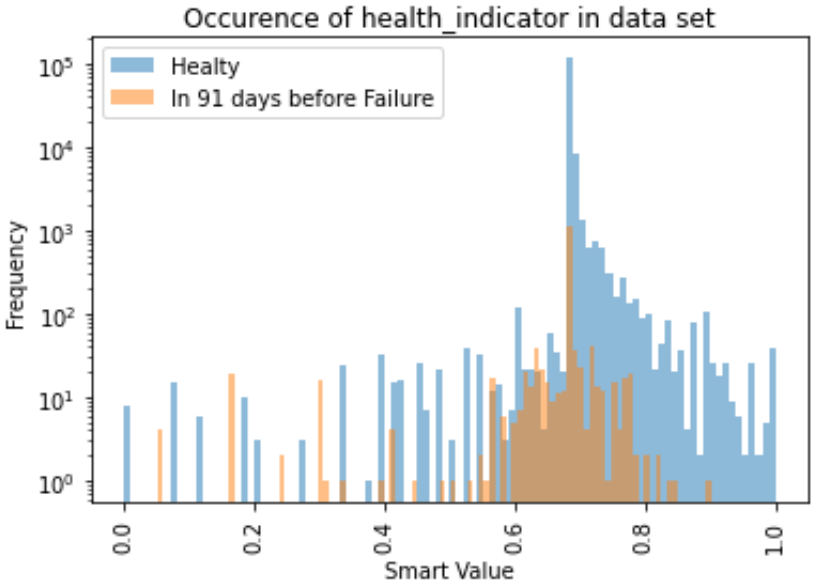


Figure 5.6: repartition of the train data samples of the health indicator created by a binary classification 0-15 with $DayBefore$ 91

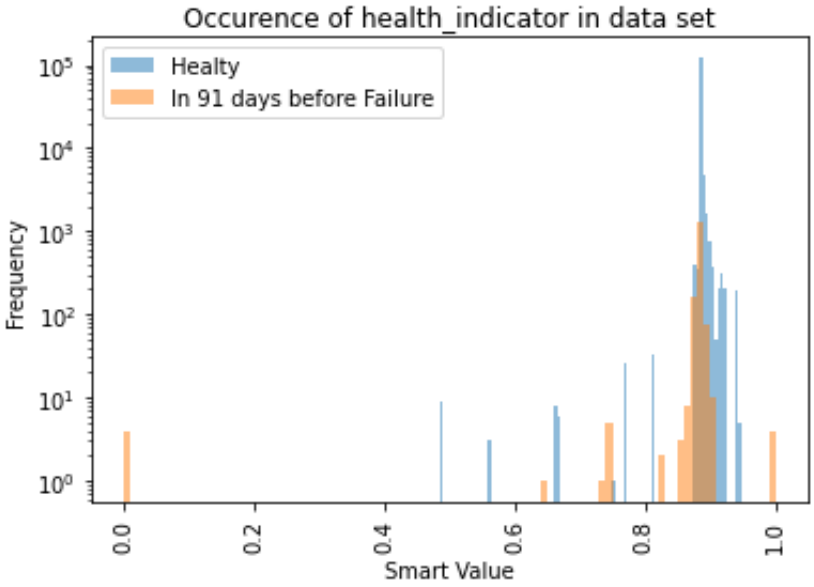


Figure 5.7: repartition of the train data samples of the health indicator created by a binary classification 0-7 with $DayBefore$ 91

and $k = 0.8849$. For each figure situation, it corresponds to a threshold just before the blue peaks.

Metrics

Once we observed the available value of the health indicators in the training runs, we can now look at the main metrics extracted from predictors on the testing runs. From the tables 5.5, 5.6 and 5.7, some main findings can be drawn.

		DayBefore 91	
MULTIVARIATE: BINARY CLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 91 days	0	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.005	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.01	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.14	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.05	0.44 / 0.6111 / 0.3438 / 0.0049 / 0.17	0.2857 / 0.6 / 0.1875 / 0.0028 / 0.25
	0.1	0.2857 / 0.6 / 0.1875 / 0.0028 / 0.25	0.25 / 0.625 / 0.1563 / 0.0021 / 0.27
Healty- failure in 15 days	0	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5333 / 0.5714 / 0.5 / 0.0083 / 0.10
	0.005	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.09
	0.01	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5333 / 0.5714 / 0.5 / 0.0083 / 0.08
	0.05	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.12	0.5424 / 0.5926 / 0.5 / 0.0076 / 0.07
	0.1	0.5667 / 0.6071 / 0.5313 / 0.0076 / 0.09	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.09
Healty- failure in 7 days	0	0.5517 / 0.6154 / 0.5 / 0.0069 / 0.16	
	0.005	0.5614 / 0.64 / 0.5 / 0.0063 / 0.14	0.3272 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.01	0.5313 / 0.5313 / 0.5313 / 0.0104 / 0.15	
	0.05	0.4828 / 0.5385 / 0.4375 / 0.0083 / 0.14	
*(F1/Prec/TPR/FPR/BScore)	0.1	0.25 / 0.625 / 0.1563 / 0.0021 / 0.23	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.13

Table 5.5: Metrics for binary methods considering an horizon of 91 days

		DayBefore 15	
MULTIVARIATE: BINARY CLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 91 days	0	0.3673 / 0.5294 / 0.2813 / 0.0056 / 0.18	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.26
	0.005	0.36 / 0.5 / 0.2813 / 0.0063 / 0.18	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.29
	0.01	0.3265 / 0.4706 / 0.25 / 0.0063 / 0.21	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.35
	0.05	0.3333 / 0.5 / 0.25 / 0.0056 / 0.19	0.1905 / 0.4 / 0.125 / 0.0042 / 0.37
	0.1	0.2105 / 0.6667 / 0.125 / 0.0014 / 0.32	0.1111 / 0.5 / 0.0625 / 0.0014 / 0.59
Healty- failure in 15 days	0	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.21
	0.005	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.18
	0.01	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.16
	0.05	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2414 / 0.2692 / 0.2188 / 0.0132 / 0.19
	0.1	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.15	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.18
Healty- failure in 7 days	0	0.3793 / 0.4231 / 0.3438 / 0.0104 / 0.23	
	0.005	0.4138 / 0.4615 / 0.375 / 0.0097 / 0.19	
	0.01	0.3125 / 0.3125 / 0.3125 / 0.0153 / 0.22	
	0.05	0.2807 / 0.32 / 0.25 / 0.0118 / 0.15	
*(F1/Prec/TPR/FPR/BScore)	0.1	0.2 / 0.5 / 0.125 / 0.0028 / 0.18	

Table 5.6: Metrics for binary methods considering an horizon of 15 days

In the first place, compared to the univariate case, improvements can be observed for the main metrics. When we look at the metrics of the best predictors for

		DayBefore 7	
MULTIVARIATE: BINARY CLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 91 days	0	0.3265 / 0.4706 / 0.25 / 0.0063 / 0.09	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.12
	0.005	0.32 / 0.4444 / 0.25 / 0.0069 / 0.07	0.2051 / 0.5714 / 0.125 / 0.0021 / 0.22
	0.01	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.11	0.2051 / 0.5714 / 0.125 / 0.0021 / 0.18
	0.05	0.3043 / 0.5 / 0.2188 / 0.0049 / 0.08	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.06
	0.1	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.12	0.15 / 0.375 / 0.0938 / 0.0035 / 0.12
Healty- failure in 15 days	0	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2333 / 0.25 / 0.2188 / 0.0146 / 0.10
	0.005	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2456 / 0.28 / 0.2188 / 0.0125 / 0.17
	0.01	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2666 / 0.2857 / 0.25 / 0.0139 / 0.17
	0.05	0.3333 / 0.4091 / 0.2813 / 0.0090 / 0.08	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.18
	0.1	0.3077 / 0.4 / 0.25 / 0.0083 / 0.26	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.15
Healty- failure in 7 days	0	0.3103 / 0.3462 / 0.2813 / 0.0118 / 0.08	
	0.005	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.17	
	0.01	0.2687 / 0.2571 / 0.2813 / 0.0181 / 0.17	
	0.05	0.2903 / 0.3 / 0.2813 / 0.0146 / 0.17	
*(F1/Prec/TPR/FPR/BScore)	0.1	0.2 / 0.5 / 0.125 / 0.0028 / 0.25	

Table 5.7: Metrics for binary methods considering an horizon of 7 days

different parameters in binary classification, we can conclude that for $DayBefore$ 91 and $DayBefore$ 7 cases, the best horizon is when we label all samples in the 15 days before failure as faulty. Whereas, for $DayBefore$ 15, surprisingly, the best horizon is the 7 days window.

As for the univariate case, using some rolling average for the threshold doesn't improve the *F1_score* and the more restrictive we get with $DayBefore$, the worse are the metrics.

If we add some probability in the second level, there is still, mainly, no improvement in the scores.

For the 91 $DayBefore$ in table 5.5, adding a probability range of 0.05 to the horizon 15 doesn't improve the *F1_score* but will improve the timing of the true prediction. Indeed, the Bscore increase of 2% for the same predictions, compared to a zero probability level 2. The *F1_score* we get for this classifier, 0.5763, would be the best score from the binary methods.
Although, for the 7 days horizon, scores are smaller than for 15 days horizon, adding a probability range of 0.05 allow an improvement of 0.97% in the *F1_score*. However, this cost 2% in the Bscore and would be less pertinent if we consider more importance on the timing than the veracity of the predictions.

In the table 5.6 of 15 $DayBefore$, we observe an improvement of 3.45% for the 7 days horizons, with the use of probability. For this specific case, improving *F1_score* decrease the Bscore by 4%. We can assume the new predictions are made at a worse timing than the existing prediction and it will lower the average Bscore.

Finally for 7 $DayBefore$ in table 5.7, there is improvement for the 15 days horizon. Indeed, the $F1\_score$ increase by 1.19% but the Bscore decrease by 8%.

ROC curves

To conclude the analysis of the improvement brought by the probability measure, it could be interesting to compare the ROC (receiver operating characteristic) curves of our best deterministic methods with the best probabilistic equivalent. "The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings"[17].

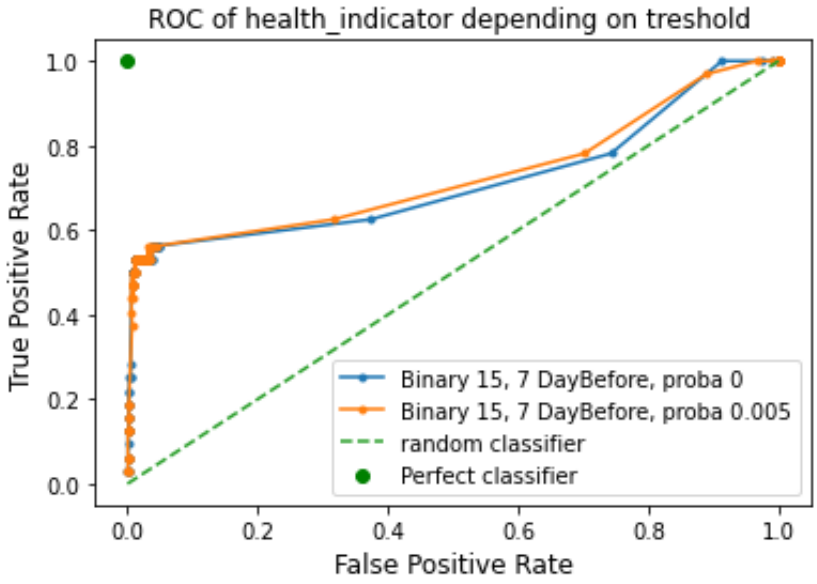


Figure 5.8: Comparison between deterministic and the best probabilistic ROC curves of binary classification 0-7 with $DayBefore$ 91

In the figures 5.8, 5.9 and 5.10, we compare the ROC curves of the best probabilistic improvement for each of the $DayBefore$ objective. For 91 and 15 $DayBefore$, it is the binary 0-7 method which is improved when adding a $ProbaRange$ of 0.005. For 7 $DayBefore$, it would be the binary 0-15 method which would be improved with a $ProbaRange$ of 0.05. On the figures, the improvement observed is however marginal.
Each point on the curves is the combination of TPR and FPR of a possible threshold.

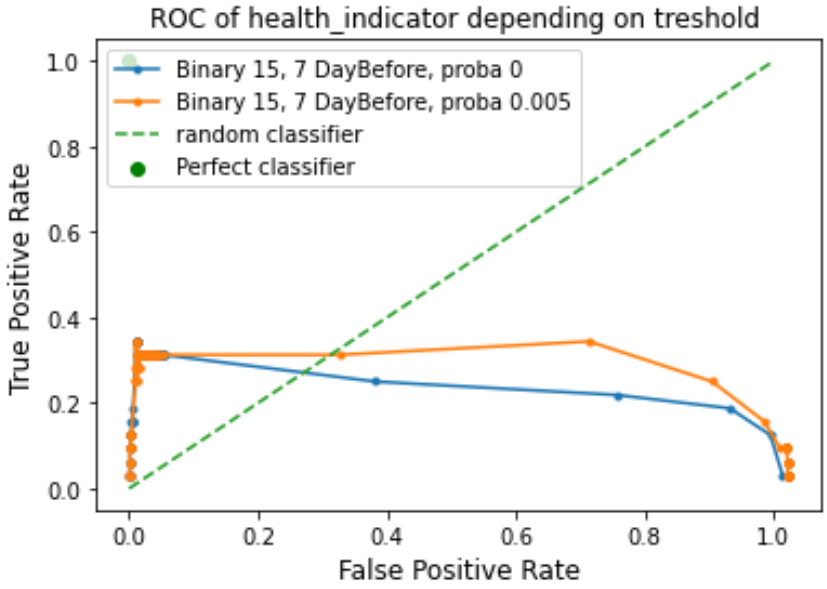


Figure 5.9: Comparison between deterministic and the best probabilistic ROC curves of binary classification 0-7 with $DayBefore$ 15

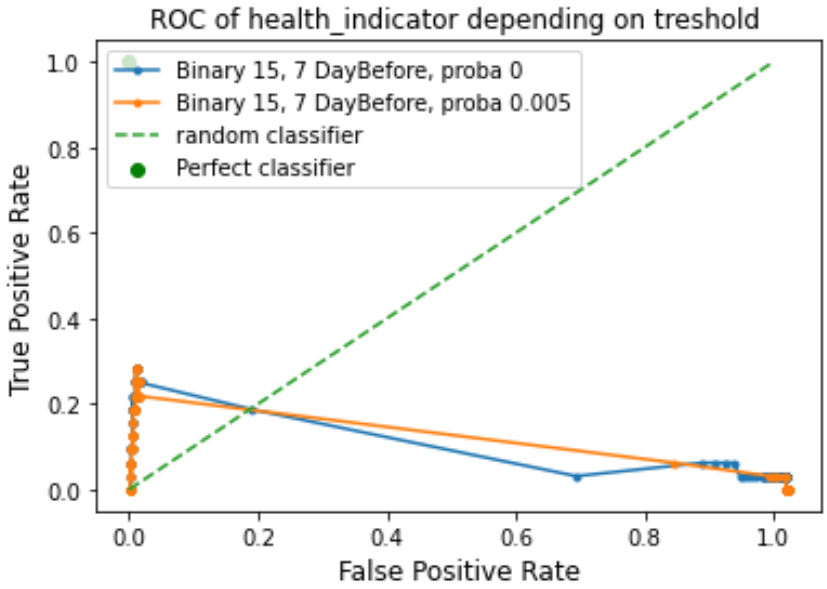


Figure 5.10: Comparison between deterministic and the best probabilistic ROC curves of binary classification 0-15 with $DayBefore$ 7

The green point upper-left represents the perfect predictor which gives all the true positive and no false alarms.

The first thing we observe on the figures 5.9 and 5.10 are the extreme points with a false positive rate close to one and a true positive rate near zero. Usually, these points are almost impossible to get since it means predicting failure for all the healthy data but not for the actual failing runs. This anomaly came from the parameter $DayBefore$. Indeed, these extreme points represent the scenario where we pull the trigger anyway for the biggest health indicator values. The alarm will be raised during the first days of the run and since these predictions are not in the 15 or 7 $DayBefore$, we consider it as a false positive. Every healthy run is predicted faulty while every failing run is predicted too soon.

Looking to the left side extreme points shows potential thresholds which should give the best metrics. Indeed, the binary (0-7) model for 91 $DayBefore$ has as best coordinates (TPR=0.5313 ; FPR=0.0069). The binary (0-7) model for 15 $DayBefore$ is placed on (0.375 ; 0.0097) while the binary (0-15) model for 7 $DayBefore$ has as best coordinates (0.2813 ; 0.0090).

The orange curves which represent the behavior of all possible thresholds when adding the probability measure, are slightly above the deterministic curves. The improvement is not obvious yet. Above all, the biggest improving points on these figures do not seem to be the points close to the best ratio TPR/FPR. We may think the probability improves better for insignificant thresholds.

5.2.2 Multi-class

In this section, we don't consider, as for binary classification, a run either healthy or faulty anymore. We consider several classes in which a run can be considered. For this study, we will explore two 3-class and one 6-class separation. As explained in section 4.1.2, we proceed with 3 different labelings 0/7/15, 0/15/30 and 0/5/10/15/20/25 for the 6 classes division.

Feature occurrence

For the three labelling separations, we can observe different health indicator feature repartitions. In the figures 8.16, 8.17 and 8.18, we can see the distribution for the three different separations with a $DayBefore$ of 91. Notice, the two categories of features, shown in the figures, are the true state of samples as output of the second level. It is not the labeling done by the classification in the first level.

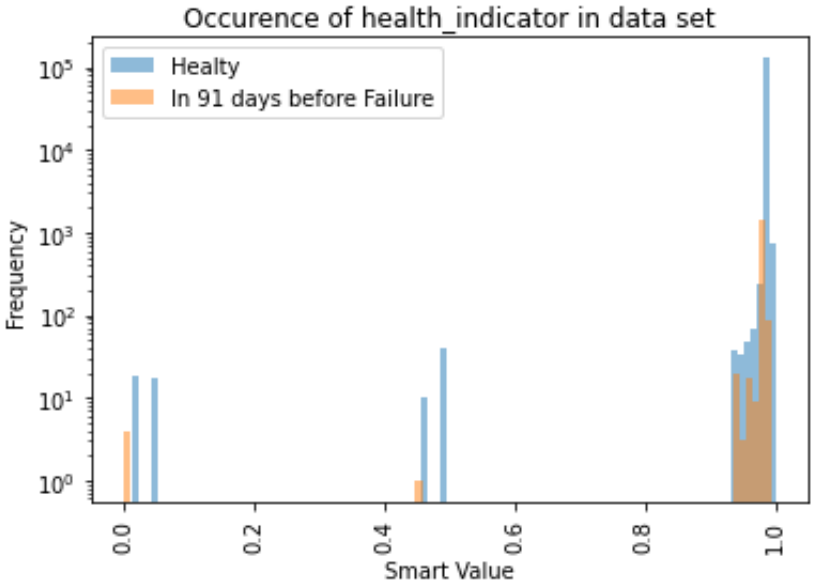


Figure 5.11: repartition of the train data samples of the health indicator created by a multi classification 0/7/15 with $DayBefore$ 91

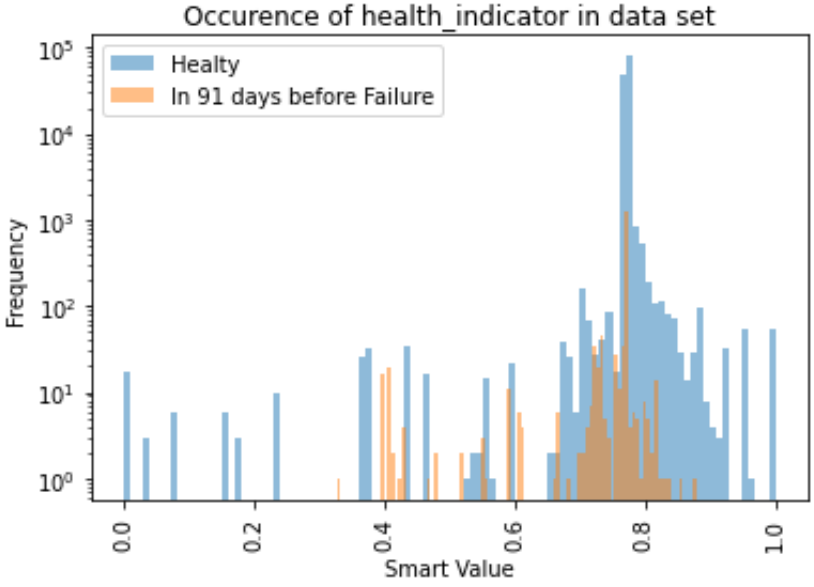


Figure 5.12: repartition of the train data samples of the health indicator created by a multi classification 0/15/30 with $DayBefore$ 91

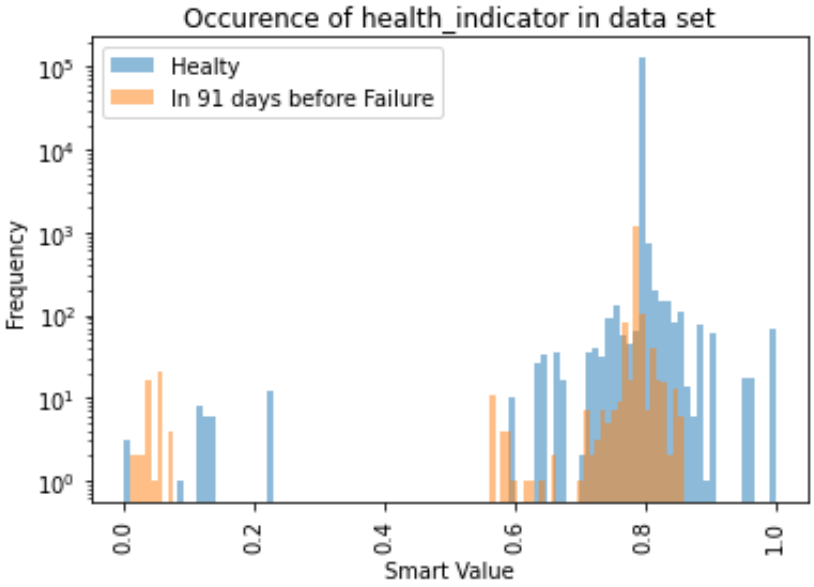


Figure 5.13: repartition of the train data samples of the health indicator created by a multi classification 0/5/10/15/20/25 with $DayBefore$ 91

The repartitions for other $DayBefore$ are shown in the appendix.
As for the binary repartition, we cannot conclude any good obvious threshold on the health indicator from these figures. The main part of failing samples lies over the healthy samples.

Again, since the scale of the frequency of the available values is logarithmic, we imagine the threshold would never exceed the blue peaks visible on each figure for each labeling separation. If it does, the false alarm rate would be excessively high compared to the true positive prediction we could gain.

Consequently, we observe that the threshold for the method multi-class 0/7/15, multi-class 0/15/30 and multi-class 0/5/10/15/20/25 which give us the best results are respectively $k = 0.9821$, $k = 0.761$ and $k = 0.7825$. As for binary methods, for each figure situation, it corresponds to a threshold just before the blue peaks.

Metrics

After visualizing the available health indicator value created from the multi-class machine learning methods, we can analyze the metrics which can be computed during the second level of the predictors.

As for the other results, adding a rolling average never improves the scores and a bigger $DayBefore$ imply better metrics.

MULTIVARIATE:		**DayBefore 91**	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 7 an 15 days	0	0.5454/ 0.6522 / 0.4688 / 0.0056 / 0.16	0.5091 / 0.6087 / 0.4375 / 0.0063 / 0.13
	0.005	0.5185 / 0.6364 / 0.4375 / 0.0056 / 0.14	0.4528 / 0.5714 / 0.375 / 0.0063 / 0.12
	0.01	0.5185 / 0.6364 / 0.4375 / 0.0056 / 0.1§	0.4528 / 0.5714 / 0.375 / 0.0063 / 0.14
	0.05	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.08	0.3922 / 0.5263 / 0.3125 / 0.0063 / 0.13
	0.1	0.1111 / 0.5 / 0.0625 / 0.0014 / 0	0.0426 / 0.0218 / 1 / 0.9979 / 0.06
Healty- failure in 15 an 30 days	0	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.12	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.005	0.5161 / 0.5333 / 0.5 / 0.0097/ 0.13	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.01	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.11	0.5161 / 0.5333 / 0.5 / 0.0097/ 0.08
	0.05	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.14	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.1	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.14	0.5 / 0.5357 / 0.4688 / 0.0090 / 0.06
6 Classes between 0 and 25	0	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.09
	0.005	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5667 / 0.6071 / 0.5313 / 0.0076 / 0.08
	0.01	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5574 / 0.5862 / 0.5313 / 0.0083 / 0.08
	0.05	0.5862 / 0.6538 / 0.5313 / 0.0063 / 0.08	0.5091 / 0.6087 / 0.4375 / 0.0063 / 0.07
*(F1/Prec/TPR/FPR/BScore)	0.1	0.5965 / 0.68 / 0.5313 / 0.0056 / 0.09	0.5357 / 0.625 / 0.4688 / 0.0063 / 0.08

Table 5.8: Metrics for multi-class methods considering an horizon of 91 days

MULTIVARIATE:		**DayBefore 15**	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 7 an 15 days	0	0.4074 / 0.5 / 0.3438 / 0.0076 / 0.20	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.20
	0.005	0.3774 / 0.4762 / 0.3125 / 0.0076 / 0.20	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.16
	0.01	0.3774 / 0.4762 / 0.3125 / 0.0076 / 0.17	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.19
	0.05	0.3571 / 0.4167 / 0.3125 / 0.0097 / 0.23	0.2745 / 0.3684 / 0.2188 / 0.0083 / 0.17
	0.1	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0040 / 0.0020 / 0.0938 / 1 / 0.65
Healty- failure in 15 an 30 days	0	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.20	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.18
	0.005	0.3226 / 0.3333 / 0.3125 / 0.0139 / 0.22	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.18
	0.01	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.22	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.16
	0.05	0.3509 / 0.4 / 0.3125 / 0.0104 / 0.18	0.2545 / 0.3043 / 0.2188 / 0.0111 / 0.16
	0.1	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.18	0.2581 / 0.2667 / 0.25 / 0.0153 / 0.16
6 Classes between 0 and 25	0	0.3860 / 0.44 / 0.3438 / 0.0097 / 0.16	0.3051 / 0.3333 / 0.2813 / 0.0125 / 0.16
	0.005	0.3793 / 0.4231 / 0.3438 / 0.0104 / 0.18	0.3051 / 0.3333 / 0.2813 / 0.0125 / 0.14
	0.01	0.3729 / 0.4074 / 0.3438 / 0.0111 / 0.18	0.2950 / 0.3103 / 0.2813 / 0.0139 / 0.17
	0.05	0.3729 / 0.4074 / 0.3438 / 0.0111 / 0.16	0.2456 / 0.28 / 0.2188 / 0.0125 / 0.18
*(F1/Prec/TPR/FPR/BScore)	0.1	0.3929 / 0.4583 / 0.3438 / 0.009 / 0.14	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.19

Table 5.9: Metrics for multi-class methods considering an horizon of 15 days

MULTIVARIATE:		DayBefore 7	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 7 an 15 days	0	0.3390 / 0.3704 / 0.3125 / 0.0118 / 0.14	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.11
	0.005	0.3396 / 0.4286 / 0.2813 / 0.00083 / 0.21	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.25
	0.01	0.3509 / 0.4 / 0.3125 / 0.0104 / 0.21	0.2963 / 0.3636 / 0.25 / 0.0097 / 0.22
	0.05	0.2400 / 0.3333 / 0.1875 / 0.0083 / 0.06	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.22
	0.1	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0013 / 0.0007 / 0.0313 / 1 / 0.95
Healty- failure in 15 an 30 days	0	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.07	0.2623 / 0.2759 / 0.25 / 0.0146 / 0.05
	0.005	0.3019 / 0.3810 / 0.25 / 0.009 / 0.19	0.2759 / 0.3077 / 0.25 / 0.0125 / 0.15
	0.01	0.3019 / 0.3810 / 0.25 / 0.009 / 0.15	0.2581 / 0.2667 / 0.25 / 0.0153 / 0.17
	0.05	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.32	0.25 / 0.2917 / 0.2188 / 0.0118 / 0.16
	0.1	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.15	0.2545 / 0.3043 / 0.2186 / 0.0111 / 0.16
6 Classes between 0 and 25	0	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.08	0.2712 / 0.2963 / 0.25 / 0.0132 / 0.07
	0.005	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.08	0.2712 / 0.2963 / 0.25 / 0.0132 / 0.05
	0.01	0.3103 / 0.3461 / 0.2813 / 0.0118 / 0.08	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.15
	0.05	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.17	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.11
*(F1/Prec/TPR/FPR/BScore)	0.1	0.2857 / 0.3333 / 0.25 / 0.0111 / 0.05	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.20

Table 5.10: Metrics for multi-class methods considering an horizon of 7 days

When we look at tables 5.8, 5.9 and 5.10, we are able to observe the best combination of parameters for each $DayBefore$ goals. For 91 $DayBefore$, the best *F1_score* (0.5965) is obtained with the 6 classes method for a probability range of 0.1 without rolling average. When we consider a prediction before 15 days as missed, even if the 6 classes method is close, the 0/7/15 separation methods without probability measure nor rolling average is the best combination and obtain a *F1_score* of 0.4074. Finally, we use the same labelling separation for 7 $DayBefore$ but adding a probability range of 0.01 to obtain 0.3509 as *F1_score* .

In the 91 $DayBefore$ table 5.8, probability doesn't improve the 0/7/15 method but does improve the two other separations. Indeed, adding a $ProbaRange$ of 0.05 for the 3-class 0/15/30 improves the business score and hence the timing of prediction by 2%. For the 6-class level 1, we improve the *F1_score* by 2.02% and concede 1% in Bscore when we add 0.1 $ProbaRange$.

For 15 $DayBefore$, the same methods show improvement with a probability measure, as we can see in table 5.9. For the method 0/15/30, the probability range 0.05 improve by 2.3% the *F1_score* by decreasing the FPR but decrease also the Bscore by 2%. For the 6-class ML method, the *F1_score* is improved by 0.69% with an addition of probability (range 0.1). However, the best results are from the 0/7/15 and do not improve with any $ProbaRange$.

Finally, in the last table 5.10, we observe probability improvement for each multi-class method. A $ProbaRange$ of 0.01 improve both *F1_score* and Bscore by respectively 1.19 and 7%, with the smallest 3 class separation. For the other 3-class method, adding a probability measure of $ProbaRange = 0.05$ will improve

by 2.49 and 7% both scores while it only improves the Bscore by 9% for the 6-class level 1 method.

The results are better for multi-class machine learning methods than for binary separation. Moreover, in this section, results show bigger improvements when we evaluate the *F1_score* with probability in the second level. On the other hand, using a rolling average is apparently never beneficial.

ROC curves

After analyzing the metrics, it would be interesting to visualize these probability improvements with the respective ROC curves for each $DayBefore$.
For 91 $DayBefore$, we plot the comparison of the 6-class method for $ProbaRange$ 0 and 0.1. Then for 15 $DayBefore$, the 3-class 0/7/15 method gives the best scores. However, we plot the comparison between $ProbaRange$ 0 and 0.05 for 0/15/30 method because it is the biggest probability improvement. Finally, for 7 $DayBefore$, we plot the comparison of the multi-class 0/15/30 method for $ProbaRange$ 0 and 0.05 too.

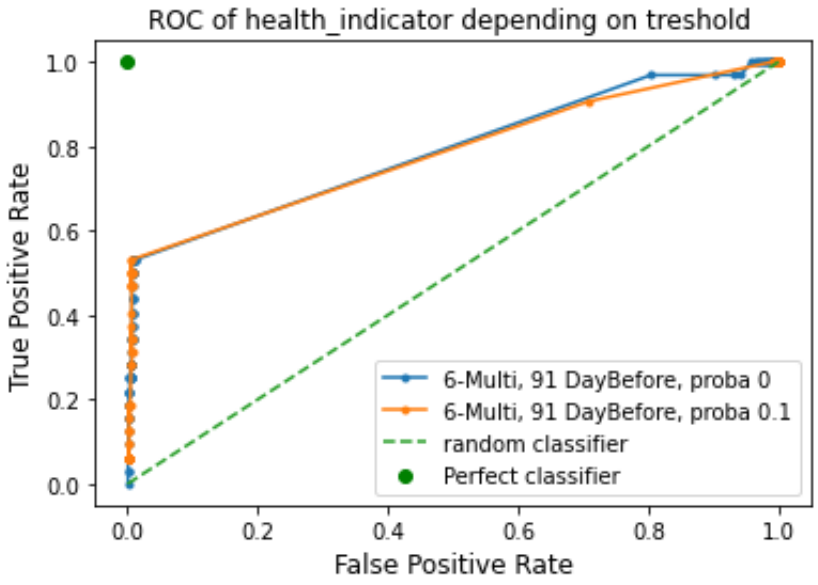


Figure 5.14: Comparison between deterministic and the best probabilistic ROC curves of multi classification 0/5/10/15/20/25 with $DayBefore$ 91

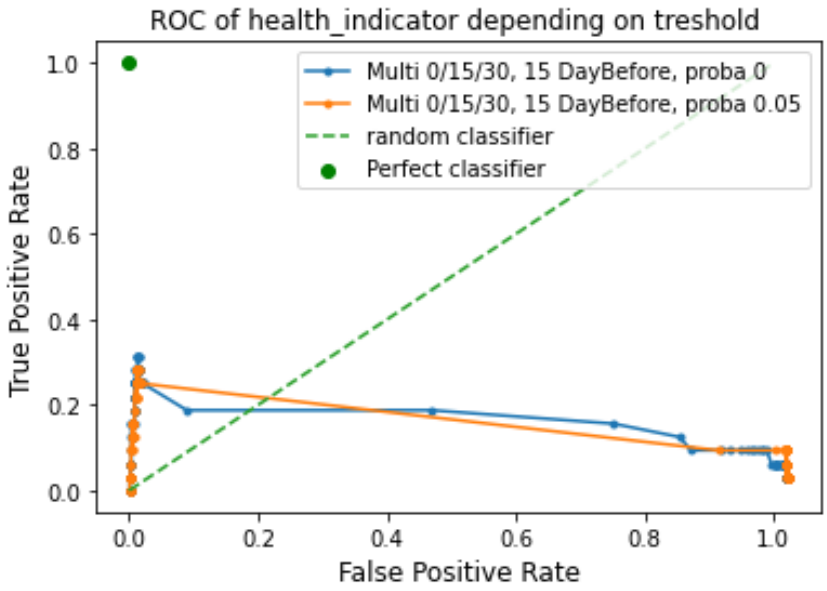


Figure 5.15: Comparison between deterministic and the best probabilistic ROC curves of of multi classification 0/15/30 with $DayBefore$ 15

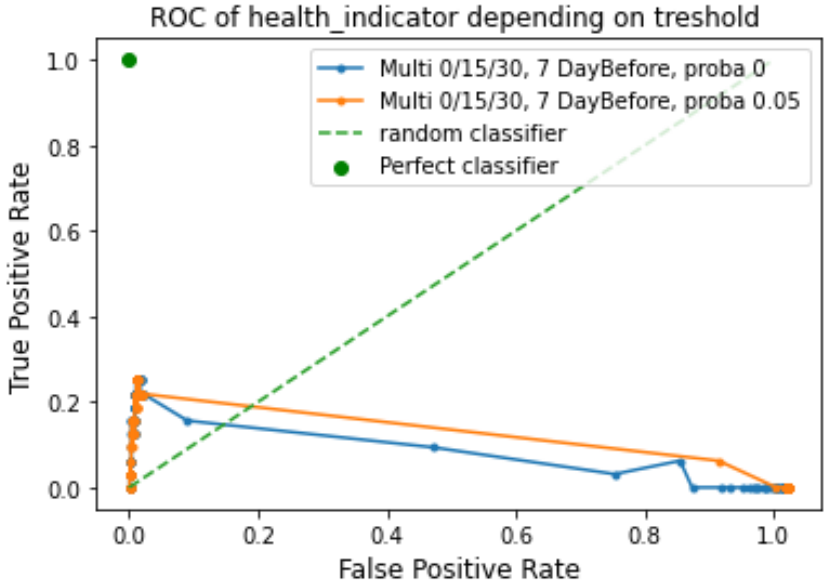


Figure 5.16: Comparison between deterministic and the best probabilistic ROC curves of of multi classification 0/15/30 with $DayBefore$ 7

As we can see on the figures 5.14 and 5.16, the probabilistic curve is barely improved and particularly in 5.15, the best threshold point seems to be the deterministic one. This is surprising since the *F1_score* is higher for the probabilistic measure. We suppose this comes from the fact that the *F1_score* is a harmonic average between precision and TPR while the figures show a combination of TPR and FPR.

5.2.3 Unsupervised

In this section, we present the results for our unsupervised level 1 method. Indeed, we build the health indicator with a one-class SVM machine learning method, on healthy training runs. This method aims to detect anomalies from a healthy set and for our data, it predicts approximately half of the data set as an anomaly. This seems like bad prediction results at first sight, but in level 1, we do not keep the predictions themselves but build the health indicator based on the decision function.

Feature occurrence

From these decision function values, we can plot their repartition figures as for the other methods. For this category, we plot the same level 1 health indicator for the three $DayBefore$ restrictions.

From the histograms, we can see the failing samples decline especially between 0.3 and 0.4 where some values drop from approximately 90 to a dozen of samples. Moreover, the extreme right failing samples drop from one thousand to a hundred with an increasing restriction of $DayBefore$. Compared with other methods' repartitions, the health indicator seems more scattered. However, in general, the failing samples seem to be more condensed in the left part of the figures and separation seems to be more straightforward.

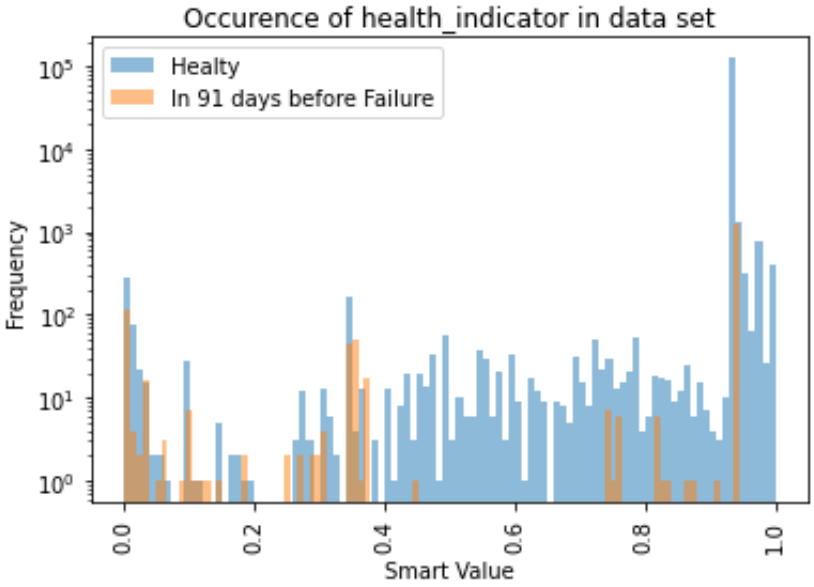


Figure 5.17: repartition of the train data samples of the health indicator created by One-class anomaly detection with $DayBefore$ 91

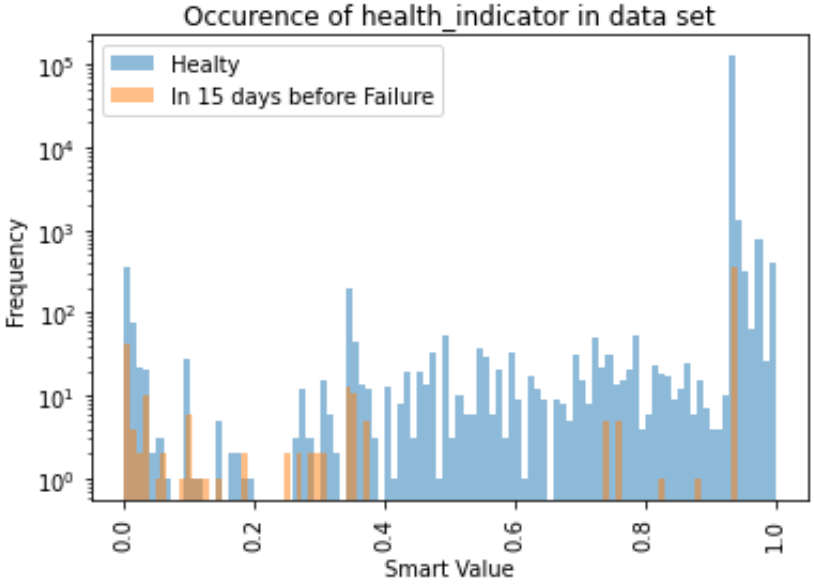


Figure 5.18: repartition of the train data samples of the health indicator created by One-class anomaly detection with $DayBefore$ 15

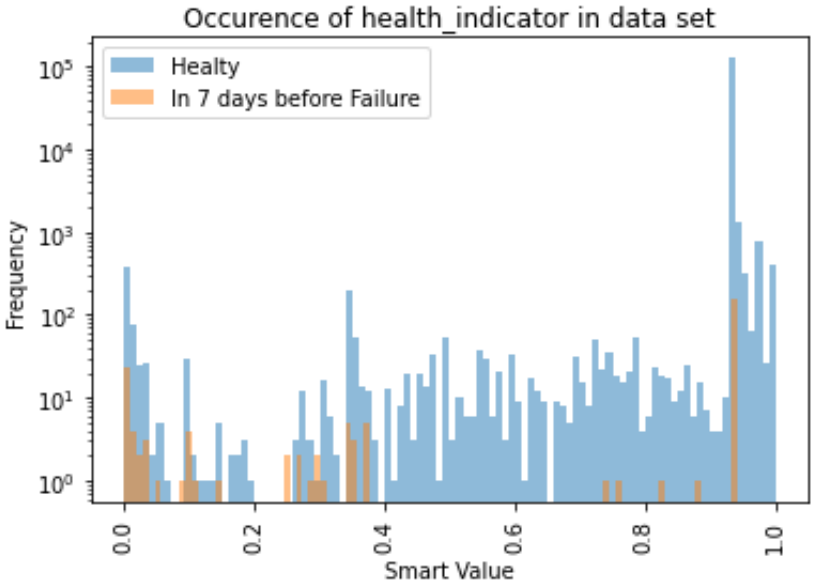


Figure 5.19: repartition of the train data samples of the health indicator created by One-class anomaly detection with $DayBefore$ 7

Metrics

When we look at the metrics of this method, the results do not match what we could expect from our first sight of repartition figures.
Indeed, the best *F1_score* results obtained for this method is 0.4706 and is lower than what we got from the 6-class probabilistic classification (0.5965).

MULTIVARIATE:		**DayBefore 91**	
UNSUPERVISED:	**ProbaRange**	Not Rolling average	Rolling average
OneClass SVM	0	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.005	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.15	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.01	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3226 / 0.3333 / 0.3125 / 0.0139 / 0.13
	0.05	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.15	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.1	0.4615 / 0.6 / 0.375 / 0.0055 / 0.20	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.13
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.22	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.005	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.30	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.01	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.22	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.05	0.36 / 0.5 / 0.2813 / 0.0063 / 0.18	0.1558 / 0.1333 / 0.1875 / 0.0271 / 0.27
	0.1	0.3396 / 0.4286 / 0.2813 / 0.0083 / 0.30	0.1569 / 0.2105 / 0.125 / 0.0104 / 0.55
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.14	0.1143 / 0.0822 / 0.1875 / 0.0465 / 0.12
	0.005	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.11	0.1143 / 0.0822 / 0.1875 / 0.0465 / 0.12
	0.01	0.2813/ 0.2813 / 0.2813 / 0.0160 / 0.37	0.1132 / 0.0811 / 0.1875 / 0.0472 / 0.12
	0.05	0.2857 / 0.4118 / 0.2188 / 0.0069 / 0.13	0.1558 / 0.1333 / 0.1875 / 0.0271 / 0.22
*(F1/Prec/TPR/FPR/BScore)	0.1	0.28/ 0.3889 / 0.2188 / 0.0076 / 0.10	0.1351 / 0.1190 / 0.1563 / 0.0257 / 0.38

Table 5.11: Metrics for one-class methods considering an horizon of 91 days

As for all other results, adding a rolling average over 7 days in the second level of the predictor does not improve any metrics.
For the 91 and 7 $DayBefore$ objectives, the probability measure does not improve the deterministic measure of *F1_score*. Nevertheless, for the objective of 15 days, adding a $ProbaRange$ of 0.05 improve by 0.71% the *F1_score* while dropping the Bscore by 4%. On the other hand, adding 0.005 of probability in the second level keep the same *F1_score* but improve the timing by increasing the Bscore by 8%.

5.3 General interpretation of results

After analyzing the metrics of each category of the first level method, we can highlight several prediction trends as a conclusion:

First of all, as expected, univariate methods are the less relevant predictors.
For the binary category, the improvement of the probabilistic measure is limited. Only the binary method with horizon 7 for $DayBefore$ 91 and 15, as well as horizon 15 for $DayBefore$ 91 and 7, improves with a probabilistic range. The best improvement is, however, relatively consequent (+3.45%).
For the multi-class methods with labeling 0-15-30 and 0-5-10-15-20-25, the probability always gives better results. For 0-7-15, we observe improvement, only in $DayBefore$ 7. Furthermore, for the multi-class category, the best increase is +2.3%.
Then, for OneClass SVM, there is only amelioration for $DayBefore$ 15.

Furthermore, if we look at each of the $DayBefore$ objectives, we can conclude several observations about the best metrics obtained among our tests.
Firstly, for 91 $DayBefore$, the best *F1_score* is 0.5965 obtained with a probabilistic range of 0.1. For this timing goal, the best methods are the 6-class classification, followed by binary 0-15 and binary 0-7.
Secondly, for 15 $DayBefore$, the best *F1_score* is 0.4138 with a $ProbaRange = 0.005$. For this $DayBefore$, the best methods are the binary 0-7 classification, following by binary 0-15 and the 6-class together.
Finally, for 7 $DayBefore$, the best *F1_score* is 0.3509 with a $ProbaRange = 0.01$. For this timing goal, the best methods are the multi-class 0-7-15 classification, following by binary 0-91 and binary 0-15.

In regards to the main goal of this thesis, we can conclude that the probabilistic measure brought in the second level of prediction, improves our best results for each of the timing objectives we consider. In fact, it improve the best *F1_score* for 91, 15 and 7 $DayBefore$ by 2.02, 3.45 and 1.19 respectively.

Comparison with literature

When we compare our results and main conclusion to the state of the art, presented in section 3, we can highlight some conclusions.
The results for both the article of Valentin Hamaide [11] and this thesis, show better scores for the supervised method than anomaly detection. Compared to their study, for this database, multi-class methods perform better than binary classification. Unlike their conclusion, our univariate predictor is not a good predictor compared to other methods.
In article [7], they added historical context but stated no significant improvement of the prediction. An equivalence can be made with the use of a rolling average introduced in our level 2. This change of data never improves the metrics of our different methods.
From the document [9], we highlight a binary 0-30 classification which used Optimal classification tree method and got a equivalently computed *F1_score* of 0.4896. Our binary metrics for 91 $DayBefore$, are significantly above when we label the sample for smaller, 15 and 7 days horizons. Indeed, the *F1_score* are respectively 0.5763 and 0.5614. When we build the health indicator with a larger horizon, the *F1_score* is only a bit lower: 0.4706. By projection, we can assume our two-level predictor gets better results than their OCT method.
In the article [10], we can compute a *F1_score* of 0.5475 for their decision tree ML method, based on their confusion matrix. It is beneath our best methods which got a *F1_score* of 0.5965 but the timing seems better in their case since 72% of their predictions are made within the 30 days before failure.
In paper [13], when they use only SMART features, their best *F1_score* is 0.58 for binary classification with a 10 days horizon window. This study was made over a very large population of disks and their results surpass our best predictor in the timing of the prediction since our *F1_score* of 0.5965 is obtained for 91 $Daybefore$.

Limitations

From the promising results we got in this chapter, it would be interesting to keep some reserves vis-à-vis some limitations of our problem.

First of all, as implemented and explained in section 4.3, when we add the probability to the second level of prediction, we proceed to some probabilistic phenomena. These parts of our predictions add a certain uncertainty to the timing of the alarms raise and could slightly alter the scores from one code execution to another. However, we run our programs several times and the differences are not significant. The variations observed never exceed 0.0007.

Another limitation comes from the choices we made in section 2.2. Indeed, we discarded all the failures in the first 15 days of registration in the training data. These failures could have shown some information patterns missing in our training. Moreover, to gain computational time, we deleted a main part of the healthy runs. These choices were made to allow the predictor to focus on failing data, but it may lead to a situation that would be non-representative enough. Besides, if a failure occurs in the next days after the 91 days collected in a run, the run would be considered healthy for our training while it should have been faulty.

Finally, some parameters such as $DayBefore$, the labeling classification or $ProbaRange$ were chosen arbitrarily, based on our proper business and logistic incentives, explained in chapter 4. It is important to notice that using other parameters could have lead to better results. We, however, had choices to make and could not realistically test every possibility for each parameter.

Chapter 6

Future work

For the future possible ideas that could be considered to improve our study, we propose to consider more than a unique disk model, to use a bigger dataset, to study the disks on longer runs or to improve the feature selection using more complex selection methods.

In this chapter, a possible way of improving the probability measure effects on the predictions is presented. In this thesis, the probability measure has an impact on the decision-making process of the predictive maintenance predictor. We believed this impact can be improved by incorporating the probability in the training of the machine learning model.

The benefits of this approach should be that the probability range would be considered variable and therefore be optimized during the training of the model. Besides, the threshold would be directly optimized in the first level with the weight coefficients which emphasize the importance of each feature.

New model

By facility for the modelization of this newly defined problem, let's define our goal as the maximization of

$$F_{score} = \frac{(1 - FPR) * TPR}{(1 - FPR) + TPR}$$

It is a different metric from the *F1_score* used in this thesis, but it brings the quality of predictions out, as well. What will differ from the previous section, is the false and true positive rates. In section 4, the FPR is defined as the amount of false-positive runs (healthy run predicted failing), over all the healthy runs. When

we incorporate probability, the amount of false-positive runs are replaced by the expected value of predicting a healthy run failed. The same change applies to the TPR and is present in the optimization model explained below.

The modelization of this problem is the following:

$$\max \ F_{score} \tag{6.1}$$

$$\max \ \frac{(1 - FPR) * TPR}{(1 - FPR) + TPR} \tag{6.2}$$

$$\max \ \frac{(1 - \frac{1}{|N|} * \sum_{r \in N} \sum_{t \in T} c^{(r)}(t) * 1) * (\frac{1}{|P|} * \sum_{r \in P} \sum_{t \in T} c^{(r)}(t) * 1)}{(1 - \frac{1}{|N|} * \sum_{r \in N} \sum_{t \in T} c^{(r)}(t) * 1) + (\frac{1}{|P|} * \sum_{r \in P} \sum_{t \in T} c^{(r)}(t) * 1)} \tag{6.3}$$

where

$$c^{(r)}(t) = p^{(r)}(t) * \prod_{u<t} (1 - p^{(r)}(u)) \tag{6.4}$$

$$p^{(r)}(t) = -\frac{Hi^{(r)}(t)}{l} + \frac{1}{2} + \frac{k}{l} \tag{6.5}$$

$$Hi^{(r)}(t) = \sum_{f \in F} w_f * sample(r, t, f) \tag{6.6}$$

The variables of this optimization problem are:

- w_f the coefficients (weight) indicating the relative effect of a feature f on the outcome.
- k the threshold explained in section 4.3
- l the probability range explained in section 4.3

In the model equations,

- P is the set of positive runs (5886 runs in our case)
- N is the set of negative runs (128 runs)
- T is the set of days in a run (91 days)
- F is the set of features to train (7 SMART features)
- $sample(r, t, f)$ is the data of a feature f, for a day t, in a run r

In equation 6.6, we define $Hi^{(r)}(t)$ which is the health indicator of a sample in a day t of a run r. This indicator is formed by the sum of each weighted feature of a sample. These weights are the coefficient of a linear classifier and control how strongly the features would influence the output. They are variables that have to be optimized. Some bias coefficient b could be added to ensure that the result is not too big or too small on average. We decide to keep it simple and not to add it to the model. Different machine learning models have different ways of learning patterns from features. For this model, we use the linear kernel but other ways could have been exploited [14].

The equation 6.5 is a function that takes the health indicator as input and returns the probability of triggering an alarm for the sample with a threshold k and a probability range l. This equation is the function of the figure 4.2, in section 4.3.

The equation 6.4 allows to take the cumulative probability over a run, until day t. Let's say there is a 0.4 probability of triggering the alarm on the first day. For the same health indicator, the next day, there is a 0.4 probability over the remaining 0.6 probability. Consequently, for the second health indicator, there is a 0.24 probability of raising the alarm. This function modelizes conditional probability: the probability of triggering the alarm in t, considering we had previous probability before t.

The part $\sum_{t \in T}(c^{(r)}(t) * 1)$ in equation 6.3, represent the expected value of triggering an alarm for the run r. In the loss function we aim to maximize, to represent the probabilistic false positive rate (resp. *true positive rate*) , we sum the expected value of triggering an alarm over each healthy (resp. *failing*) run and divide this addition by the amount of actual healthy (resp. *failing*) runs. Then we can use these rates to compute the F_{score}.

This optimization problem is however particularly computationally heavy. For the 7 features used to train our predictor, there are 7 corresponding weights which are variable. Adding the threshold k and the probability range l append two more variables. For one loss function evaluation, each of the 91 days of the 5886 runs is reviewed and passed through several functions.
Moreover the cumulative function $c^{(r)}(t)$ is recursive since it use the function $p^{(r)}(u)$ for each $u < t$. This particularity does not allow us to compute manually the gradient. Indeed, we never know how many health indicator value will exceed the probability threshold and gives a non-null probability of being triggered. Therefore, each analyzed run could have a different cumulative function.

To avoid using the manual gradient descent method, we tested to solve the problem with *scipy* optimizer [3]. After implementing the model, the code seems to run for a ridiculously long time when we tested the "basin hopping", the "simplicial homology global optimization (SHGO)" or the "differential evolution" optimization methods from scipy. The performance of this model didn't seem to be reasonable when we try to solve it with a low amount of function evaluations and iterations. We imagine the reason could be a large number of parameters, variables and evaluations of runs during the optimization process.
After failing to optimize this model with *scipy*, we tried to use ampl, an algebraic modeling language used for "for large-scale mathematical computing" [16]. Because of the recursive function 6.4, the modelization of the problem in this language appears to be much difficult.
Finally, a good direction for further research could be using deep learning with tensor flows and recurrent neural networks. The idea would be to do Many-to-one RNN for each run. For each run, every sample would be an input in time and it would result in one output, at the end of the 91 days. The output should be the expectation of the prediction for the run $\sum_{t \in T} (c^{(r)}(t) * 1)$. This approach is however complex to model and would require considerable additional work. Besides, this approach could be inefficient if the evaluation of the function still takes a large amount of time. Maybe a reformulation of the model could improve the performance.

Chapter 7

Conclusion

In this thesis, we proposed a predictor which can determine whether or not we have to change a hard drive disk before it fails. The methods are based on the SMART attributes, collected every day for each disk, and follow three main objectives of timing: predictions within the last 7 or 15 days before the failure and predicting a failure in the whole run.

For the implementation of the predictor, we used the two-level approach, developed by Valentin Hamaide and his co-authors for a high-speed rotating machine [11]. In the first level, we applied SVC for binary and multi-class classification and One-class SVM for anomaly detection. For the decision-making, we determined an optimal threshold based on the built health indicator or its moving average. On our dataset, these methods obtained more than satisfying results. We obtain for the best combination of parameters a *F1_score* of 0.5763, while most of the literature reviews got lower scores. This metric is obtained via a Support vector 6-class classifier, predicting a failure in the 91 days before. Besides, applied to the hard drive disk data, the two-level approach provides the same conclusions on the efficiency of the multivariate supervised method. Indeed, supervised methods perform better than univarite and anomaly detection. Contrary to the study [11], multi-class methods perform better than binary classification.

In a second time, we highlight a different way to make the decision to trigger an alarm in the second level. We add a probability measure which reveals improvement of scores for most of the methods tried. Indeed, using probability improves the score by 2.02 %, in our best case. Our best *F1_score* is therefore 0.5965.

Adding a probability measure as we did in this thesis, already allows us to improve our prediction scores for this dataset. In addition to that, it could increase the quality of our prediction in very particular cases such as the appearance of

isolated health value changes in a run or in the case of small recurrent decreases in health indicator values, which in the deterministic case, would never exceed the threshold but could nevertheless represent a failure behavior.

In this paper, a final idea has been presented for possible future improvement of the model. By incorporating the probability in the training of our predictor, we could expect even more improvement in the prediction scores.

Chapter 8

Appendix

8.1 Occurence tables

Binary methods feature repartition

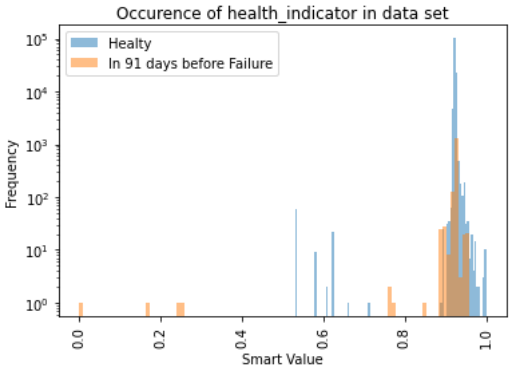


Figure 8.1: repartition of the train data samples of the health indicator created by a binary classification 0-91 with $DayBefore$ 91

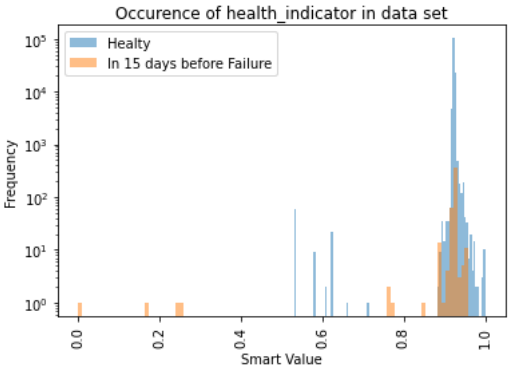


Figure 8.2: repartition of the train data samples of the health indicator created by a binary classification 0-91 with $DayBefore$ 15

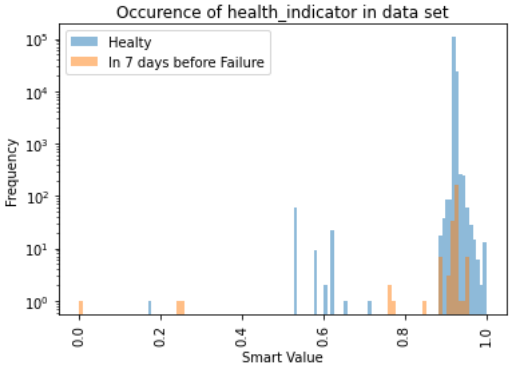


Figure 8.3: repartition of the train data samples of the health indicator created by a binary classification 0-91 with $DayBefore$ 7

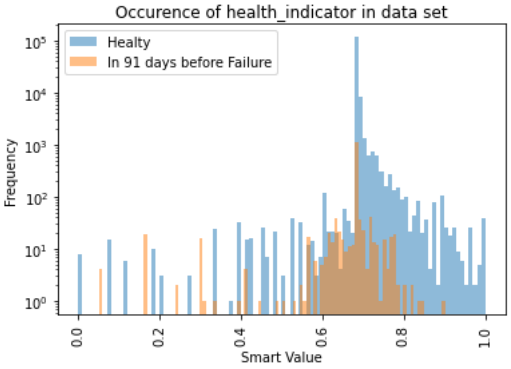


Figure 8.4: repartition of the train data samples of the health indicator created by a binary classification 0-15 with $DayBefore$ 91

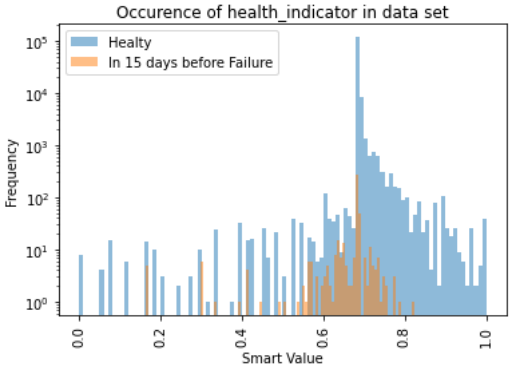


Figure 8.5: repartition of the train data samples of the health indicator created by a binary classification 0-15 with $DayBefore$ 15

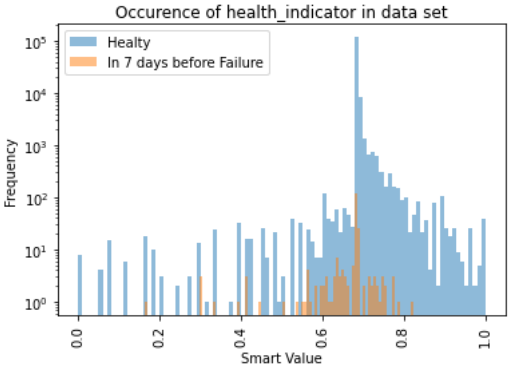


Figure 8.6: repartition of the train data samples of the health indicator created by a binary classification 0-15 with $DayBefore$ 7

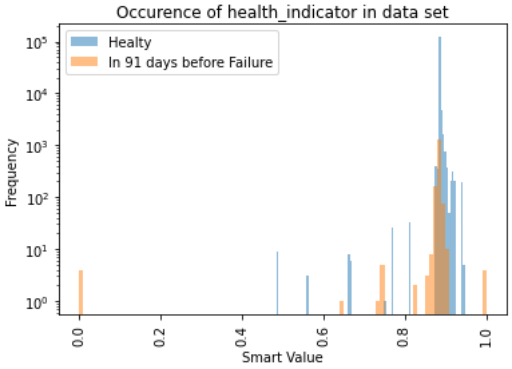


Figure 8.7: repartition of the train data samples of the health indicator created by a binary classification 0-7 with $DayBefore$ 91

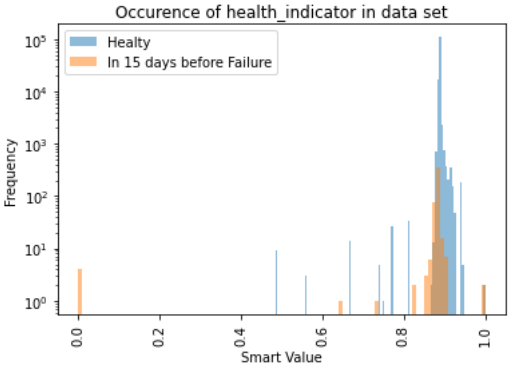


Figure 8.8: repartition of the train data samples of the health indicator created by a binary classification 0-7 with $DayBefore$ 15

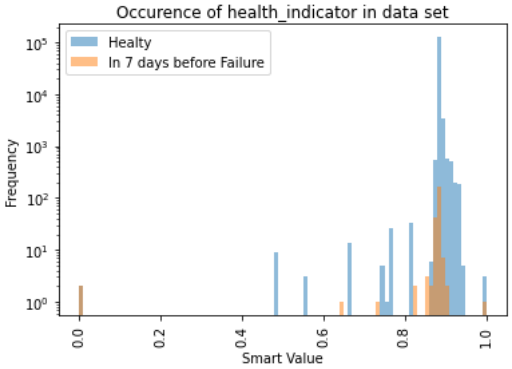


Figure 8.9: repartition of the train data samples of the health indicator created by a binary classification 0-7 with $DayBefore$ 7

Multi-class methods feature repartition

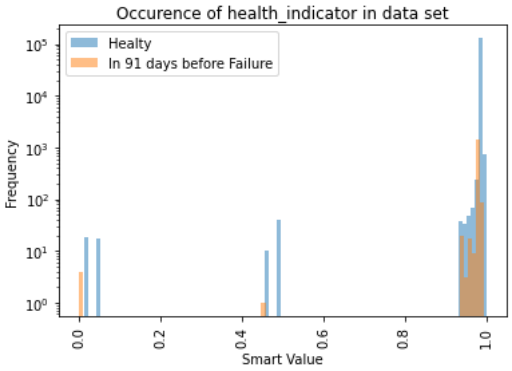


Figure 8.10: repartition of the train data samples of the health indicator created by a multi classification 0/7/15 with $DayBefore$ 91

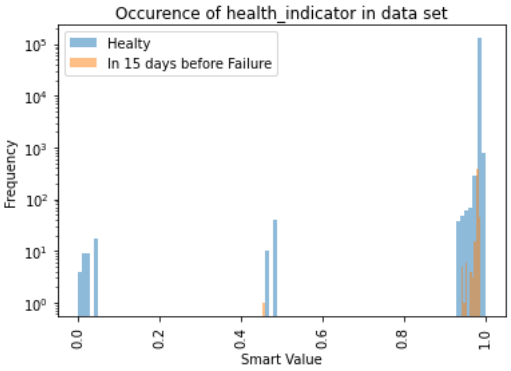


Figure 8.11: repartition of the train data samples of the health indicator created by a multi classification 0/7/15 with $DayBefore$ 15

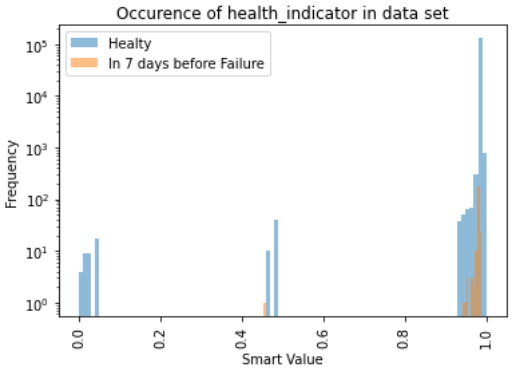


Figure 8.12: repartition of the train data samples of the health indicator created by a multi classification 0/7/15 with $DayBefore$ 7

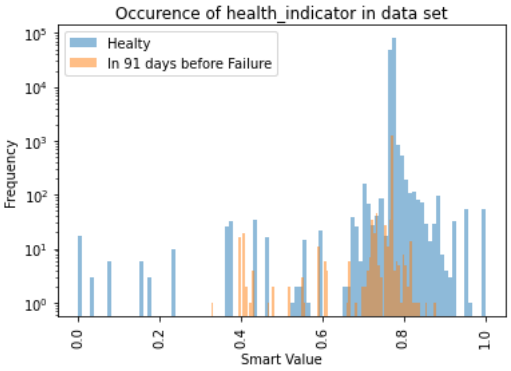


Figure 8.13: repartition of the train data samples of the health indicator created by a multi classification 0/15/30 with $DayBefore$ 91

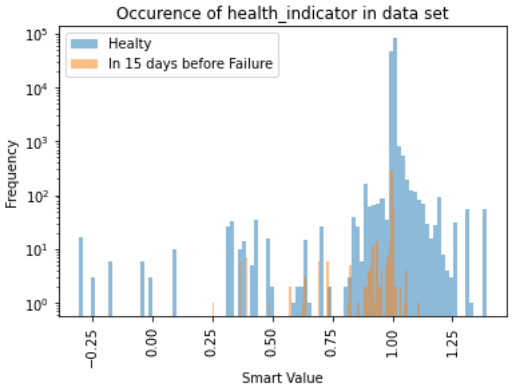


Figure 8.14: repartition of the train data samples of the health indicator created by a multi classification 0/15/30 with $DayBefore$ 15

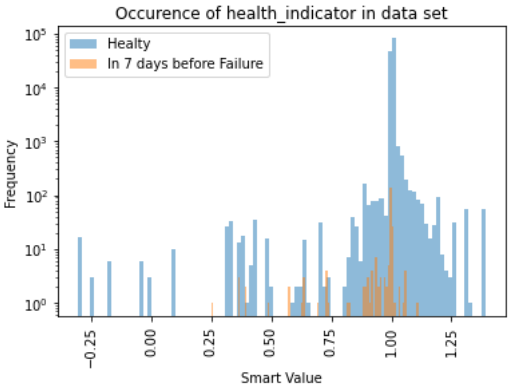


Figure 8.15: repartition of the train data samples of the health indicator created by a multi classification 0/15/30 with $DayBefore$ 7

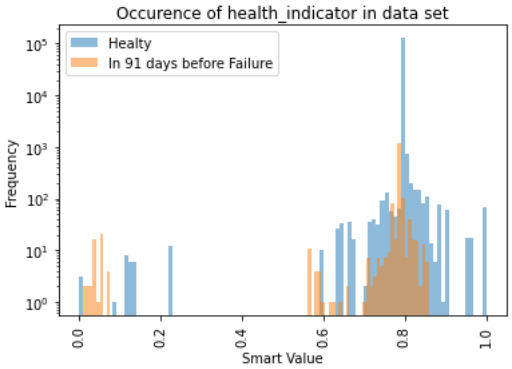


Figure 8.16: repartition of the train data samples of the health indicator created by a multi classification 0/5/10/15/20/25 with $DayBefore$ 91

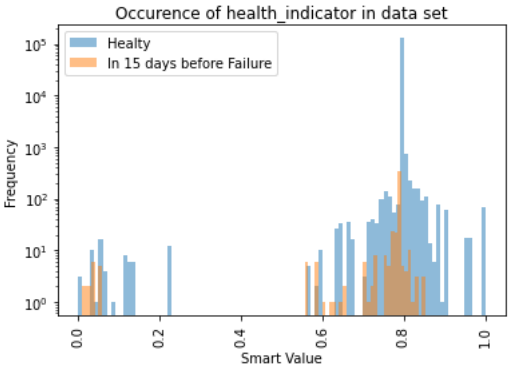


Figure 8.17: repartition of the train data samples of the health indicator created by a multi classification 0/5/10/15/20/25 with $DayBefore$ 15

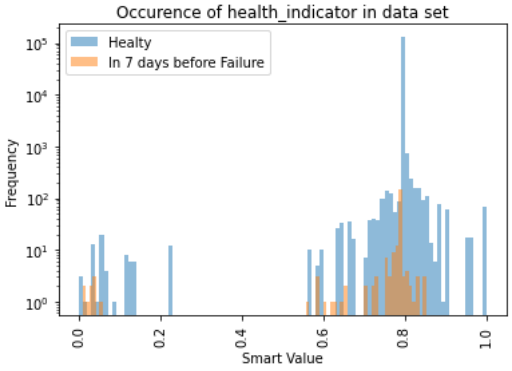


Figure 8.18: repartition of the train data samples of the health indicator created by a multi classification 0/5/10/15/20/25 with $DayBefore$ 7

8.2 Metrics tables

		DayBefore 91	
UNIVARIATE:	**ProbaRange**	Not Rolling average	Rolling average
Smart05:	0	0.2222 / 1 / 0.125 / 0 /0	0.2222 / 1 / 0.125 / 0 /0
	5	0.2222 / 1 / 0.125 / 0 /0	0.2222 / 1 / 0.125 / 0 /0
Smart187:	0	0.4444 / 0.5455 / 0.375 / 0.0069 / 0.045	0.25 / 0.375 / 0.1875 / 0.0069 / 0.06
	2	0.38 / 0.5 / 0.3125 / 0.0069 / 0.054	0.25 / 0.375 / 0.1875 / 0.0069 / 0.06
	5	0.32 / 0.4444 / 0.25 / 0.0069 / 0.0675	0.25 / 0.375 / 0.1875 / 0.0069 / 0.09
Smart188:	0	0.0423 / 0.0217 / 1 / 1 /0.0291	0.0423 / 0.0217 / 1 / 1 /0.0291
Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0
		DayBefore 15	
		Not Rolling average	Rolling average
Smart05:	0	0.1111 / 0.5 / 0.0625 / 0.0014 /0.09	0.0606 / 1 / 0.0313 / 0 /0.36
	5	0.1111 / 0.5 / 0.0625 / 0.0014 /0.09	0.0571 / 0.3333 / 0.0313 / 0.0014 /0.8
Smart187:	0	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.1395 / 0.2727 / 0.0938 / 0.0056 / 0.12
	2	0.1961 / 0.2632 / 0.1563 / 0.0097 / 0.11	0.1395 / 0.2727 / 0.0938 / 0.0056 / 0.17
	5	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.077	0.1 / 0.25 / 0.0625 / 0.0042 / 0.18
Smart188:	0	0.0013 / 0.0007 / 0.03125 / 1 / 0.93	0.0013 / 0.0007 / 0.03125 / 1 / 0.93
Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0
		DayBefore 7	
		Not Rolling average	Rolling average
Smart05:	0	0.0606 / 1 / 0.0313 / 0 /0.95	0.0606 / 1 / 0.0313 / 0 /0.36
	5	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0013 / 0.0007 / 0.0313 / 1 / 1
Smart187:	0	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.0930 / 0.1818 / 0.0625 / 0.0063 / 0.18
	2	0.2308 / 0.3 / 0.1875 / 0.0097 / 0.09	0.0930 / 0.1818 / 0.0625 / 0.0063 / 0.18
	5	0.2963 / 0.3636 / 0.25 / 0.00010 / 0.0675	0.1364 / 0.25 / 0.0938 / 0.0063 / 0.12
Smart188:	0	0 / 0 / 0 / 1 / 0	0 / 0 / 0 / 1 / 0
Smart197:	0	0.1176 / 1 / 0.0625 / 0 / 0.09	0.0606 / 1 / 0.0313 / 0 / 0

Table 8.1: Metrics for univariate methods considering an horizon of 91, 15 and 7 days

MULTIVARIATE: BINARY CLASS:	**ProbaRange**	**DayBefore 91**	
		Not Rolling average	Rolling average
Healty- failure in 91 days	0	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.005	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.01	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.14	0.3111 / 0.5385 / 0.2188 / 0.0042 / 0.21
	0.05	0.44 / 0.6111 / 0.3438 / 0.0049 / 0.17	0.2857 / 0.6 / 0.1875 / 0.0028 / 0.25
	0.1	0.2857 / 0.6 / 0.1875 / 0.0028 / 0.25	0.25 / 0.625 / 0.1563 / 0.0021 / 0.27
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3673 / 0.5294 / 0.2813 / 0.0056 / 0.18	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.26
	0.005	0.36 / 0.5 / 0.2813 / 0.0063 / 0.18	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.29
	0.01	0.3265 / 0.4706 / 0.25 / 0.0063 / 0.21	0.2564 / 0.7142 / 0.1563 / 0.0014 / 0.35
	0.05	0.3333 / 0.5 / 0.25 / 0.0056 / 0.19	0.1905 / 0.4 / 0.125 / 0.0042 / 0.37
	0.1	0.2105 / 0.6667 / 0.125 / 0.0014 / 0.32	0.1111 / 0.5 / 0.0625 / 0.0014 / 0.59
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3265 / 0.4706 / 0.25 / 0.0063 / 0.09	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.12
	0.005	0.32 / 0.4444 / 0.25 / 0.0069 / 0.07	0.2051 / 0.5714 / 0.125 / 0.0021 / 0.22
	0.01	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.11	0.2051 / 0.5714 / 0.125 / 0.0021 / 0.18
	0.05	0.3043 / 0.5 / 0.2188 / 0.0049 / 0.08	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.06
*(F1/Prec/TPR/FPR/BScore)	0.1	0.1579 / 0.5 / 0.0938 / 0.0021 / 0.12	0.15 / 0.375 / 0.0938 / 0.0035 / 0.12

Table 8.2: Metrics for binary 0-91 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
BINARY CLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 15 days	0	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5333 / 0.5714 / 0.5 / 0.0083 / 0.10
	0.005	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.09
	0.01	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5333 / 0.5714 / 0.5 / 0.0083 / 0.08
	0.05	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.12	0.5424 / 0.5926 / 0.5 / 0.0076 / 0.07
	0.1	0.5667 / 0.6071 / 0.5313 / 0.0076 / 0.09	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.09
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.21
	0.005	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.18
	0.01	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.16
	0.05	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.23	0.2414 / 0.2692 / 0.2188 / 0.0132 / 0.19
	0.1	0.3929 / 0.4583 / 0.3438 / 0.0090 / 0.15	0.2667 / 0.2857 / 0.25 / 0.0139 / 0.18
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2333 / 0.25 / 0.2188 / 0.0146 / 0.10
	0.005	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2456 / 0.28 / 0.2188 / 0.0125 / 0.17
	0.01	0.3214 / 0.375 / 0.2813 / 0.0104 / 0.16	0.2666 / 0.2857 / 0.25 / 0.0139 / 0.17
	0.05	0.3333 / 0.4091 / 0.2813 / 0.0090 / 0.08	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.18
*(F1/Prec/TPR/FPR/BScore)	0.1	0.3077 / 0.4 / 0.25 / 0.0083 / 0.26	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.15

Table 8.3: Metrics for binary 0-15 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
BINARY CLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 7 days	0	0.5517 / 0.6154 / 0.5 / 0.0069 / 0.16	
	0.005	0.5614 / 0.64 / 0.5 / 0.0063 / 0.14	0.3272 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.01	0.5313 / 0.5313 / 0.5313 / 0.0104 / 0.15	
	0.05	0.4828 / 0.5385 / 0.4375 / 0.0083 / 0.14	
	0.1	0.25 / 0.625 / 0.1563 / 0.0021 / 0.23	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.13
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3793 / 0.4231 / 0.3438 / 0.0104 / 0.23	
	0.005	0.4138 / 0.4615 / 0.375 / 0.0097 / 0.19	
	0.01	0.3125 / 0.3125 / 0.3125 / 0.0153 / 0.22	
	0.05	0.2807 / 0.32 / 0.25 / 0.0118 / 0.15	
	0.1	0.2 / 0.5 / 0.125 / 0.0028 / 0.18	
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3103 / 0.3462 / 0.2813 / 0.0118 / 0.08	
	0.005	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.17	
	0.01	0.2687 / 0.2571 / 0.2813 / 0.0181 / 0.17	
	0.05	0.2903 / 0.3 / 0.2813 / 0.0146 / 0.17	
*(F1/Prec/TPR/FPR/BScore)	0.1	0.2 / 0.5 / 0.125 / 0.0028 / 0.25	

Table 8.4: Metrics for binary 0-7 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 7 an 15 days	0	0.5454/ 0.6522 / 0.4688 / 0.0056 / 0.16	0.5091 / 0.6087 / 0.4375 / 0.0063 / 0.13
	0.005	0.5185 / 0.6364 / 0.4375 / 0.0056 / 0.14	0.4528 / 0.5714 / 0.375 / 0.0063 / 0.12
	0.01	0.5185 / 0.6364 / 0.4375 / 0.0056 / 0.1§	0.4528 / 0.5714 / 0.375 / 0.0063 / 0.14
	0.05	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.08	0.3922 / 0.5263 / 0.3125 / 0.0063 / 0.13
	0.1	0.1111 / 0.5 / 0.0625 / 0.0014 / 0	0.0426 / 0.0218 / 1 / 0.9979 / 0.06
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.4074 / 0.5 / 0.3438 / 0.0076 / 0.20	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.20
	0.005	0.3774 / 0.4762 / 0.3125 / 0.0076 / 0.20	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.16
	0.01	0.3774 / 0.4762 / 0.3125 / 0.0076 / 0.17	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.19
	0.05	0.3571 / 0.4167 / 0.3125 / 0.0097 / 0.23	0.2745 / 0.3684 / 0.2188 / 0.0083 / 0.17
	0.1	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0040 / 0.0020 / 0.0938 / 1 / 0.65
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3390 / 0.3704 / 0.3125 / 0.0118 / 0.14	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.11
	0.005	0.3396 / 0.4286 / 0.2813 / 0.00083 / 0.21	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.25
	0.01	0.3509 / 0.4 / 0.3125 / 0.0104 / 0.21	0.2963 / 0.3636 / 0.25 / 0.0097 / 0.22
	0.05	0.2400 / 0.3333 / 0.1875 / 0.0083 / 0.06	0.2642 / 0.3333 / 0.2188 / 0.0097 / 0.22
*(F1/Prec/TPR/FPR/BScore)	0.1	0.0556 / 0.25 / 0.0313 / 0.0021 / 0	0.0013 / 0.0007 / 0.0313 / 1 / 0.95

Table 8.5: Metrics for multi-class 0/7/15 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
Healty- failure in 15 an 30 days	0	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.12	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.005	0.5161 / 0.5333 / 0.5 / 0.0097/ 0.13	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.01	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.11	0.5161 / 0.5333 / 0.5 / 0.0097/ 0.08
	0.05	0.5246 / 0.5517 / 0.5 / 0.0090 / 0.14	0.5079 / 0.5161 / 0.5 / 0.0104 / 0.10
	0.1	0.5172 / 0.5769 / 0.4688 / 0.0076 / 0.14	0.5 / 0.5357 / 0.4688 / 0.0090 / 0.06
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.20	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.18
	0.005	0.3226 / 0.3333 / 0.3125 / 0.0139 / 0.22	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.18
	0.01	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.22	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.16
	0.05	0.3509 / 0.4 / 0.3125 / 0.0104 / 0.18	0.2545 / 0.3043 / 0.2188 / 0.0111 / 0.16
	0.1	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.18	0.2581 / 0.2667 / 0.25 / 0.0153 / 0.16
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.2909 / 0.3478 / 0.25 / 0.0104 / 0.07	0.2623 / 0.2759 / 0.25 / 0.0146 / 0.05
	0.005	0.3019 / 0.3810 / 0.25 / 0.009 / 0.19	0.2759 / 0.3077 / 0.25 / 0.0125 / 0.15
	0.01	0.3019 / 0.3810 / 0.25 / 0.009 / 0.15	0.2581 / 0.2667 / 0.25 / 0.0153 / 0.17
	0.05	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.32	0.25 / 0.2917 / 0.2188 / 0.0118 / 0.16
*(F1/Prec/TPR/FPR/BScore)	0.1	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.15	0.2545 / 0.3043 / 0.2186 / 0.0111 / 0.16

Table 8.6: Metrics for multi-class 0/15/30 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
MULTICLASS:	**ProbaRange**	Not Rolling average	Rolling average
6 Classes between 0 and 25	0	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.09
	0.005	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5667 / 0.6071 / 0.5313 / 0.0076 / 0.08
	0.01	0.5763 / 0.6296 / 0.5313 / 0.0069 / 0.10	0.5574 / 0.5862 / 0.5313 / 0.0083 / 0.08
	0.05	0.5862 / 0.6538 / 0.5313 / 0.0063 / 0.08	0.5091 / 0.6087 / 0.4375 / 0.0063 / 0.07
	0.1	0.5965 / 0.68 / 0.5313 / 0.0056 / 0.09	0.5357 / 0.625 / 0.4688 / 0.0063 / 0.08
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3860 / 0.44 / 0.3438 / 0.0097 / 0.16	0.3051 / 0.3333 / 0.2813 / 0.0125 / 0.16
	0.005	0.3793 / 0.4231 / 0.3438 / 0.0104 / 0.18	0.3051 / 0.3333 / 0.2813 / 0.0125 / 0.14
	0.01	0.3729 / 0.4074 / 0.3438 / 0.0111 / 0.18	0.2950 / 0.3103 / 0.2813 / 0.0139 / 0.17
	0.05	0.3729 / 0.4074 / 0.3438 / 0.0111 / 0.16	0.2456 / 0.28 / 0.2188 / 0.0125 / 0.18
	0.1	0.3929 / 0.4583 / 0.3438 / 0.009 / 0.14	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.19
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.08	0.2712 / 0.2963 / 0.25 / 0.0132 / 0.07
	0.005	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.08	0.2712 / 0.2963 / 0.25 / 0.0132 / 0.05
	0.01	0.3103 / 0.3461 / 0.2813 / 0.0118 / 0.08	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.15
	0.05	0.3158 / 0.36 / 0.2813 / 0.0111 / 0.17	0.2951 / 0.3103 / 0.2813 / 0.0139 / 0.11
*(F1/Prec/TPR/FPR/BScore)	0.1	0.2857 / 0.3333 / 0.25 / 0.0111 / 0.05	0.2182 / 0.2609 / 0.1875 / 0.0118 / 0.20

Table 8.7: Metrics for multi-class 0/5/10/15/20/25 method considering an horizon of 91, 15 and 7 days

MULTIVARIATE:		**DayBefore 91**	
UNSUPERVISED:	**ProbaRange**	Not Rolling average	Rolling average
OneClass SVM	0	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.005	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.15	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.01	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.17	0.3226 / 0.3333 / 0.3125 / 0.0139 / 0.13
	0.05	0.4706 / 0.6316 / 0.375 / 0.0049 / 0.15	0.3273 / 0.3913 / 0.2813 / 0.0097 / 0.10
	0.1	0.4615 / 0.6 / 0.375 / 0.0055 / 0.20	0.3279 / 0.3448 / 0.3125 / 0.0132 / 0.13
		DayBefore 15	
		Not Rolling average	Rolling average
	0	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.22	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.005	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.30	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.01	0.3529 / 0.4737 / 0.2813 / 0.0069 / 0.22	0.1333 / 0.1163 / 0.1563 / 0.0264 / 0.40
	0.05	0.36 / 0.5 / 0.2813 / 0.0063 / 0.18	0.1558 / 0.1333 / 0.1875 / 0.0271 / 0.27
	0.1	0.3396 / 0.4286 / 0.2813 / 0.0083 / 0.30	0.1569 / 0.2105 / 0.125 / 0.0104 / 0.55
		DayBefore 7	
		Not Rolling average	Rolling average
	0	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.14	0.1143 / 0.0822 / 0.1875 / 0.0465 / 0.12
	0.005	0.3137 / 0.4211 / 0.25 / 0.0076 / 0.11	0.1143 / 0.0822 / 0.1875 / 0.0465 / 0.12
	0.01	0.2813/ 0.2813 / 0.2813 / 0.0160 / 0.37	0.1132 / 0.0811 / 0.1875 / 0.0472 / 0.12
	0.05	0.2857 / 0.4118 / 0.2188 / 0.0069 / 0.13	0.1558 / 0.1333 / 0.1875 / 0.0271 / 0.22
*(F1/Prec/TPR/FPR/BScore)	0.1	0.28/ 0.3889 / 0.2188 / 0.0076 / 0.10	0.1351 / 0.1190 / 0.1563 / 0.0257 / 0.38

Table 8.8: Metrics for one-class methods considering an horizon of 91,15 and 7 days

Bibliography

[1] Backblaze Hard Drive Stats. https://www.backblaze.com/b2/hard-drive-test-data.html#overview-of-the-hard-drive-data. Accessed: 2022-05-10.

[2] Hard Drive Test Data. https://www.kaggle.com/backblaze/hard-drive-test-data. Accessed: 2022-05-26.

[3] SciPy. https://scipy.org/. Accessed: 2022-06-05.

[4] sklearn.svm.SVC. https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2022-05-11.

[5] XGBoost Documentation — xgboost 1.6.1 documentation. https://xgboost.readthedocs.io/en/stable/. Accessed: 2022-05-17.

[6] How to Manage for Hard Drive Failures and Data Corruption. https://www.backblaze.com/blog/managing-for-hard-drive-failures-data-corruption/, July 2019. Accessed: 2022-05-31.

[7] Machine Learning for Predictive Maintenance - Part 2: Predicting Hard Drive Failure. https://datatonic.com/insights/machine-learning-predictive-maintenance-hard-drive-failure/, March 2020. Accessed: 2022-05-017.

[8] Klein A. Using Machine Learning to Predict Hard Drive Failures. https://www.backblaze.com/blog/using-machine-learning-to-predict-hard-drive-failures/, October 2021. Accessed: 2022-05-10.

[9] Maxime Amram, Jack Dunn, Jeremy J. Toledano, and Ying Daisy Zhuo. Interpretable predictive maintenance for hard drives. *Machine Learning with Applications*, 5:100042, 2021.

[10] Carlos A. C. Rincón, Jehan-François Pâris, Ricardo Vilalta, Albert M. K. Cheng, and Darrell D. E. Long. Disk failure prediction in heterogeneous environments. In *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 1–7, July 2017.

[11] Valentin Hamaide, Denis Joassin, Lauriane Castin, and François Glineur. A two-level machine learning framework for predictive maintenance: comparison of learning formulations. Technical Report arXiv:2204.10083, arXiv, April 2022.

[12] Delua Julianna. Supervised vs. Unsupervised Learning: What's the Difference? `https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning`, March 2021. Accessed: 2022-05-11.

[13] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. Making Disk Failure Predictions SMARTer! *18th USENIX Conference on File and Storage Technologies*, page 19.

[14] Souza, Cesar. Kernel Functions for Machine Learning Applications. `http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/`, March 2017. Accessed: 2022-06-05.

[15] Kevin Speyer. Using a predictive maintenance model of hard drive disks failure to back up data in a cost-effective way. *CYBERTEC | Data Science PostgreSQL Services*, page 10, June 2019.

[16] Wikipedia contributors. Ampl — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=AMPL&oldid=1088071785`, 2022. Accessed: 2022-06-02.

[17] Wikipedia contributors. Receiver operating characteristic — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=1089770766`, 2022. Accessed: 2022-05-24.

[18] Wikipedia contributors. S.m.a.r.t. — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=S.M.A.R.T.&oldid=1087627755`, 2022. Accessed: 2022-05-10.

[19] Zeki Murat Çınar, Abubakar Abdussalam Nuhu, Qasim Zeeshan, Orhan Korhan, Mohammed Asmael, and Babak Safaei. Machine Learning in Predictive

Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. *Sustainability*, 12(19):8211, January 2020. Publisher: Multidisciplinary Digital Publishing Institute.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**