POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

A Machine Learning approach to Predictive Control: study on a real industrial application

Relatore:
Prof.ssa Elena Maria Baralis

Abdeljalil Hajjoubi

Supervisori aziendali
Istituto Superio Mario Boella
Dott.ssa Sophie Marie Fosson
Dott.ssa Rosaria Rossini

Anno accademico 2016-2017

Summary

For many years, Model Predictive Control (MPC) has been used in industries. MPC is a control methodology to predict the response of a system given a process model [33]. MPC has had a great impact on industrial process control since 1980s. This is due to the facts that its idea is easy to understand and its formulation extends to multivariate plants.

In MPC, the control is based on a model which has quite stringent mathematical constraints; most of MPC literature is based on linear models, i.e., a linear function is assumed to describe the relation between input and output of the controled system. In many cases, this is a simplification of the real system, but has the advantage to be mathematically tractable. Nowadays, we are surrounded by cheap sensors that collect a lot of data (the so-called Big Data) and we are moving towards industry 4.0, from model-driven control to data-driven control. We can use this data to create the relation of the control system. The classic models, i.e., the linear model, often, are very restrictive for the big amount of available data. Machine Learning (ML), and especially Deep Learning (DL), helps us to build more complex relations between input and output of the system, taking better advantage of Big Data. For this reason, we are moving to them. DL allows us to build non-linear models, through Artificial Neural Networks (ANNs), which are more complex and need more powerful CPUs and GPUs. At the same time, we have more powerful computing resources (CPUs and GPUs) able to process Big Data and build these models.

In this thesis, we have studied new solutions based on ML to perform Predictive Control (PC). In the past, PC was based on manual measurements that, once reported in spreadsheets, allow to define a programmed intervention. The advent more and more domineering of the Internet of Things (IoT) has changed the way PC can be performed. With these devices, it is possible to have a low cost and a real time monitoring of the industry equipments. Thanks to the continuous sending of data to the algorithm, this one can monitor when an equipment is

not working as it should, or make a prediction using ML. Predictive Maintenance (PdM) is exactly going to this direction, and the mentioned solution can be defined as "IoT PC". The advantages that derive from having sensors that detect data in real time are remarkable. This real-time data is inserted into predictive models to determine when something is going to break, or what is the residual life of a machine. Based on this information, it will be possible to program maintenance. In particular, it is effective for all manufacturing companies that want to provide a PdM service along with the machinery they offer [24].

We have worked on a real application, namely on data coming from a machine (called Buss Mixer) of an aluminium process industry. This data, acquired in the framework of the MONSOON project[1], is available through a network of sensors that monitors different variables of this machine every five seconds and collects data in files. The data is a time series; there are some measurments in which the machine has had a failure (and the associated line in the file is labeled as failure) and others in which the machine has worked regularly (the label in the file is no-failure). These files have been processed by the classification ML algorithms to implement PdM. We have used the standard classification algorithms and the ANNs algorithms: amongst the first, we have used LR, SVM, DT and RF. A way to do PdM has been to introduce the Predictive Horizon (PH) in the data: we have labeled as failure also the previous m measurments. In this way, the classification algorithms may extract some patterns which are used to predict future breakdowns.

With the studied schemes, we have been able to predict the failures of the Buss Mixer successfully with some minutes of advance, and with this, we have reached the objective of this thesis. The obtained results are similar to those obtained by the experts of MONSOON project. This work can be enhanced; an open problem is predicting failures with more advance to meet more the ability of intervention of the industry. For this, we will need more accurate data.

The thesis has been carried out in Istituto Superiore Mario Boella (ISMB)[2] and the main activities have been the following:

- Theoretical study of ML and DL techniques: we have analyzed the state of the art of ML and DL algorithms, especially the classification, ANNs, and classic ones;

[1]https://www.spire2030.eu/monsoon

[2]http://www.ismb.it/

- Pre-processing of data: we have processed it using Pandas and Numpy libraries;
- Implementation of the classification algorithms in Python and Tensorflow (TF): we have implemented the classic classification algorithms (mensioned before) in Python, using Scikit-learn library. TF library has been used to implement the ANNs;
- Analysis of the data;
- Study of the best strategies to make prediction: we have relied on some indices to choose the best strategies;
- Comparison/validation of our results with the results obtained from the MONSOON experts.

Glossary

ANN	Artificial Neural Network. iii, iv, 16, 20, 44, 68, 70, 71
CNN	Convolutional Neural Network. 6, 67, 71
DL	Deep Learning. ii, iii, 2, 70
DNN	Deep Neural Network. 8, 16, 32, 66, 68, 71
DT	Decision Tree. iii, 13–15, 20, 24, 30–32, 36, 37, 48, 54, 57, 60–62, 64, 65
GCML	Google Cloud Machine Learning. 3
IoT	Internet of Things. iii, 3
KNN	K-Nearest Neighbors. 9, 20, 23, 24, 30
LR	Logistic Regression. iii, 9–12, 15, 17, 18, 20, 23, 24, 30, 31, 37, 41, 54, 60, 63–65
LTU	Linear Threshold Unit. 16
ML	Machine Learning. ii, iii, 2–9, 11, 13, 15, 16, 21, 25, 29–33, 44, 47, 68, 70
MLP	Multilayer Perceptron. 17, 31, 44, 60, 64–66
MPC	Model Predictive Control. ii, 2
PC	Predictive Control. ii, iii, 1, 4, 25, 26, 70
PdM	Predictive Maintenance. iii, 1–4, 26, 29, 30

PH Predictive Horizon. 30, 33, 36, 37, 44, 47, 70

RF Random Forest. iii, 13, 15, 20, 24, 30, 31, 36, 37, 39, 54, 58, 60, 62, 64

RNN Recurrent Neural Network. 67, 71

SVM Support Vector Machine. iii, 11, 12, 14, 15, 18, 20, 23, 30, 31, 37, 42, 48, 54–56, 60, 64, 65, 68

TF TensorFlow. iv, 30, 32, 67, 68

Contents

Chapter 1

Introduction

In this chapter, we introduce the problem of PC in process industries. Afterwards, we introduce some possible solutions present in the market to face the PdM problem.

1.1 Predictive Control in process industries

The availability of data is changing the manner decisions are taken in industry [1], in areas such as scheduling [2], maintenance management [3], and quality improvement [4], [5]. Furthermore, the management of maintenance is becoming important to decrease the costs associated with downtime [6], especially in manufacturing industries such as semiconductor manufacturing.

We can group the approaches to maintenance management into three categories [7]:

1. Run-to-failure - where maintenance is performed after the occurrence of failures. This is obviously the simplest approach and the most adopted one, but it is also the least effective, as the cost of interventions and associated downtime are much more higher than those associated with planned interventions.
2. Preventive maintenance - where maintenance is carried out based on a scheduled plan. With this approach, failures are usually prevented, but sometimes unnecessary corrective actions are often performed.
3. Predictive Maintenance (PdM) - where maintenance is performed based on

the health status of a machine. PdM systems allow us to detect failures and to make interventions before their occurrences. This is possible thanks to historical data.

Among PdM methods, the classic one is MPC. Nowadays, the most used solution is the combination of ML techniques and data collected by sensors. MPC, like ML algorithms, has the objective to minimize a functional cost. The difference is that it is based on more rigid models. The classic MPC is linear and the optimization problem to solve is convex, and therefore resolvable; but the linear model itself is rigid. ML techniques (and particularly DL ones) move from a model-based approach to a data-based one. This is the relevant difference. Models that are data-based are less rigid because they are constructed on a lot of data.

1.2 Use of Machine Learning in industry: state of the art

In this section, we introduce the state of the art of the use of ML in industry.

In general, to do PdM, we add some sensors to the system that will monitor and collect data. Data used for PdM is, in general, a time series collected with a given frequency. The goal of PdM is to predict at the time "t", using the data previous to this time, if the equipment will fail in future. There are two approaches:

- Classification approach - predicts if there ican be a failure in the next steps.
- Regression approach - predicts how much time is left before the next failure. We call this *remaining useful life*.

The first approach provides a boolean result, and it can provide greater accuracy with less data. The second approach use more data but it provides the information about when the failure will occure. A brief example is shown below.

Turbofan engine degradation dataset. Turbofan engine is a gas turbine engine used by the NASA space exploration agency. NASA has created a dataset to predict the failures of Turbofan engines. The dataset is available at PCoE Datasets [1].

[1] http://ti.arc.nasa.gov/c/6/

In the dataset, there is a time series for each engine. Each engine starts with different degrees of wear, which the user does not know. Each engine has 21 sensors that collects different measurements related to the engine state. Collected data is contaminated with sensor noise.

The engines develop some faults. Data includes engine number, time stamps, some settings variables, and readings for 21 sensors. The objective is to predict when the next failure will occur. This is not very different from our case study: we need to predict the next failure 45 minutes before its occurrence.

1.2.1 Some industrial initiatives exploiting Machine Learning

In this section, we briefly present some outstanding industrial initiatives where ML has been introduced.

IoT and ML for industrial PdM. We could combine IoT and ML together: Attach sensors everywhere and extract infomation from the data collected by sensors. A complete tool chain that can help us is Google Cloud ML (GCML) [2] and the Losant IoT Platform[3]. GCML engine is a service that allow to build and deploy ML models easily. Losant is an IoT industry that allows to create scalable IoT solutions. The system provided by Google Cloud and Losant is a complete system that take data in input, build and run a ML model, and predict what we need (generating alerts and visualizing data). These combined tools provide many benefits (quick time to production, increased efficiency, cost reductions, scalable, flexible, and reusable IoT architecture). In this way, we can build a complete PdM system.

Some implementations exploit the sound and the *vibration analysis* to know when an equipment may be working outside of its normal condition.

ACME. ACME Industrial[4] has launched an initiative to monitor the health status of its equipment using a system that launch an alert if there is a problem. This information will be used by the PdM.

As we have mentioned before, a possible solution to this problem is combining the Losant IoT Platform and GCML engine.

[2]https://cloud.google.com/ml-engine/?hl=en

[3]https://www.losant.com/

[4]http://www.acmeindustrial.com/

Innovative testbed. The Industrial Internet Consortium (IIC) is the leading organization that has transformed business and society accelerating the adoption of the Industrial Internet of Things (IIoT). It has announced the Smart Factory Machine Learning for Predictive Maintenance testbed, ledded by two companies, Plethora IIoT and Xilinx.

This testbed explores ML techniques for time-critical PdM. This will help industries to increase the availability, improve the efficiency and extend the life time industrial equipments. Downtime costs are very high for an industry and may reach a lot of millions of dollars per minute. Therefore, unexpected failures are one of the main problems [21].

A Microsoft solution to Predictive Maintenance. Microsoft Azure ML is a technique designed by Microsoft to enable computers to recognize and learn from existing data in order to be able to predict future behavior. Devices and applications are maked smarter by predictions based on ML data, supplied by devices that break down as well as functioning properly. This data can be used by ML to detect trends. The result provide an impression of the value, at which we should receive a notification indicating that the device is about to break down and therefore prevent failure before it occurs [22].

In Chapter 3, we will develop ML strategies for a specific industrial PC problem. Before that, in Chapter 2, we will review some basic principles of ML and the algorithms that will be considered for our purpose.

Chapter 2

Machine Learning algorithms for classification

In this chapter, we introduce ML, with special focus on the classification methods which are of interest for our real application study. At the end of the chapter, we will propose some preliminary tests, that will allow us to investigate the efficiency of the different algorithms, and then to choose the algorithms suitable for our purpose.

2.1 Introduction to Machine Learning

When we hear about ML, our mind immediately deeps on a picture of a robot beeing ordered to do something by humans, or at least, this was what I tought the first time. This has something of magic, but it is all about Mathematics. However, ML is not just this; it surrounds us and perhaps we do not know it. It has been used for many years in many applications, from filtering spam to other applications such as optical character recognition (recognition of handwritten characters) [9], better reccomendations (Amazon), or voice recognition (Siri, Cortana, Google Assistant, etc).

Here, we propose definitions minted by some authors:

"Machine Learning is the science of programming computers so they can learn from data." Aurélien Géron [10]

"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." Arthur Samuel, 1959 [11]

"Machine learning is about predicting the future based on the past." Hal Daumé III [12]

The last definition gives us a clear idea of what is ML: we train our system on many data (*training data*) and when a new data arrives (a new *instance*), the system knows to tell us something about them. A famous example is the spam filter that recognize if an email is spam or not (also called *"ham"*). It is curious how ML can solve some tasks that are complex or impossible to aolve through a traditional program written in some computer languages.

Why not simply write the spam filter by using traditional programming techniques? We should proceed in this way: when an email arrives, checking if it contains, in the fields or in its body, the most frequently used words by spammers; if it contains them, classifying the new email as ham.

However, this job will becomes tedious because spammers are always at work, trying new words; hence, we always need to update the traditional program. Furthermore, the program will become difficult to maintain because of the big list of written rules.

A ML algorithm can detect automatically new spam emails. The trick is that it learns from the big amount of emails we feed it: it tries to find how new emails are similar to spam emails and as a result, it classifies them.

The spam filter is a simple application of ML. A complex one is the voice recognition. How can we solve it? The best solution, again, is to write an algorithm that learns by itslef, giving many recordings example for each word.

An incredible task, difficult for humans, is finding patterns not immediatily apparent and extracting useful information from a large amount of data. This process is called *data mining.*

ML is used when:

- we write a traditional program and this involves many complex rules and difficult maintenance
- we face a complicated problem like voice or image recognition solvable with ML techniques (see CNN in chapter 4)
- we need to understand the data by finding patterns not immediately visible into a large amount of data.

2.1.1 Supervised vs unsupervised learning

In ML, we distinguish two main categories of approach, the supervised and the unsupervised learning.

Supervised learning. The data that we give to the ML algorithm have the desired solutions (*labels*). A supervised learning task is the classification and an example of it is the spam filter: giving to the classifier the emails and telling it if they are spam or ham, it will classify (predict) the new emails based on what it have learned.

Another supervised learning task is regression that provides a numeric value instead of a label.

The most popular supervised learning algorithms are:

- K-Nearest neighbors (KNN);
- Logistic Regression (LR);
- Support Vector Machines (SVM);
- Decision Tree (DT);
- Random Forest (RF);
- Artificial Neural Networks (ANNs).

These supervised learning algorithms will be studied in depth in the next chapters.

Unsupervised learning. The training data is unlabeled. So the system learns by examining the data and seeing if it can find some patterns on the data itself. An unsupervised learning algorithm is *clustering* (k-Means is one of them); it is used when we want to organize some individuals into groups with similar features.

2.1.2 Data

ML is data. If we do not have a lot of good data, even the best algorithm will not generalize well.

Differently from a toddler, ML algorithms need a lot of data to work well: a famous paper [28] shows how different ML algorithms have almost the same performances on a complex problem of natural language disambiguation when they are fed with enough data. We need to consider the tradeoff between spending energy on algorithm development versus spending it on corpus development. Another paper [29] has emphasized the idea that data matters more than algorithms. However, nowadays, since we still deal with small- and medium-sized datasets, we can not abandon algorithms yet.

A relevant part in a ML project is the choice of a good set of features to train on. Select well the features is very important.

This process is called *feature engineering* and it involves:

- *Feature selection*: choosing the most useful features to train
- *Feature extraction*: combining existing features to produce a more useful one. *Dimentionality reduction* algorithms can do this.
- Creating new features by gathering new data

Overfitting the training data. In ML, *overfitting* means that the model performs well on the training data, but it does not generalize properly on new data.

Complex models such as DNNs (that we will introduce later) can detect thin patterns in the data, but if the training set is noisy, the model will probably detect patterns in the noise and, as a result, the model will not generalize well.

Overfitting happens when the model is complex relative to the amount and noisiness of the training data. Then possible solutions to solve the problem are:

- choosing a simple model, with fewer parameter to tune,
- reducing the number of features in the training data, through *dimentionality reduction* algorithms,
- reducing the noise.

Regularization. Constraining the model to make it simpler and reduce the risk of overfitting is called .

Suppose we use a linear model $y = mx + b$, we need to fit the model to our dataset. The free parameters to tune are m and b; if we want to simplify the algorithm, we can try to constraint m to have low value and allow the algorithm to modify the height b. It is required to find a balance between fitting the data and keeping the model simple enough to generalize well.

The amount of regularization can be controlled by a *hyperparameter* of the learning algorithm. Tuning hyperparameters is an important part of building a ML algorithm and may take time.

2.2 Focus on classification algorithms

In this section, we present the main classification algorithms, that will be used later for our purpose. We will see the Mathematics behind supervised learning algorithms mentioned before and the difference between them. We will also present the famous handwritten digits dataset and some results obtained by applying these ML algorithms. Finally, based on comparison between them, we will select the best supervised learning algorithm for this dataset.

2.2.1 K-Nearest Neighbors

The KNN algorithm is a simple algorithm used for the classification of objects based on features of the objects near the considered one. It is the easiest one amongst ML algorithms.

For the sake of simplicity, we will not explain it in detail because it is very slow in predicting, for large dataset, such as the handwritten digits dataset.

It has a complexity that grows with the number of instances: when a label of a new instance has to be predicted, the KNN algorithm has to find the k nearest labeled instances and then it assigns the tag to the new one based on the majority vote, namely it counts the number of neighbors of each class and it assigns to the new instance the label of the majority class. k is the hyperparameter to tune.

2.2.2 Logistic Regression

The LR classifier is used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class.

Suppose we have K classes. LR model arises from the desire to pattern the posterior probabilities of the K classes via linear functions in x, and ensuring that they sum to one and remain in the interval [0,1], as they are probabilities. x is a vector that contains the values associated to the features of the new instance. We want a linear model because it is the simplest one, and it has the form

$$\log \frac{Pr(G=1|X=x)}{Pr(G=K|X=x)} = \beta_{10} + \beta_1^T x$$
$$\log \frac{Pr(G=2|X=x)}{Pr(G=K|X=x)} = \beta_{20} + \beta_2^T x$$
$$\vdots$$
$$\log \frac{Pr(G=K-1|X=x)}{Pr(G=K|X=x)} = \beta_{(K-1)0} + \beta_{K-1}^T x.$$

β_i are the coefficients of the model and the entire parameter set is $\theta = \{\beta_{10}, \beta_1^T, \cdots, \beta_{(K-1)0}, \beta_{K-1}^T\}$ The model is specified in terms of *K - 1 log-odds* or *logit* transformations. This is a way to reflect the constraint that the probabilities sum to one. Although the model uses the last class as the denominator in the odd-ratios, the choise of denominator is arbitraty. A simple calculation shows that

$$Pr(G=k|X=x) = e^{\beta_{k0}+\beta_k^T x}, k = 1, \cdots, K-1$$
$$Pr(G=K|X=x) = \frac{1}{1+\sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_l^T x}}$$

LR models are fit by maximizing the conditional likelihood of G given X. The log-likelihood for N observations (our instances) is

$$\ell(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i;\theta)$$

where $p_{g_i}(x_i;\theta) = Pr(G=k|X=x_i;\theta)$.

Let us talk about the two-classes case, since the algorithm simplify considerably and we will experiment a test about a binary classification problem. It is convenient to code the two classes g_i via a 0/1 response y_i, where $y_i = 1$ when $g_i = 1$, and $y_i = 0$ when $g_i = 2$. Let $p_i(x,\theta) = p(x,\theta)$, and $p_2(x,\theta) = 1 - p(x;\theta)$. The log-likelihood can be written as

$$\ell(\beta) = \sum_{i=1}^{N} \{y_i \log p(x_i;\beta) + (1-y_i) log 1 - p(x_i;\beta)\}$$
$$= \sum_{i=1}^{N} \{y_i \beta^T x_i - log 1 + e^{\beta^T x_i}\}$$

We need to find the β parameters that maximize the log-likelihood, so we set its derivatives to zero. We obtain *p + 1* non-linear equations in β.

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - p(x_i;\beta))$$

There are many ways to solve the equations and one is the Newton-Raphson algorithm: it starts by a small β and than it tries to adjust it. Typically the algorithm converges, since the log-likelihood is concave, that means that there is only a maximum and we can find it.

Typically many models are fit in a search for a parsimonious model involving a subset of the variables. A *parsimonius model* is a model with a high sparsity, that means a model with many coefficents to zero or very close to zero. Computationally, few coefficients are easy to store in memory and when an instance arrives, the prediction is fast.

ℓ_1 regularized Logistic Regression. In general, we try the ML algorithm witout penalization and see if it fits well the data. if not, we introduce this penalization that could help us to fit well the data. We can introduce an ℓ_1 penalty, corresponding to the definition of absolute value, or we can introduce an ℓ_2 penalty (not much used).

Using ℓ_1 penalty with LR model, we would maximize this log-likelihood:

$$\max_{\beta_0,\beta}\{\textstyle\sum_{i=1}^{N}[y_i(\beta_0+\beta^T x_i)-\log 1+e^{\beta_0+\beta^T x_i}]-\lambda\sum_{j=1}^{p}|\beta_j|\}$$

where λ is the hyperparameter used to tune the model.

2.2.3 Support Vector Machine

SVM is a ML model able of performing linear or non-linear classification, regression, and outlier detection. We will focus only on classification (two-classes case) as we mentioned before.

Linear Support Vector Machine Classification. SVM classifier traces a line that separates the two classes and stays as far away from the closest training instances as possible. It tries to fit the widest street between the classes. This is called *margin classification.* The decision boundary is determined by the instances located on the edge of the street. These instances are called the *support vectors.* SVM is sensitive to the feature scales: changing the scale of features can affect changing in the decision boundary.

If we impose that all instances be outside the street and on the right side, this is called *hard margin classification* and in many cases it is difficult to obtain because there are always some instances that stay outside the street or even on the wrong side.
There are two problems with hard margin classification: it works only if the data is

linearly separable and it is sensitive to outliers. The objective is to find a tradeoff between keeping the street as large as possible and limiting the margin violations. This is the *soft margin classification.* We can control this balance using the C hyperparameter.

Unlike LR classifier, SVM classifier does not output a probability for each class, but it gives us directly the label based on the decision boundary.

Non-linear Support Vector Machine classification. We can handle non-linear datasets by adding more features, such as *polynomial* features; in some cases this can result in a linearly separable dataset. At a low polynomial degree, this aproach can not deal with very complex datasets, and with a high polynomial degree, it creates a great number of features, making the model slow.

To solve this problem, SVM can use the *kernel* trick: it allows to get the same result as if we added many polynomial features. As a result, there is no explosion of the number of features since we do not add any features. In literature, there are few *kernel function* able to run this trick.

Mathematics behind Support Vector Machine. Our training data consists of N pairs
$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$, with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. We want to separate the classes (linearly separable) by hyperplanes that have this equation

$$\{x : f(x) = x^T\beta + \beta_0 = 0\},$$

where β is a unit vector: $||\beta|| = 1$.

The equation of hyperplane gives the signed distance from a point x to the hyperplane $f(x) = x^T\beta + \beta_0 = 0$. The optimization problem

$$\begin{gathered}\max_{\beta,\beta_0,||\beta||=1} M \\ \text{subject to } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, \cdots, N,\end{gathered}$$

captures the concept. M is the margin from the hyperplane, so $2M$ is the "street" that SVM has to maximize. The optimization problem can be rephrased as

$$\begin{gathered}\min_{\beta,\beta_0} ||\beta|| \\ \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \cdots, N,\end{gathered}$$

Suppose now that classes overlap in the feature space. A way to deal with this case is to maximize M and allow for some points to be on the wrong side of the margin (the soft margin classification mentioned before). Let us define the *slack* variables $\xi = (\xi_1, \xi_2, \cdots, \xi_N)$. The constaint can be modified in this way

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i),$$

$\forall i, \xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq constant$. The value ξ_i in the constant $y_i(x_i^T\beta + \beta_0) \geq M(1-\xi_i)$ is the amount by which the prediction $f(x_i) = x_i^T\beta + \beta_0$ is on the wrong side of its margin. The sum $\sum \xi_i$ is the total amount by which predictions fall on the wrong side of their margin. Misclassifications occur when $\xi_i > 1$, so bounding $\sum \xi_i$ at a value K, bounds the total number of training misclassifications at K. The optimization problem becomes

$$\begin{gathered} \min ||\beta|| \text{ subject to} \\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \forall i, \\ \xi_i \geq 0, \sum \xi_i \leq constant. \end{gathered}$$

Computing the support vector classifier. This is a convex optimization problem. We describe a quadratic programming solution using *Lagrange multipliers*. Computationally it is convenient to rephrase the problem in the equivalent form

$$\begin{gathered} \min_{\beta,\beta_0} \tfrac{1}{1}||\beta||^2 + C\sum_{i=1}^{N} \xi_i \\ \text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \forall i, \end{gathered}$$

where the cost parameter C replaces the constant K. We can solve the *primal* or the *dual* problem. The *Lagrange primal* function is

$$L_P = \tfrac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0)) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i\xi_i$$

which we minimize with respect to β, β_0 and ξ_i by setting their derivatives to zero. We find that the solution for β has the form

$$\hat{\beta} = \sum_{i=2}^{N} \hat{\alpha}_i y_i x_i.$$

Given the solutions $\hat{\beta}_0$ and $\hat{\beta}$, the decision function can be written as

$$\hat{G}(x) = sign[\hat{f}(x)] = sign[x^T\hat{\beta} + \hat{\beta}_0].$$

2.2.4 Decision Tree

DT is an other ML algorithm able to perform classification in a different way. DT is a fundamental component of RF which is among the most powerful ML algorithms

available today.
DT classifier requires very little data preparation. In particular, it does not require feature scaling or centering (like SVM).

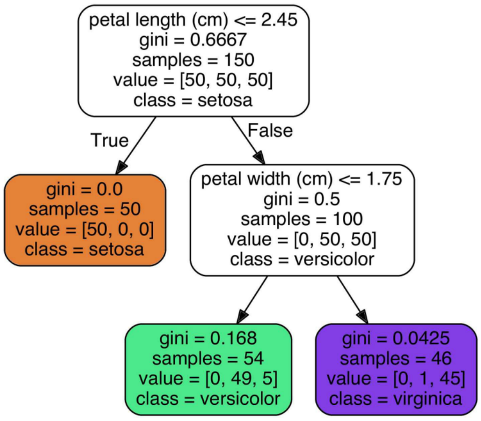

Figure 2.1. Example of a DT for the Iris dataset

How does it make prediction. Let us consider the Iris dataset[1]. It contains 3 classes of 50 instances each, where each class refers to a type of an iris plant. The DT classifier builds a binary tree: when a new instance comes, we need to traverse the tree to decide the class of the instance.

A samples attribute of a node counts how many training instances it applies to. A value attribute of a node tells us how many training instances of each class this node applies to. Finally, a gini attribute of a node measures its *impurity*: a node is "*pure*" (gini = 0) if all training instances it applies to belong to the same class.

The *gini index* is calculated as

$$G_i = 1 - \sum_{k=1}^{N} p_{i,k}^2$$

where $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

How a Decision Tree is built. The most used implementation is *CART (Classification And Regression decision Tree)*: the algorithm splits the training set

[1]https://archive.ics.uci.edu/ml/datasets/iris

in two subsets using a feature k and a threshold t_k. It searches for the pair (k, t_k) that produces the purest subsets. After, it splits the subsets recursively using the same logic until it can not find a split that will reduce impurity. **Hyperparameters.** DT makes few assumptions about the training set. If left unconstrained, the tree structure will adapt itself to the training set, fitting it closely. This model is called a *non-parametric model*, because the number of parameters is not determined before starting the training. A *parametric model* has a predetermined number of parameters reducing the risk of overfitting. To avoid overfitting the training data, we need to restrict the freedom of DT during training.

Decision Tree instability. Decision boundaries created by DT are ortogonal (all splits are perpendicular to an axis), which makes it sensitive to training set rotation.

2.2.5 Random Forest

In ML, we can also use different classifiers together to predict. Suppose we ask a question to thousands of random people, then aggregate their answers. In many cases, we will find that this aggregated answer is better than an answer of an expert. This is called the *wisdom of the crowd.* If we aggregate the predictions of a group of classifiers, we will often get better predictions than with the best individual classifier. A group of classifiers (*predictors* or *estimaros*) is called an *ensemble*; this technique is called *ensemble learning.* For example, we can train a group of DT classifiers, each on a different random subset of the training set. To make predictions, we get the predictions of all individual trees, then predict the class that gets the most votes. This ensemble of DTs is called RF.

A RF classifier has all the hyperparameters of a DT classifier (to control how trees are grown) and all the hyperparameters to control the ensemble itself. The RF algorithm introduces extra randomness; instead of searching for the best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity that lead to a better model and a saving of time.

Feature importance. If we look at a DT, important features appears closer to the root of the tree, while less relevant features appears closer to the leaves; it is possible to estimate the importance of a feature by computing the average depth at which it appears in the forest. RF allows us to have a quick understanding of which features actually matter, in particular if we need to perform feature selection. This remember us the ℓ_1 regulatization of SVM and LR model that shrinks to zero some coefficients.

2.3 Artificial Neural Networks

If we look at the architecture of the brain, we could inspired on how to build an intelligent machine. This is the idea that inspired ANNs. ANNs have gradually become different from their biological cousins. In this chapter, we will introduce ANNs, starting from the first ANN architecture. Then we will present MultiLayer Perceptron (MLP) and its evolution to DNNs.

2.3.1 Perceptron

The perceptron is a simple form of ANNs. It is based on a *Linear Threshold Unit* (LTU): the inputs and outputs are numbers and each input connection is associated with a weight. The LTU applies a step function to the weighted sum of its inputs:

$$Z = w_1x_1 + w_2x2 + \cdots + w_nx_n = W^T \cdot X),$$
$$h_w(X) = step(Z) = step(W^TX)$$

The most used step function in perceptron is the *Heaviside* function (sometimes the sign function is used).

LTU computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class otherwise it outputs the negative class (like the standard ML algorithms). Training a LTU means finding the right values for the weights.

Perceptrons are trained in this way: it is fed one training instance at a time, and for each instance it makes its predictions. For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.

$$w_{i,j}^{t+1} = w_{i,j} + \gamma(\hat{y}_j - y_j)x_i$$

where $w_{i,j}$ is the connection weight between i^{th} input neuron and the j^{th} output neuron,
x_i is the i^{th} input value of the current training instance,
$\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance,
y_j is the target output of the j^{th} output neuron for the current training instance,
γ is the learning rate.

Since the decision boundary of each unit is linear, perceptron can not learn complex patterns (like LR classifiers). If the training instances are linearly separable, Rosenblatt demonstrated that this algorithm would converge to a solution. This is called the *perceptron convergence theorem.*

2.3.2 Multilayer Perceptron

Now we will show the Mathematics under MLP with a single hidden layer. We deal with the general case where the number of classes is K.

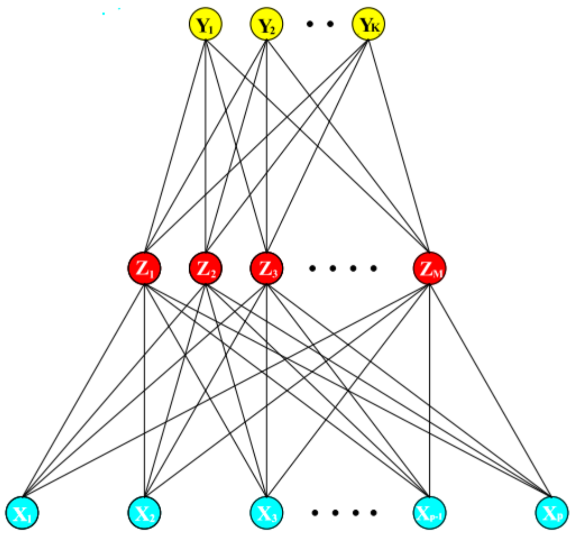

Figure 2.2. Schematic of a MLP with a singe hidden layer. This a feed-forward neural netowork

With reference to Figure 2.2, for K-class classification, there are K units at the top, with the kth unit representing the probability of class k. There are K target measurements $Y_k, k = 1, \cdots, K$, coded as a 0 - 1 variable. Z_m are derived features and are created from linear combinations of the inputs. The target Y_k is a function of linear combinations of the Z_m,

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \cdots, M,$$

$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, \cdots, K,$$

$$f_k(X) = g_k(T), k = 1, \cdots, K,$$

where $Z = (Z_1, Z_2, \cdots, Z_M), and T = (T_1, T_2, \cdots, T_K)$.

The activation function $\sigma(\nu)$ is usually the sigmoid $\sigma(\nu) = \frac{1}{1+e^{-\nu}}$. Sometimes Gaussian radial basis functions are used for the $\sigma(\nu)$.

Other two famous functions are: *hyperbolic tangent* function $tanh(\nu) = 2\sigma(2\nu) - 1$ and *ReLU* function. The first is like the logistic function, but its output value ranges from –1 to 1 (instead of 0 to 1), and this tends to make each output of layer more or less normalized (centered around 0). This helps speed up convergence. The second, $ReLU(\nu) = max(0, \nu)$, is continuous but not differentiable at $\nu = 0$. However, it works well and has the advantage of being fast to compute. Furthermore, the fact that it does not have a maximum output value also helps reduce some issues during gradient descent (it will be explained momentarily).

Neural networks are sometimes drawn with an additional *bias* unit in the hidden and output layers. The bias unit captures the intercepts α_{0m} and β_{0k} in the model. The output function $g_k(T)$ allows a final transformation of the vector of outputs T. One of most used function for the final transformation is the *softmax* function

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$$

This function outputs positive estimates that sum to one and it is used in the LR model. The units in the middle of the network, computing the derived features Z_m, are called *hidden units* because the values Z_m are not directly observed. In general there can be more than one hidden layer. Z_m are a basis expansion of the original inputs X as we have seen in SVM; the neural network become a standard linear model using these transformations as inputs. Here, the parameters of the basis functions are learned from the data. If σ is the identity function, then the model collapses to a linear model in the inputs. As a result, a neural network can be thought as a nonlinear generalization of the linear model. The rate of activation of the sigmoid depends on the norm of α_m , and if $||\alpha_m||$ is very small, the unit will operate in the *linear part* of its activation function.

Gradient descent. It is an iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, we make steps proportional to the negative of the gradient of the function at the current point. Instead, if we make steps proportional to the positive of the gradient, we approaches a local maximum of that function; the procedure is then known as *gradient ascent.*

Suppose we want to minimize the function $f(x_1, x_2, x_3) = x_1^2 - x_2 + x_3$ choosing as initial solution the vector $x_0 = [1,2,3]^T$. We have that $f(x_0) = 2$ and, in a circle of x_0 the function assumes different values similar to $f(x_0)$. These values suggest us that to locate points, near x_0, where the function assumes a value lower than $f(x_0)$, it is convenient to move along some directions. Furthermore, there are some preferential directions along which the function decreases more rapidly. The procedure can be iterated starting from a new point (near x_0) for which we got a value lower than $f(x_0)$, until reaching a minimum for f. This is a procedure that updates the solution in an iterative way based on information locally available and could locate a minimum point for the assigned function. If the function is not convex, the gradient descent technique can find a local minimum and not the global one.

It is like a human that try to reach the valley from the mountains in a dark night having only a torch.

Fitting the Multilayer Perceptron. The neural network model has unknown parameters, called *weights*, and we need to give them values that make the model fit the training data well. The set of weights are:

$$\{\alpha_{0m}, \alpha_m; m = 1, 2, \cdots, M\} \; M(p+1) \text{ weights,}$$
$$\{\beta_{0k}, \beta_k; k = 1, 2, \cdots K\} \; K(M+1) \text{ weights.}$$

For classification, we use sum of squared errors as our measure of fit (error function).

$$R(\theta) = -\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(x_i))^2$$

Typically we do not want to find the global minimum of $R(\theta)$, as this is likely to be an overfit solution. Instead some regularization is needed: this is achieved directly through a penalty term, or indirectly by *early stopping*. Details are given in the next section. The approach to minimizing $R(\theta)$ is by gradient descent, called $back-propagation$:

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, and let $z_i = (z_{1i}, z_{2i}, \cdots, z_{Mi})$. Then we have

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2,$$

with derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)z_{mi}$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il}.$$

Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$
$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},$$

where γ_r is the learning rate.

Overfitting. Neural networks have many weights and risk to overfit the data at the global minimum of R. An early stopping rule could be used to avoid overfitting, that means that we train the model before we arrive to the global minimum. We can exploit validation dataset to know when to stop, since we expect the validation error to start increasing. A method for regularization is *weight decay*. We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

and $\lambda \geq 0$ is a tuning parameter. Larger values of λ will tend to shrink the weights toward zero: typically cross-validation is used to estimate λ. There are other types of penalty, for example,

$$J(\theta) = \sum_{km} \frac{\beta_{km}^2}{1+\beta_{km}^2} + \sum_{ml} \frac{\alpha_{ml}^2}{1+\alpha_{ml}^2},$$

known as the *weight elimination* penalty. This shrink smaller weights more than the previous penalty.

This concludes the introduction to ANNs.

2.4 Preliminary tests on MNIST dataset

In this section, we will introduce the MNIST dataset, see some definition and see the results obtained by classification using some of the mentioned algorithms. In particular, we will see SGD classifier, KNN classifier, LR classifier, DT classifier, RF classifier and SVM classifier. We will understand which are the important performance measures used to compare more classifiers. In this paragraph, we propose a few preliminary tests on the MNIST dataset [16].

The MNIST database of handwritten digits has a training set of 60000 examples and a test set of 10000 examples. It is a subset of a larger set available from NIST[2]. This set has been studied a lot that it is often called the "Hello World" of

[2]https://www.nist.gov/

ML.

Some researchers have achieved near-human performance on the MNIST database, using some neural networks [17]; The highest error rate listed [16] on the original website of the database is 12 percent, which is achieved using a simple linear classifier with no preprocessing [18]. In 2013, an approach based on regularization of neural networks using DropConnect has been claimed to achieve a 0.21 percent error rate [19].

We have simplified the problem and have only tried to identify one digit, the number 5. This "5-detector" will be an example of a binary classifier, capable of distinguishing between just two classes, 5 and not-5. *Stochastic Gradient Descent* classifier, as the name tells, uses the concept of gradient descent and it relies on randomness during training.

There are many performance measures available.

Measuring accuracy using cross-validation Cross-validation is the process of partitioning data into subsets, performing the analysis on one subset (the training set), and validating the analysis on the other subset (the validation set or testing set). This process is repeated several times using different partitions as trainning set, and the validation results are averaged to estimate a final predictive model. Cross-validation is useful when there is not enough data available to partition it into separate training and test sets without losing significant modelling or testing capability. In particular in *k-fold* cross-validation, the data is randomly partitioned into k equal-sized subsamples. Of the k subsamples, a single subsample is used as test data, and the remaining $k-1$ subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples (folds) used exactly once as test data. When the order of the data is important, cross-validation might be problematic for time series models (our real case).

We have obtained above 95% of accuracy. Accuracy is not appropriate when we are dealing with *skewed* datasets (when some classes are much more frequent than others). A better way to evaluate the performance of a classifier is to look at the *confusion matrix*. The idea is to count the number of times instances of class A are classified as class B and viceversa.

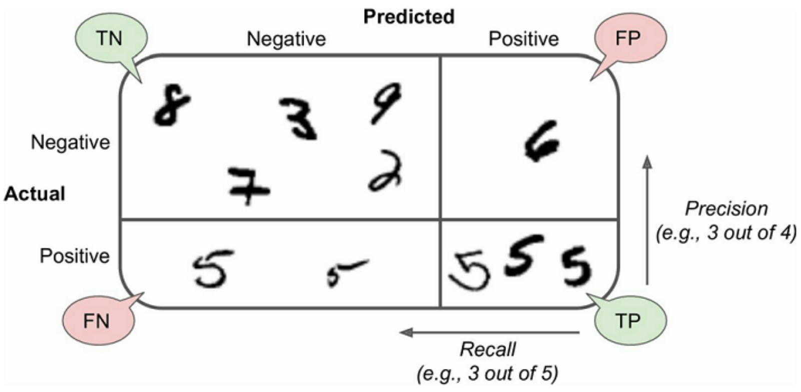

Figure 2.3. Confusion matrix showing the misclassifications

With reference to Figure 2.3, a perfect classifier has only *true positives* and *true negatives*, so its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right).

A measure derived from the confusion matrix is the accuracy of the positive predictions:

$$precision = \frac{TP}{TP+FP}$$

where TP is the number of true positives, and FP is the number of false positives (*false alarms*). Precision is used along with the *recall* metric, also called sensitivity or *true positive rate* (TPR): this is the ratio of positive instances that are correctly detected by the classifier

$$recall = \frac{TP}{TP+FN}$$

where FN is the number of false negatives.

The *F1 score* combines precision and it allows to compare classifiers. The classifier will get a high F1 score if both recall and precision are high.

$$F1_{score} = \frac{2}{\frac{1}{precision}+\frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision+recall} = \frac{TP}{TP+\frac{FN+FP}{2}}$$

In some contexts we care about precision, and in other contexts we care about recall.

For example, if we train a classifier to detect videos that are safe for kids, we prefer a classifier that rejects many good videos (low recall) but keeps only safe ones (high precision). Instead, if we train a classifier to detect thieves on surveillance images, we prefer a high recall; in this way, the security guards will get a few false alerts, but almost all thieves will get caught). Unfortunately, increasing precision reduces recall, and vice versa. This is called the *precision/recall tradeoff*.

The *receiver operating characteristic* (ROC) curve is similar to the precision/recall curve, but instead of plotting precision against recall, it plots the TP rate (recall) against the FP rate. The FPR is the ratio of negative instances that are incorrectly classified as positive. It is equal to 1 - TNR, which is the ratio of negative instances that are correctly classified as negative. The TNR is also called $specificity$. Hence the ROC curve plots *sensitivity* (recall versus 1 – specificity). Once again there is a tradeoff: the higher the recall (TPR), the more false positives (FPR) the classifier produce (because the precision is low). One way to compare classifiers is to measure the Area Under the Curve (AUC) . A perfect classifier will have a ROC AUC equal to 1, whereas a random classifier will have a ROC AUC equal to 0.5.

Table 2.1. Comparing between different algorithms on MNIST dataset using accuracy, training time and prediction time

Classifier	Accuracy (%)	Training time (sec)	Prediction time (s)
KNN	**99.24**	18.6	**775.8**
LR	97.72	7.26	0.04
Linear SVM	96.82	101.4	0.031
DT	97.76	**6.6**	0.031
RF	98.85	32.4	0.26

The algorithms was running trying several combinations for the parameters before trying the best combination to get the best score (accuracy).

As we see in Table 2.1, KNN has achievied the best accuracy but it have taken a long time to give us the predictions. This is due to how the algorithms internally predicts. We have shown at the beginning of this chapter that KNN computes the distance between the new instance and all the others in the dataset to decide the k neighbors of this instance. Finally it decides the class of the instance based on the majority vote. This work takes a lot of time and it is done in the prediction phase.

LR was very fast both in fitting the model and in predicting. It allows us to make a simple model with the help of regularization ℓ_1 that makes null some weights. Hence, the prediction was very fast in this case. Also SVM has the ℓ_1

regularization parameter. However, the training time was very high. DT was the best algorithm for this dataset as accuracy is comparable to that of KNN and LR and both training time and prediction time are low. RF, as it is an ensemble of DTs, has achieved a better result, but at the cost of an increase in training time because we need to train the model with a multitude of DTs and this takes time. These preliminary experiments showed us that KNN is definitely slower for large datasets. For this motivation, we will discard it in next experiments.

Chapter 3

Classification for Predictive Maintenance: a real industrial application

In this chapter, we present the main contribution of this thesis, that is the application of ML to a real industrial PC problem. After introducing the problem and illustrating the used dataset, we will present our approach, and finally show the obtained results.

3.1 A real appliation

Our case study is inserted into the MONSOON project [1]. Between its objectives, the MONSOON project aims to improve process industries by implementing new PC systems. The industries that will use their equipments as pilote sites to implement the techniques developed by MONSOON project are Aluminium Pechiney and GLNPlast.
This work thesis concentrates on the first one: specifically, we aim to develop a strategy to predict breakdowns of a Buss Mixer present in the Aluminium Pechiney industry.

[1]https://www.spire2030.eu/monsoon

The Buss Mixer [8] is a big equipement that mixes the various petrocarbon fractions very gently, a critical requirement in the aluminium industry. It is used for anode paste processing; it is extremely robust and made of special abrasion-resistant materials. The Buss Mixer often breaks and leads to high process cost. The undesired stops may involve the definitive failure of some components, which has relevant economic consequences. We want to predict the breakdowns to improve the process of aluminium production.

The experts in Aluminium Pechiney have evaluated an advance of 45 minutes would be optimal in order to allow the personnel to reach the mixer and perform the suitable intervention in order to prevent the breakdowns.

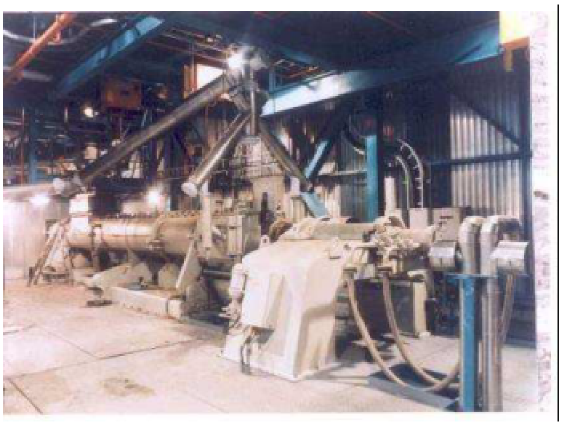

Figure 3.1. a BUSS Mixer

3.2 Introduction to the real application

The real industrial application that we consider in this thesis is a PC problem studied in the framework of the MONSOON project.

The Buss Mixer. An anode change happens every 32 hours on a given pot. An anode last 28 days. We want to apply PdM for paste plant equipments which means trying to predict future breakdowns. We need to predict the stopping of paste plant caused by degradation of anode quality and to identify their cause for avoiding them if possible.

The paste plant is made of different *chains*, each of them having a different role in the process. Each chain is composed of multiple equipments. For example, the Buss Mixer, in which we are interested, is the J160 equipment from chain J.

By looking at the list of stops/faults occurring in the paste plant, we can determine: which chain/equipment has the largest number of breakdowns and which chain/equipment has the largest cumulative duration of breakdowns.

Looking at the list of stops occurring in the paste plant, from September 2016 to February 2017, experts informe us that the stops are as below:

- the J, K and H chains account for 81,5% of the time lost to faults
- J: Mixer (52,2%)
- K: Vibrocompactor (21,3%)
- H: Heater (8%)

The chain J, in particular the Buss Mixer J160, is responsible for most of the breakdowns (occurrences and duration) and for this reason we want to optimize its maintenance.

Dataset. The dataset is contained in a CSV file. There is a header composed of a first column that is the timestamp of each acquired instance, 12 columns associated to the variables in which we are interested, such us maximum current intensity, minimum current intensity, mean current intensity of the Buss Mixer, temperature, motor speed etc. and a last column (label) which indicates if there was a breakdown or not at that timestamp. Label "1" means breakdown, while label "0" means that the equipment works as usual.

As we can understand from this explanation, the dataset is a time series. Each line is an instance taken with an acquisition frequency equal to 5 seconds. The considered period of time goes from the 1st September of 2016 to 8 June 2017. The CSV file has 4846680 lines and it is about 1 GB of space.

Our problem is particular because we do not want to predict the next value of the timeseries, but we need to know if a breakdown will occure in the next minutes, that is more complicated than the usual case where a data scientist try to predict the label for the current instance. We need to predict a breakdown some minutes before the breakdown itself.

3.2.1 MONSOON project

MOONSOON is a SPIRE project that aims to create an infrastucture in support of the process industries. By monitoring and controlling the processes, the goal of MONSOON is to identify a guided methodology from the output data, which can improve both the energy consuption and also the reuse of waste material. To test, validate and demonstrate the results of research, the project will use two real environments: an aluminium plant and a plastic factory.

The main focus of this project is enabling shared "Process Excellence Centers" to analyze large volume of available data from multiple shared pot-lines at supervision level.

Among the outcomes, MONSOON will provide early warnings or even predictive signals on global or individual anomalies in the aluminium plant, which is the topic of my thesis.

3.2.2 Aluminium smelting

Aluminium is produced following this reaction: $2Al_2O_3 + 3C + e^- \rightarrow 4Al + 3CO_2$ at the temparature of 960 °C. The process to produce the aluminium is done in an aluminium smelter. It consists of a large number of cells (pots) in which the electrolysis takes place. A typical smelter contains from 300 to 720 pots, each of which produces about a ton of aluminium a day. Smelting is run as a batch process, with the aluminium metal deposited at the bottom of the pots and periodically siphoned off.

The quality of anode affects technological, economical and environmental aspects of aluminium production. Energy efficiency is related to the nature of anode materials, as well as the porosity of baked anodes. Inhomogeneous anode quality due to the variation in raw materials and production parameters also affects its performance and the cell stability.

The anode is now defined as the electrode at which electrons leave the cell and oxidation occurs (indicated by a minus symbol, "-"), and the cathode as the electrode at which electrons enter the cell and reduction occurs (indicated by a plus symbol, "+"). Each electrode may become either the anode or the cathode depending on the direction of current through the cell.

3.2.3 Classification for prediction: the Predictive Horizon

As we have seen in the introduction, approaches to maintenance management can be grouped into three categories:

- *Run-to-failure*

- *Preventive maintenance*
- PdM

This last approch is the one we want to follow because it allows us to decrease the industial costs.

A possible approach to implement PdM (explained in paper [32]) and, as a result, prevent unexpected failures is to consider a different classification problem where, instead of only labeling the last iteration of a maintenance cycle as F (failure), we label as F the last m iterations. From a PdM perspective, this approach allows us to provide more conservative maintenance recommendations by choosing larger values for the *failure horizon* m. Moreover, by assigning more samples to the F class, we reduce the skewness of the dataset: Nm samples in class F and n - Nm samples in class NF (NF stays for "No Failure"). This can be repeated for k different values of the horizon m. The failure horizon m, called also the Predictive Horizon (PH), is in minutes in our case. Following this approach, when an instance is labeled as breakdown (1), we label as 1 also the measurmnet in the m minutes before it. In this way, we bring back in the past a situation that could occur in the furure. The ML algorithm learns this pattern and can predict if it will be a breakdown in the next m minutes.

We have taken this idea from the paper "Machine Learning for Predictive Maintenance: a Multiple Classifier Approach" [32]. In this paper, a new PdM methodology based on multiple classifiers is introduced for *integral type faults* (the most frequent in semiconductor manufacturing), a term which describes the failures that happen on a machine due to the accumulative effects of usage and stress on equipment parts. The paper compares then Multiple Classifier PdM (MC Pdm) with the Predictive Maintenance (PvM) methodology for a benchmark semiconductor manufactoring maintenance problem, namely, changing of filaments in ion implantation tools. The dataset is like ours: some iterations are faulty and others are not faulty. Two metrics are defined, namely:

- Frequency of Unexpected Breaks (ϱ_{UB}) - percentage of failures not prevented;
- Amount of Unexploited Lifetime (ϱ_{UL}) - average number of process iterations that could have been run before failure if the preventative maintenance suggested by the maintenance management module had not been performed.

Different costs, c_{UB} and c_{UL}, can be associated with ϱ_{UB} and ϱ_{UL}; it is possible minimize the total operating costs, as defined by the weighted sum: $J = \varrho_{UB}c_{UB} + \varrho_{UL}c_{UL}$. In the presented approach k classifiers work in parallel (in our customized apporach, we have used one ML algorithm at a time and repeated the experiments for k different value of PH) and a maintenance event is triggered by the decision making logic based on an operating costs minimization philosophy. Performance of the MC PdM methodology increases with the number of classifiers, k, since each classifier provides more information on the health status of the process. In our case, k is the number of different value of PH. In the paper, it has been considered two classification techniques, namely: SVM and KNN. It can be appreciated how PdM and MC PdM outperform PvM approaches for both metrics.

3.2.4 Used tools

Python. Python is simple, consistent, and math-like. The code is readable pseudocode. It comes with a huge amount of inbuilt libraries (mainly libraries for ML). Some of the libraries are TF, Scikit-learn, Pandas, Numpy (that we have also used). We can get into ML without knowing very well Python, differently from the other programming languages. Furthermore, the time spent on debugging code in python is way less when compared to other compiled languages.

Scikit-learn. Scikit-learn is a ML library for Python. It implements various classification, regression and clustering algorithms including SVM, RF, gradient boosting (technique used to enhance performance of DT), k-means and many others, and is designed to interoperate with NumPy library. It allows also dimensionality reduction, model selection and data preprocessing. Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance.

Pandas. Pandas is a Python library providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the majority of use cases in finance, statistics, social science, and many areas of engineering. Pandas is built on top of NumPy.

Numpy. NumPy (Numeric Python) is an extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices.

3.2.5 Variable selection

RF, SVM and LR give us an idea about the important features of the dataset. They can perform a feature selection, a way to highlight the most important features of the dataset. Such non-relevant features can then be neglected, which provides a more parsimonious model. Industries, often, put a lot of sensors around, because of their cheapness, that acquire measurements of many variables, without bein sure that these variables are connected to the actual purpose. For this reason, it is also important to make feature selection in the industry domain. In this way,we have few features and the model computed by the algorithm is simplier and faster to compute. We will see some results about this. SVM uses the ℓ_1 regularization to obtain a "sparse solution", that is to nullify some weights of features [27], while RF gives us an index of the importance of a feature based on its position in the "forest". The node (feature) located near the root are more important than others. SVM can use also the ℓ_2 regularization but it does not null weights as ℓ_1 regularization does. We can combine them and use the *elastic regularization*, that means that we use both of them but giving them a weight. In this manner, we adjust the model to fit well the data and null some weights, hence remove some features.

If we look to the "decision forest" created by RF algorithm, important features are likely to appear closer to the root of the tree, while unimportant features will often appear closer to the leaves (or not at all). It is therefore possible to get an estimate of the importance of a feature by computing the average depth at which it appears across all trees in the forest. RF give us a quick understanding of what features actually matter, in particular if we need to perform feature selection.

DT gives us an idea for the most relevant features, but RF, as it uses many DTs, gives us a better idea about relevant and irrelevant features.

Mathematically, the ℓ_1 regularization is a term added to the equation of the model that can be manipulated through a regularization parameter called λ.

3.3 Experiments

In this section, we present the experiments performed to predict the breakdowns of the Buss Mixer (introduced in Section 3.1)

We have divided the dataset into a training set and a test set, in a cronologically manner. This means that the first part was used for train the algorithms, while the last part was used for testing them. The training set is 70% of the whole dataset and the test set is the last remaining 30%. The algorithms used was LR, DT, RF, SVM and MLP. We have chosen these algorithms because, for an intial approach to ML, they are different between each other and some of them are used very often in practice. Furthermore, they are easily to understand. RF and SVM are very popular and perform well in many cases. MLP was used to compare a

small neural network to the traditional ML algorithms. In the next chapter, we will see the results obtained by using DNN with TF.
Let us take a look to the dataset.

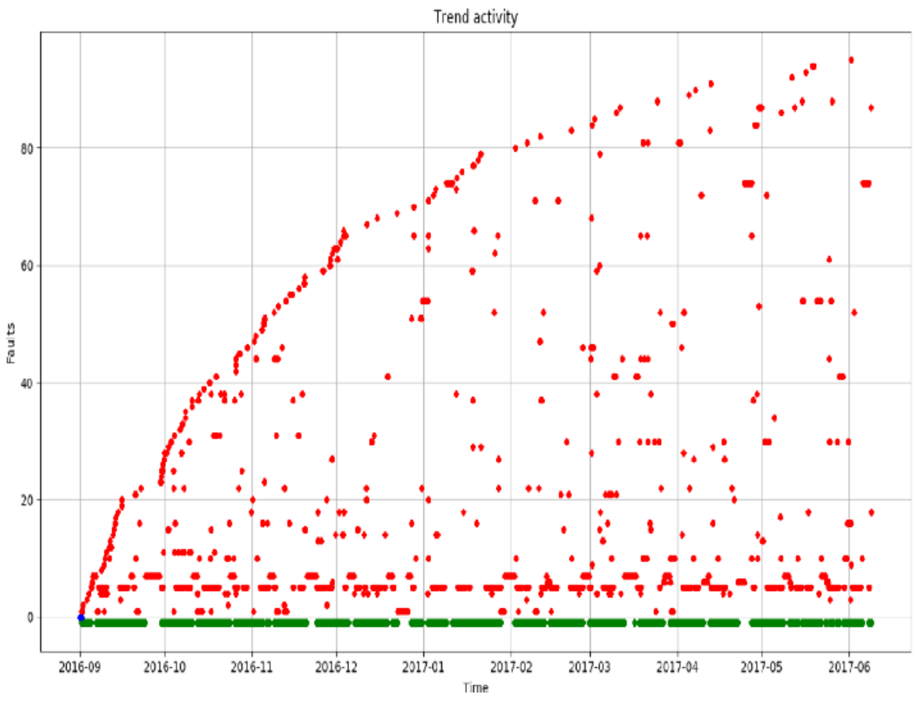

Figure 3.2. In the dataset, different faults are labeled with different natural numbers in the range [0, ...] (as shown on the y-axis). We label by -1 (and green color) the instants of normal activity.

In Figure 3.2, we have a graphical representation of the dataset. As we see, we have many faults. These faults are not only the faults of the Buss Mixer discussed previously but include also the faults of the other chains. The green points at level -1 indicate normal (non-faulty) activity. The initial nearly 9000 value have fault 0 that is not documented, so we have removed them later from the dataset; they are not so many against about 4800000 instances, hence they will not affect predictions at all. All the other points are faults relatives to the range of chains we have previously talked about.

Since we do not want to predict also the type of faults, we consider all these faults as faults of one type only and try to predict the breakdowns (faults) using the different ML algorithms mentioned. Let us start with DT.

3.3.1 Different Predictive Horizons

After the ML algorithms has decided the label for the new instances, we predict in this way: we launch the "alarm" when we encounter the first 1 (failure) in the new instances and then we compute the temporal distance between the first 1 in the test set and the 1 that we have encountered (Another approach is to launch the alarm after encountering in series a certain number of 1s). The idea has been taken by a scientific article ([32]).

The industrial experts evaluated that 45 minutes would be the optimal advance to perform an intervention and prevent breakdown. We have tried different PHs: 5 minutes, 20 minutes, 45 minutes and also 60 minutes, to see how the algorithms behave in different situations.

Table 3.1. Results for predictions with DT at different PHs

Prediction Horizon(min)	Dataset balancement(%)	Accuracy (%)	Faults	Detected faults	False alarms
5	98.47	99.03	148	99	80
20	94.54	95.78	134	91	17
45	**88.73**	**90.74**	**127**	**88**	**18**
60	85.44	87.84	123	86	17

Precision(%)	Recall(%)	AUC(%) tabular	Training time(sec)
69.03	55.80	77.72	1.74
82.88	21.75	60.75	1.54
88.20	**13.39**	**56.59**	**1.58**
90.95	11.72	55.77	1.61

In Table 3.1, we show the results. Dataset balancement is the percentage of instances labelled as -1 against instances labeled as 1, and we can see clearly that the dataset is always unbalanced. For a PH of 45 minutes, which we are interested in, the accuracy is 2% above the dataset balancement and this is good becasue it tells us that the algorithm actually does not classify randomly. The number of false alarms is low. The training time is about 1 second and a half. The number of faults decreases because there are some overlapping of "1" safter the introduction of the preditive horizon. In the next figures we see that we are able to predict the breakdowns 10 minutes of advance. These figures shows minutes in the x-axis and number of predicted faults in the y-axis. Each bar tell us how many faults has been detected and how much time before.

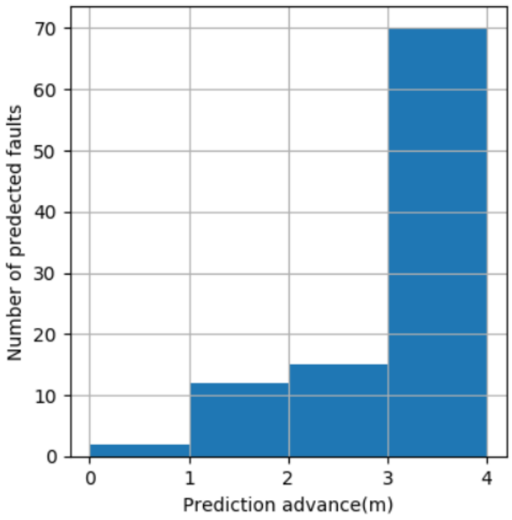

Figure 3.3. Number of predicted faults by DT for PH equal to 5 minutes.

In Figure 3.3, we see that the majority of faults detected (70 faults) was detected 4 minutes before.

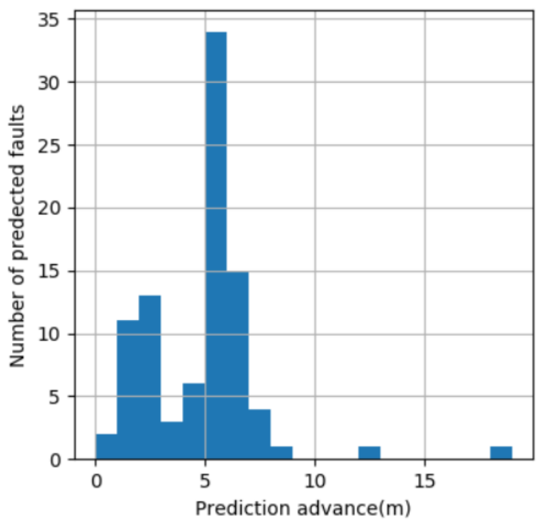

Figure 3.4. Number of predicted faults by DT for PH equal to 20 minutes.

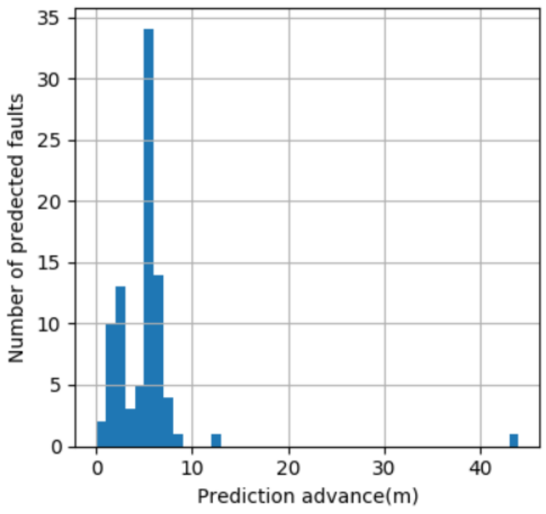

Figure 3.5. Number of predicted faults by DT for PH equal to 45 minutes.

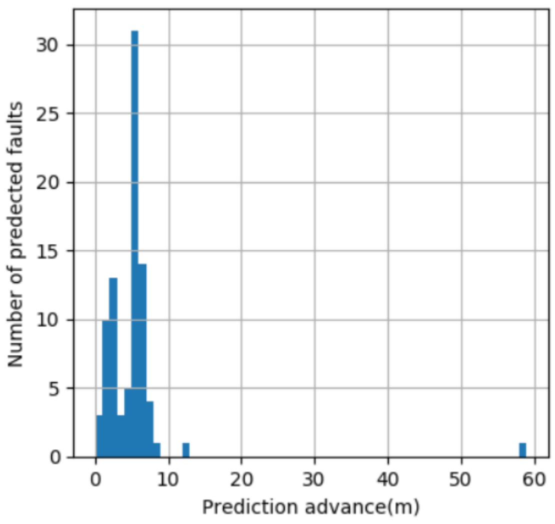

Figure 3.6. Number of predicted faults by DT for PH equal to 60 minutes.

A part the Figure 3.3 (5 minutes of PH), the other figures show us clearly that for most times, we were able to predict the faults 10 minutes before the breakdown and sometimes we were able to predict some faults 45 minutes before (see the Figures). Even with PH set to 60 minutes, we were not able to surpass 10 minutes. Now let us see the results obtained by using RF.

Table 3.2. Results for predictions with RF at different PHs

Prediction Horizon(min)	Dataset balancement(%)	Accuracy (%)	Faults	Detected faults	False alarms
5	98.47	99.02	148	97	71
20	94.54	95.75	134	91	15
45	**88.73**	**90.72**	**127**	**88**	**16**
60	85.44	87.81	123	86	15

In Table 3.2, we clearly see that there is no so much difference between using DT classifier or RF classifier, since they give us the same results. The only difference is the training time which is higher for RF. This is obvious since RF use a higher number of DT estimators. The traning time is not so interesting for our case.

Remember that RF gives us an idea of the relevant features, but we think

Precision(%)	Recall(%)	AUC(%)	Training time(sec)
69.49	52.57	77.93	5.75
82.90	20.95	60.95	6.02
88.39	**13.05**	**56.44**	**6.92**
91.13	11.45	55.88	5.29

that SVM gives us this idea in a clearer way since it shrinks to zero the weights of irrelevant features and we can see directly the coefficients of the model. RF does not allow us to see them. Momentarily, we will also see the results obtained by applying the LR classifier that uses ℓ_1 and ℓ_2 regularization like SVM.

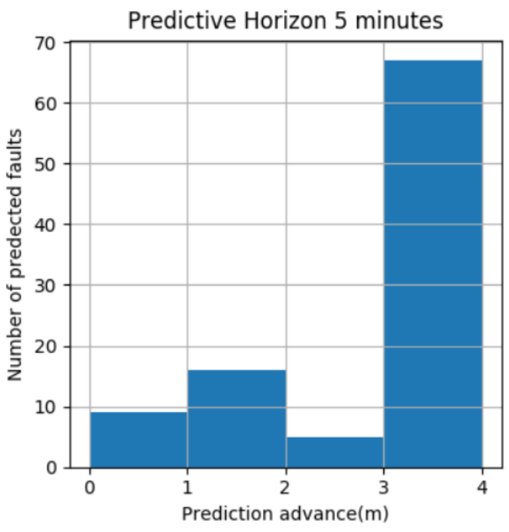

Figure 3.7. Number of predicted faults by RF for PH equal to 5 minutes.

Also using RF with a PH of 5 minutes we have obtained the same information obtained with DT.

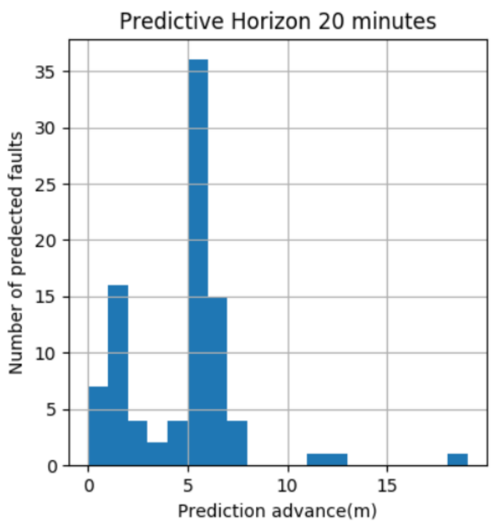

Figure 3.8. Number of predicted faults by RF for PH equal to 20 minutes.

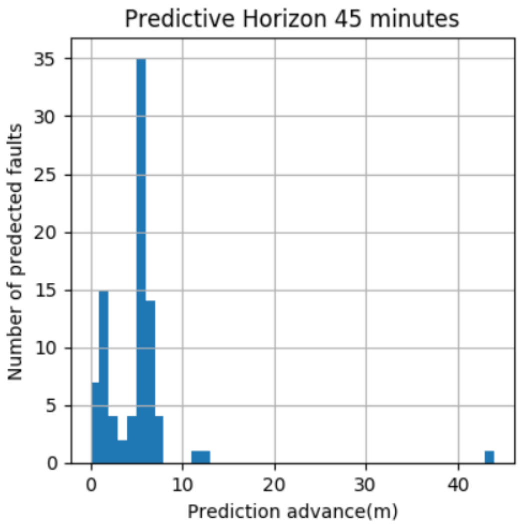

Figure 3.9. Number of predicted faults by RF for PH equal to 45 minutes.

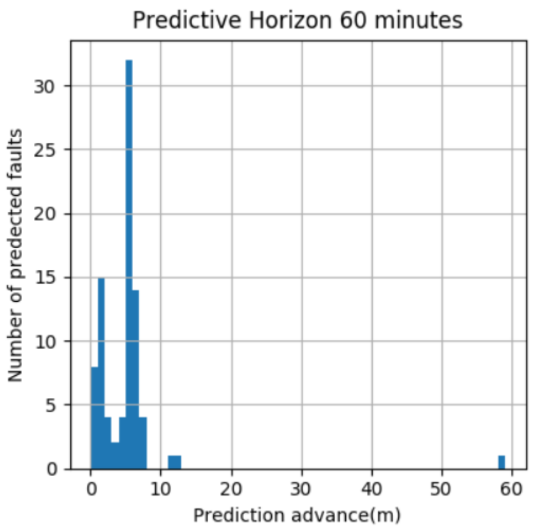

Figure 3.10. Number of predicted faults by RF for PH equal to 60 minutes.

Again, also with RF classifier, we were only able to classify within 10 minutes instead of 45 minutes.

Table 3.3. Results for predictions with LR at different PHs

Prediction Horizon(min)	Dataset balancement(%)	Accuracy (%)	Faults	Detected faults	False alarms	Precision(%)	Recall(%)	AUC(%)	Training time(sec)
5	98.47	99.14	148	85	68	90.38	42.26	81.03	15.47
20	94.54	95.71	134	89	18	82.29	20.25	64.19	18.35
45	**88.73**	**90.69**	**127**	**86**	**15**	**88.04**	**12.77**	**57.74**	**21.44**
60	85.44	87.79	123	83	15	90.91	11.28	56.24	17.06

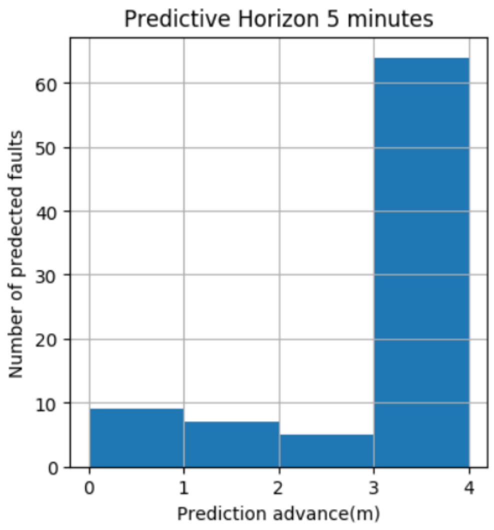

Figure 3.11. Number of predicted faults by LR for PH equal to 5 minutes.

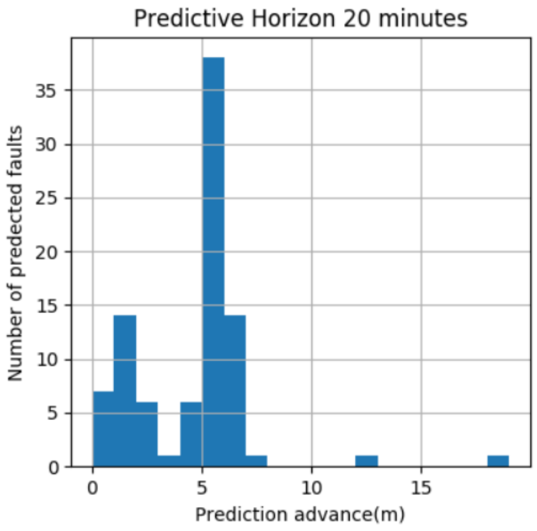

Figure 3.12. Number of predicted faults by LR for PH equal to 20 minutes.

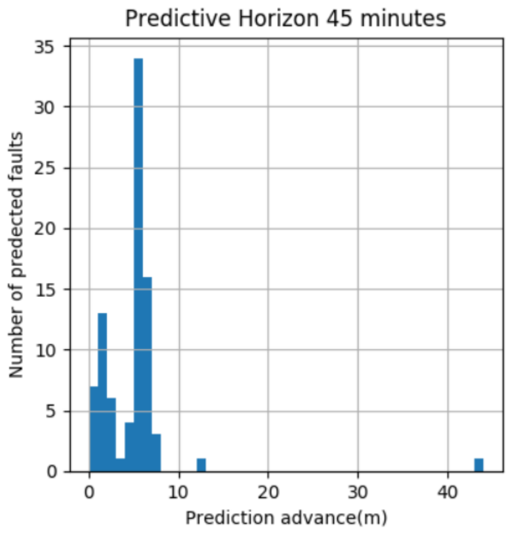

Figure 3.13. Number of predicted faults by LR for PH equal to 45 minutes.

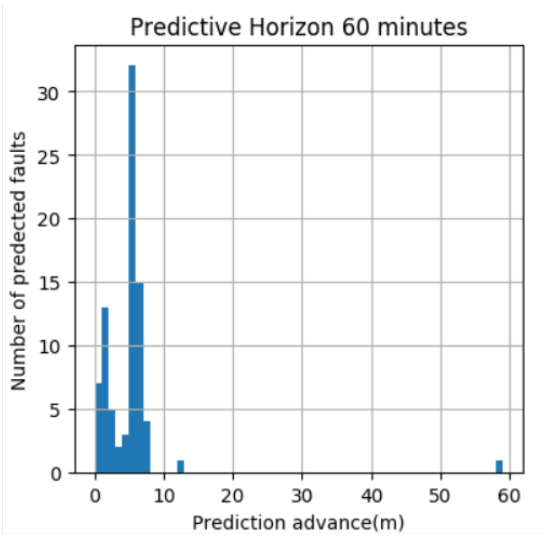

Figure 3.14. Number of predicted faults by LR for PH equal to 60 minutes.

Also LR behaves in the same way as the previous classifiers. Remember that

we are interested in predicting breakdowns 45 minutes before, but again we have been able to predict the breakdowns only 10 minutes before.
Under the Table 3.4 below, we find the results obtained by running the SVM classifier.

Table 3.4. Results for predictions with LR at different PHs

Prediction Horizon(min)	Dataset balancement(%)	Accuracy (%)	Faults	Detected faults	Faults alarms	Precision(%)	Recall(%)	AUC(%)	Training time(sec)
5	98.47	98.53	148	37	25	40.37	12.61	82.42	47.08
20	94.54	95.04	134	38	14	61.58	6.89	61.54	81.12
45	**88.73**	**90.69**	**127**	**87**	**13**	**88.25**	**12.77**	**55.82**	**11.39**
60	85.44	87.80	123	84	16	90.94	11.41	54.60	21.23

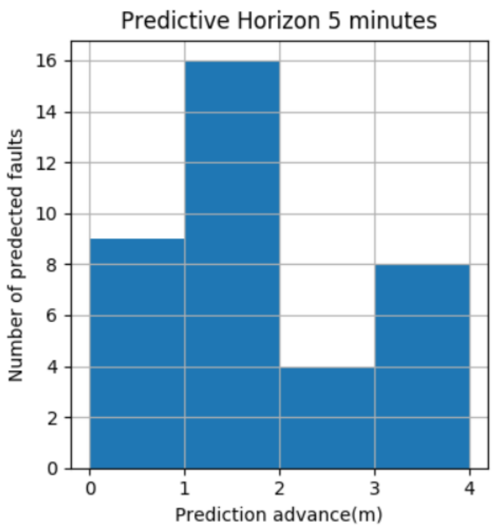

Figure 3.15. Number of predicted faults by SVM for PH equal to 5 minutes.

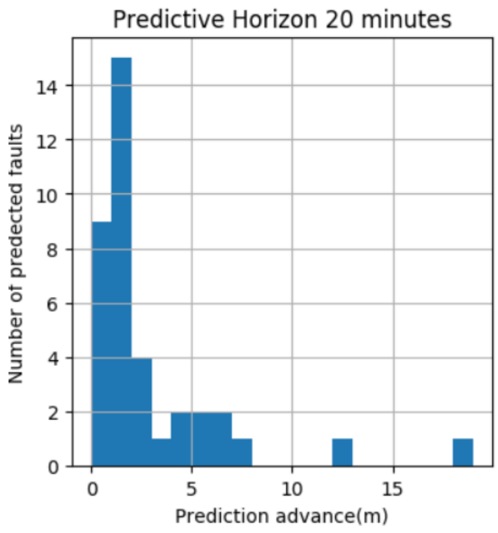

Figure 3.16. Number of predicted faults by SVM for PH equal to 20 minutes.

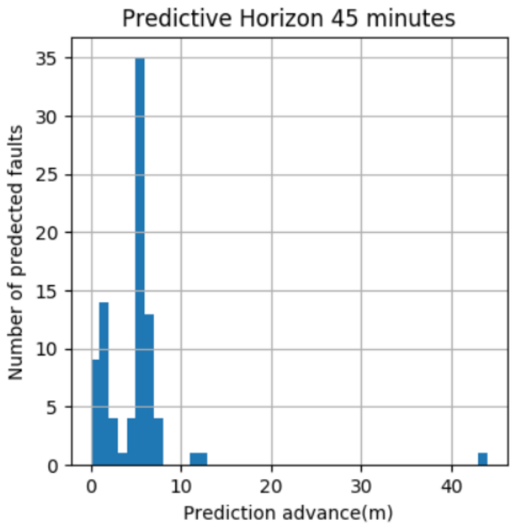

Figure 3.17. Number of predicted faults by SVM for PH equal to 45 minutes.

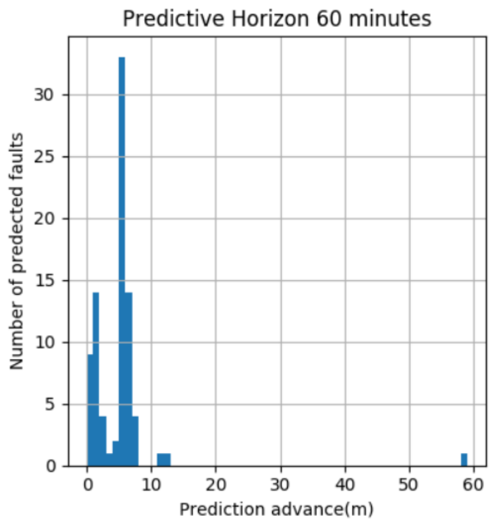

Figure 3.18. Number of predicted faults by SVM for PH equal to 60 minutes.

We have decided to use a PH of 60 minutes to see if we could obtain better results, but as we can see from the figures associated with a PH of 60 minutes, we were able only to predict within 10 minutes.

Now, let us try a shallow neural network (a MLP with one hidden layer and two units) to see if neural networks performss better than standard ML algorithms since ANNs are able to "generate" every type of non-linear function and as a result, they are able to face a lot of classification problem. The problem is that, in general, the training time is greater than the training time of traditional ML algorithms. In this case, since we have a small neural network, the training time is low.

Table 3.5. Results for predictions with neural network (one hidden layer and 2 units) at different PHs

Prediction Horizon(min)	Dataset balancement(%)	Accuracy (%)	Faults	Detected faults	False alarms
5	98.47	99.00	148	97	86
20	94.54	95.75	134	91	17
45	**88.73**	**90.72**	**127**	**88**	**17**
60	85.44	87.82	123	86	18

Precision(%)	Recall(%)	AUC(%)	Training time(sec)
67.94	53.32	78.06	12
82.74	20.91	60.98	31.35
88.14	**13.11**	**57.02**	**25.47**
90.92	11.55	55.72	20.05

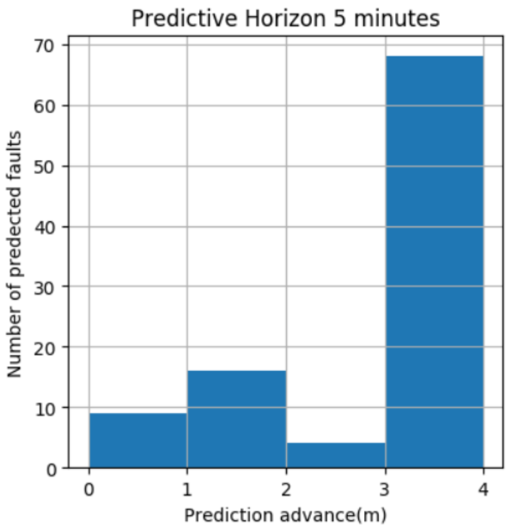

Figure 3.19. Number of predicted faults by MLP for PH equal to 5 minutes.

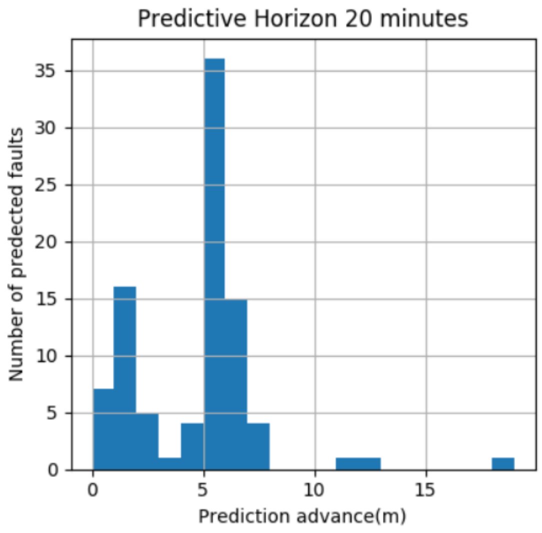

Figure 3.20. Number of predicted faults by MLP for PH equal to 20 minutes.

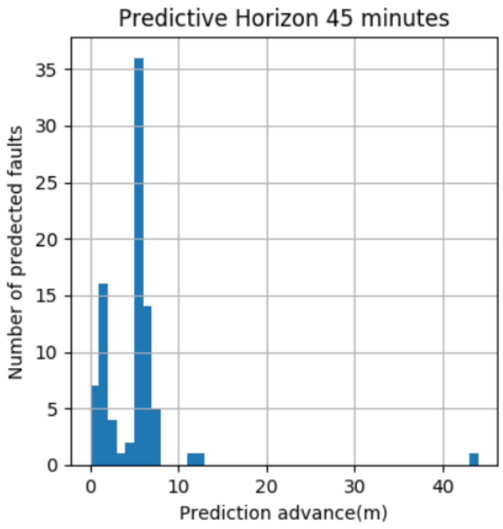

Figure 3.21. Number of predicted faults by MLP for PH equal to 45 minutes.

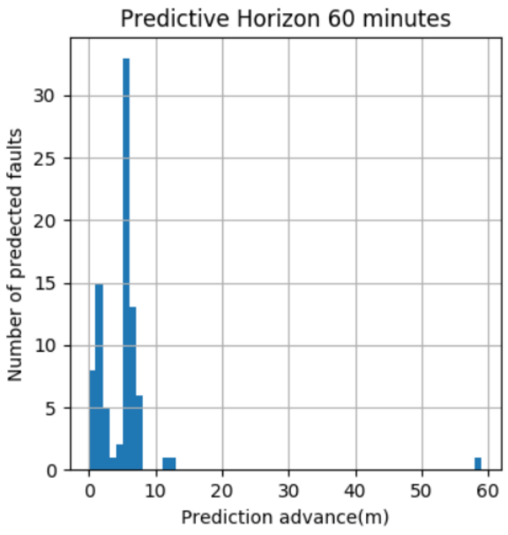

Figure 3.22. Number of predicted faults by MLP for PH equal to 60 minutes.

We can observe that also with neural networks we obtain prediction within 10 minutes of advance, similarly to previous techniques.

Let us aggregate these results in one table for PH equal to 45 minutes to compare the ML algorithms and see which one has the best performances for our use case. The dataset balancement is 88.73 %, that means that the label -1 (no fault) is noumerous with respect to the label 1 (fault). The number of detected faults is 127.

Table 3.6. Results of classifiers for PH equal to 45 minutes where Clas. = Classifiers, Acc. = Accuracy, Prec. = Precision, Rec. = Recall, Det. faults = Detected faults, False al. = False alarms, Tr. time = Training time

Clas.	Acc, (%)	Prec.(%)	Rec.(%)	AUC(%)	Det. faults	False al.	Tr. time(sec)
DT	90.74	88.20	13.39	56.59	88	18	1.58
RF	90.72	88.39	13.05	56.44	88	16	6.92
LR	90.69	88.04	12.77	57.74	86	15	21.4
SVM	**90.69**	**88.25**	**12.77**	**55.82**	**87**	**13**	**11.39**
MLP	90.72	88.14	13.11	57.02	88	17	25.47

We will focus only on the PH of 45 minutes from now on. We are interested in having a high accuracy with a high number of faults detected and few false alarms. SVM seems to be a good choice: The accuracy is similar to the other classifiers; there is no difference in the number of detected faults (88 against 87 or 86) and the number of false alarms is the lowest. Furthermore, the model computed by SVM using ℓ_1 regularization is simplier and faster because it shrinks to zero the majority of weights. We will see momentarily the coefficients of the model computed by SVM.

3.3.2 Position of faults and predictions

Having a high accuracy is not sufficient to claim that a certain classifier is the best; We need to know the position of the faults and the predictions, i.e launching the alarm when the classification pass from 0 to 1, made by the algorithms. In this paragraph, we only have some figures in which we show some situations that we have observed during esperiments.

In Figure 3.23, we show the position of faults, false alarms and predictions made by DT classifier. The other classifiers give us the same result or a very similar result.

Figures 3.23, 3.24, 3.25, 3.26, 3.27 show us the interval before a faults and tell us if we have predicted well the faults. We can also see the false alarms in Figure 3.29.

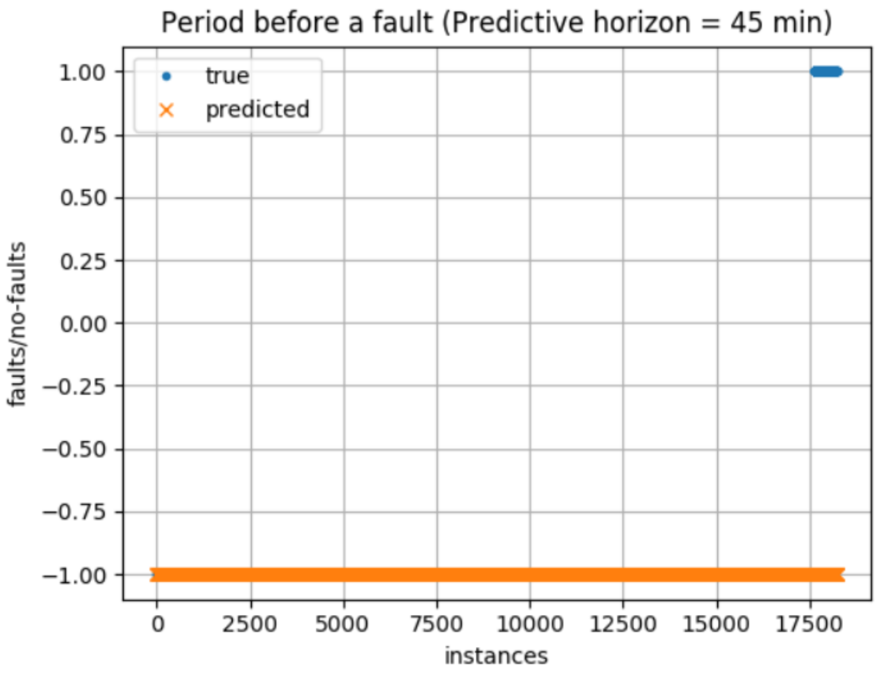

Figure 3.23. This faults was not predicted

We have plotted the labels obtained by running the classifier and the labels of the test set to see if they match. In this way, we can see if the classfier has predicted well, if there was a false alarm or how much before the classifier has predicted the fault. In the x-axis, we have the instances, while in the y-axix we have the predictions made the classifiers and the label of the test set. In this case, the classifier has not predicted the fault at all: in fact, after the 17500th instance, we have a series of faults (in blue), but the classifier has labeled them with label -1 and this is clearly a mismatch.

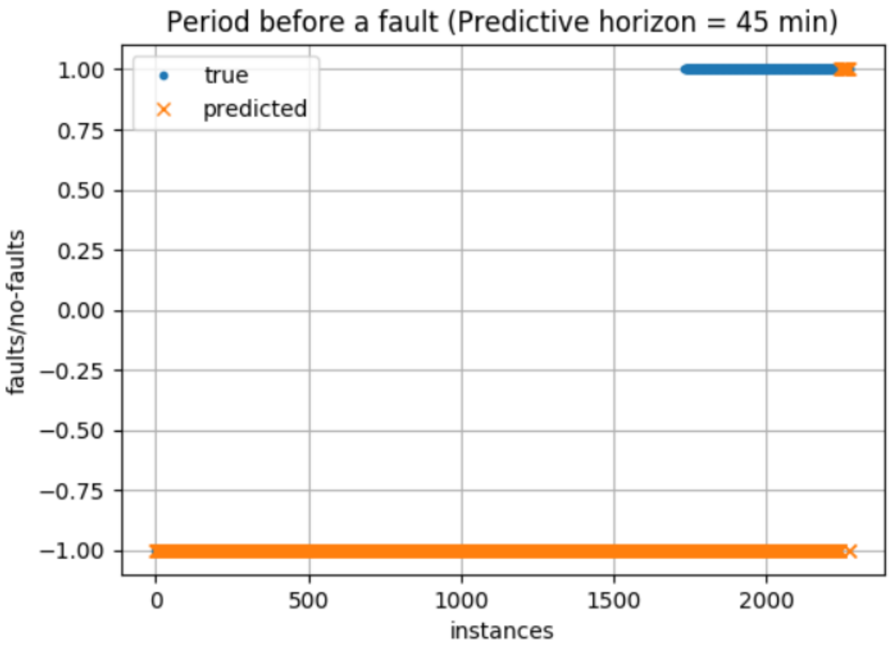

Figure 3.24. Fault not predicted 45 minutes before

In Figure 3.24, the fault was predicted well, but only at the end of the interval of time. This matches with the histogram we have shown before in which we have seen that the classifier is able to predict only 10 minutes before in the most cases.

The predictions were right for different instances at the end of the interval of breakdowns, but it is enough for us only one right prediction within this interval to tell that this breakdown (a sequence of labels 1) was predicted.

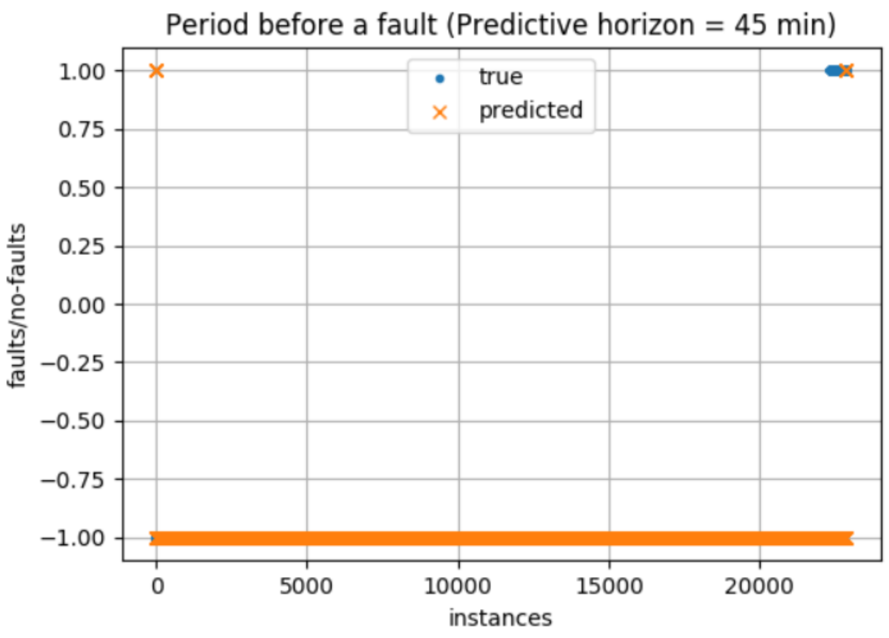

Figure 3.25. Fault predicted at the end of 45 minutes preceded with a false alarm

In Figure 3.25, the fault was predicted, but a false alarm preceded it. Looking closely, this false alarm is not important as the series of breakdowns is far from it.

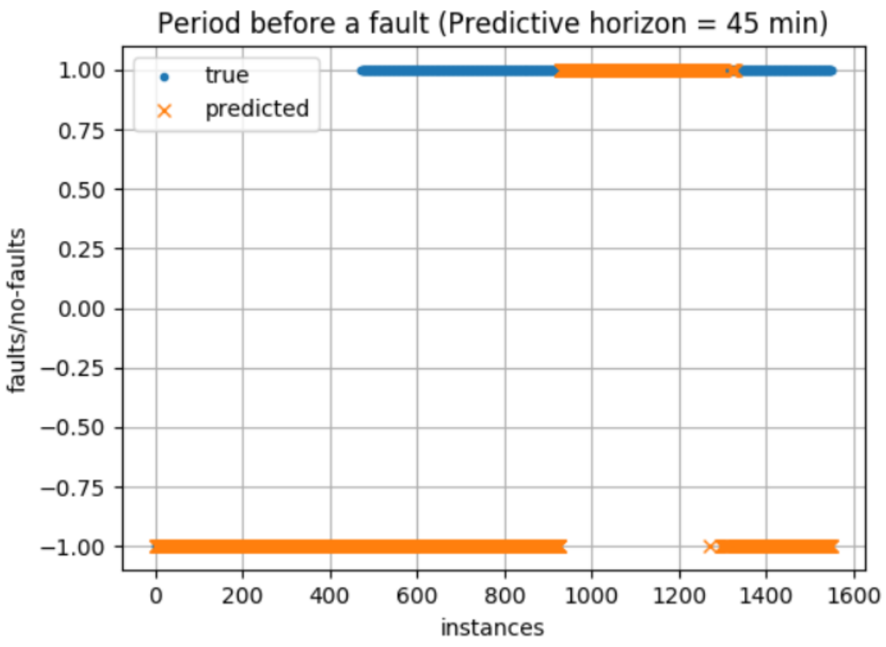

Figure 3.26. Fault predicted with advance

Figure 3.26 shows a fault that was predicted with a little of advance with respect to the 45 minutes. As said, the first prediction, made at the 900th instance, is sufficient to tell that the fault was predicted. To know how much time before we have predicted the fault, we need to start from this instance to the last instance of the blue line. Passing from an instance to the other is 5 seconds.

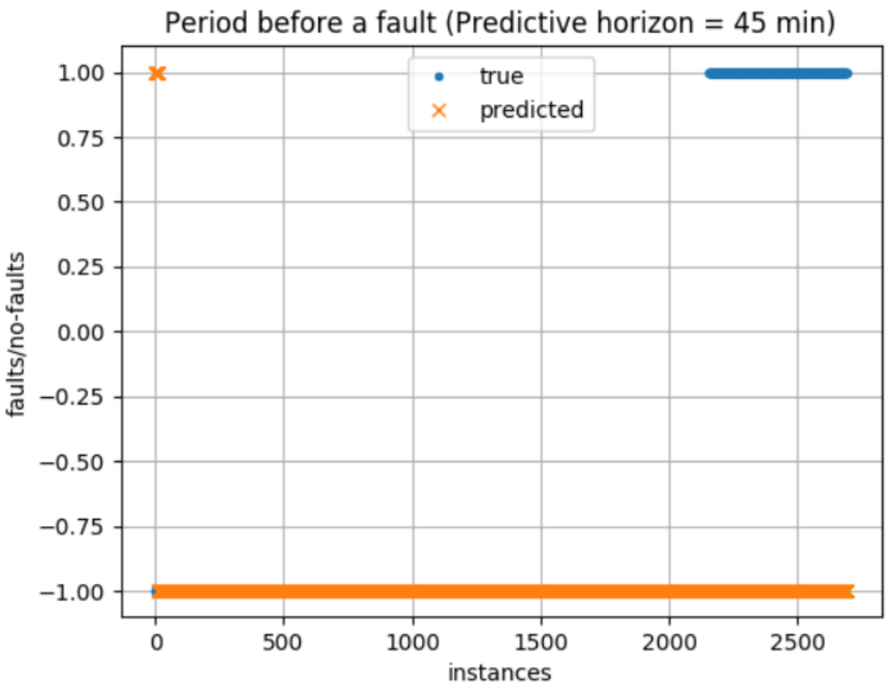

Figure 3.27. A clear false alarm

Figure 3.27 is a false alarm. Fortunately, false alarms are few with respect to the detected faults.

Let us have a look to the position of detected faults and false alarms for the different classifier.

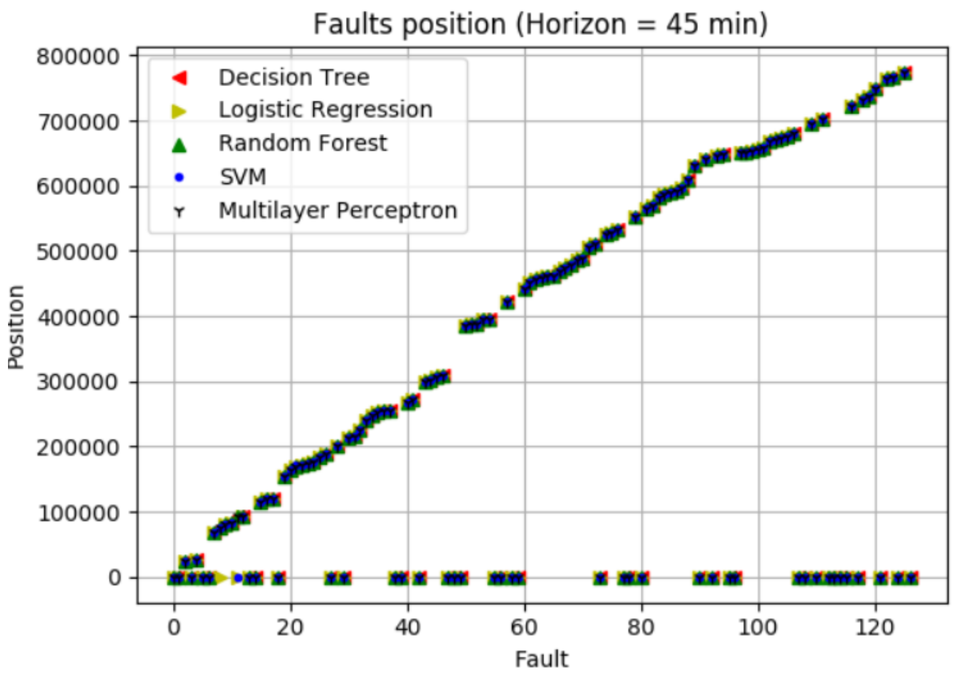

Figure 3.28. Positions of faults detected by classifiers for PH of 45 minutes

Figure 3.28 shows the positions of the detected faults by each classifier. In the x-axis, we see the faults, while in the y-axis we have the position in the dataset. Over the 127 faults, none of the classifiers have detected all of them. The majority of the classifiers have detected 88. The faults not detected was labeled with the value 0.

All classifiers detect 88 faults over the 127 present in the dataset, except SVM classifier that detects 87 faults and LR classifier that detects 86 faults. We can see this in the graph: already at the beginning, SVM classifier misses a fault and LR classifier misses two faults (see the blue dot associated to SVM classifier and the two yellow right-arrow associated to LR classifier)

One thing not visible in the graph is that in most cases, DT classfier and RF classifier detect the faults from 5 second to 50 seconds before the other classifiers.

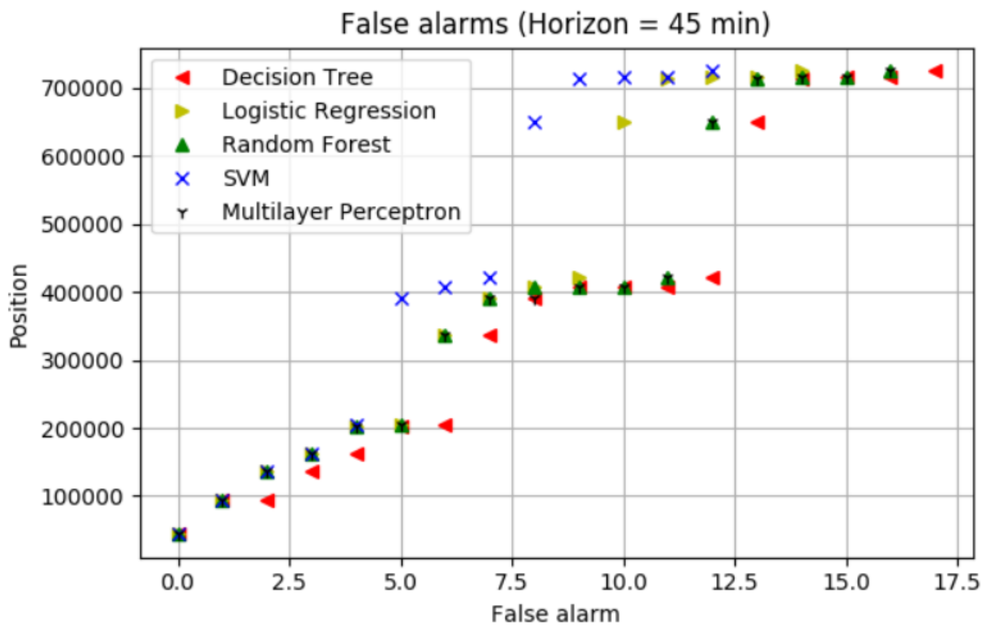

Figure 3.29. Positions of false alarms detected by classifiers

Here instead, we see false alarms detected by classifiers. On average, the number of false alarms is 15 and they are almost all detected in the same position. The classifiers who have detected a higher number of false alarms, in reality, they have detected a unique false alarm because they are too close.

We can conclude after this detailed analysis that the classifiers behave in the same way in front of this dataset even if they are different and they use different approaches to classify. We have seen that there is only one important feature and this can tell us that we have a lot or irrelevant features.

3.3.3 ℓ_1 regularization

Using the ℓ_1 regularization of the SVM classifier, we have found an interesting result. These are the coefficients of the model computed by SVM classifier:

$$\beta = [-0.0014, \mathbf{-0.15}, -0.011, -0.0013, 0, 0, 0.0032, 0.015, -0.00088, 0.0071, -0.046, 0]$$

We notice immediately that the second coefficient -0.15 (associated with maximum current intensity of the Buss Mixer) has an order of magnitude greater than other coefficients. We therefore ask ourselves what happens if we put all the coefficients to zero except -0.15. We have removed all the features from the feature space and used only this feature to see if we would obtain the same result that we have obtained considering all the 12 features. The result is shown in Figure 3.30.

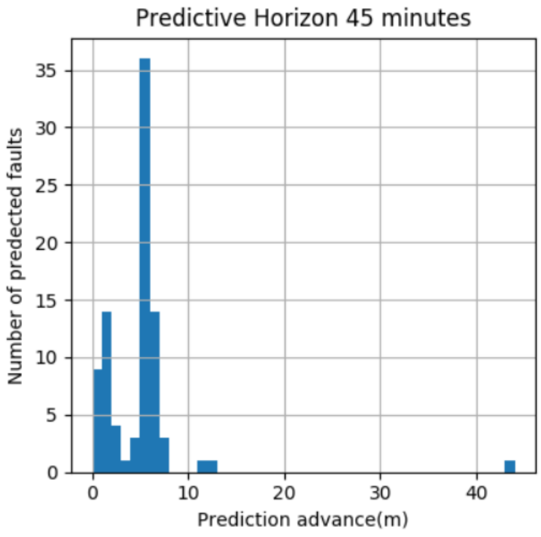

Figure 3.30. Number of predicted faults by SVM for PH equal to 45 minutes using only the most important feature

As we see, this plot that shows that the SVM classfier predict the faults only 10 minutes before is similar to the plot that we have seen in the Figure 3.17

The traning time is lower (about 1 second) because the feature space is very small now. The performance measures are: accuracy is 90.70 %, precision is 88.08 %, recall is 12.93 % and AUC is 57.09 %. Over the 127 faults, the classifier has detected 87 as expected and 16 false alarms, namely 3 more false alarms with respect to the previous case. This confirms that there is only one relevant feature; hence, the other features could be dropped because they have not given us useful information.

Let us try to add some "derived" features to see if we are able to enhance the results got until now.

3.3.4 Adding means and variances to the feature space

For each feature, we have added two other features: the variance and the mean. These variancces and means are computed over a windows of n instances. Than we have slided this window to the next instance and have computed the variance and the mean over the n previous instances. For the first n instances of the dataset, the mean is the value itself of the variable and the variance is zero. Totally, the dataset has 36 features. Let us see the result for the different classifiers.

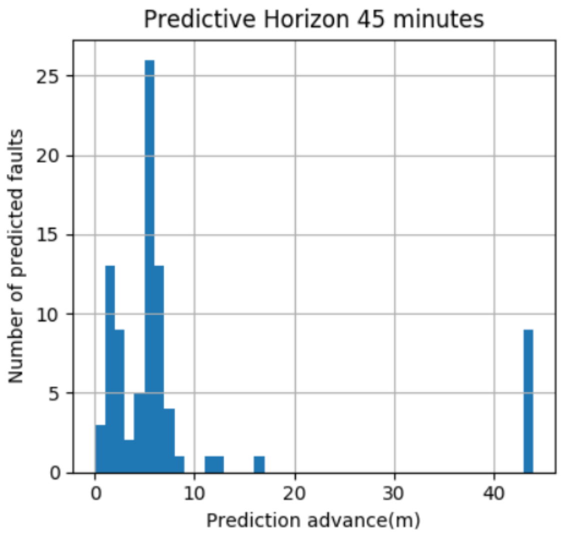

Figure 3.31. Number of predicted faults by DT for PH equal to 45 minutes adding means and variances to the initial 12 variables of the dataset

With respect to Figure 3.5, we note that DT algorithm has predicted many times the breakdowns 6 minutes before. Also the number of breakdowns predicted 45 minutes before has increased. Again, we notice two breakdowns predicted with 11 and 12 minutes of advance.

Even if the general trend is similar to the previous figures (the majority of breakdowns was predicted nearly 10 minutes of advance), here we pass from 1 to 9 predictions with an avance of 45 minutes.

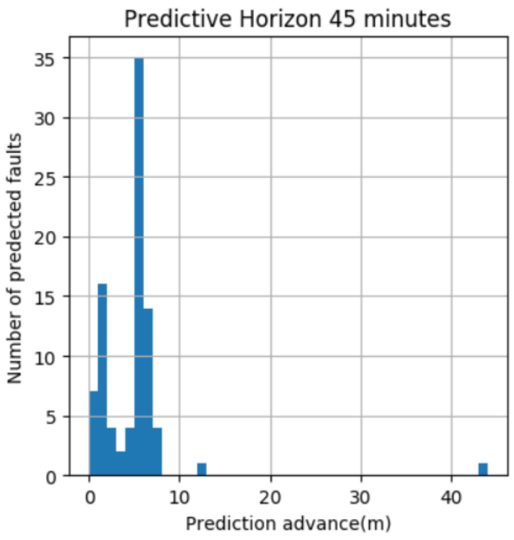

Figure 3.32. Number of predicted faults by RF for PH equal to 45 minutes adding means and variances to the initial 12 variables of the dataset

For RF algorithm, we do not see big differences with respect to Figure 3.9: a breakdown, that was predicted 12 minutes before, has not been predicted at all.

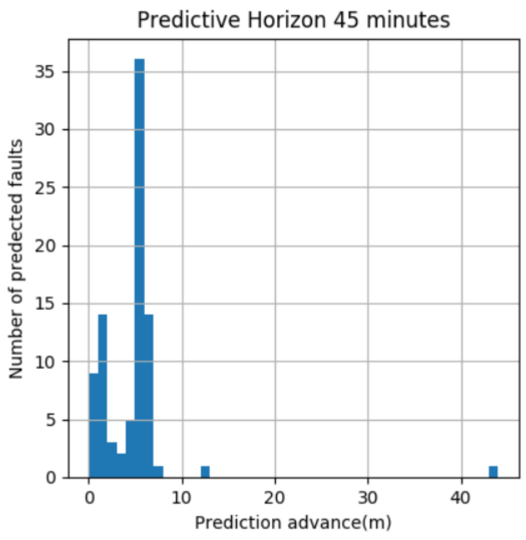

Figure 3.33. Number of predicted faults by LR for PH equal to 45 minutes adding means and variances to the initial 12 variables of the dataset

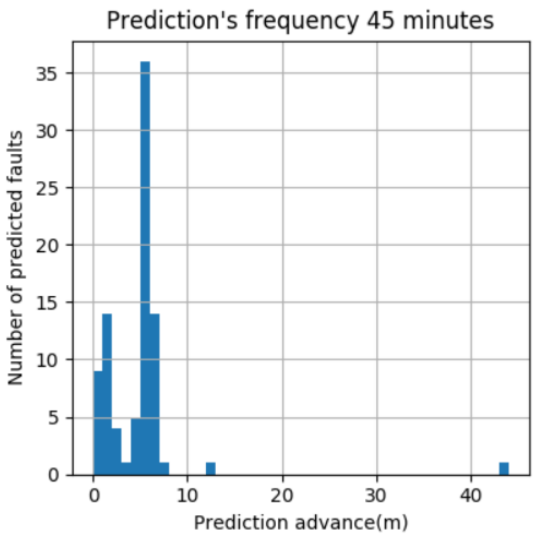

Figure 3.34. Number of predicted faults by SVM for PH equal to 45 minutes adding means and variances to the initial 12 variables of the dataset

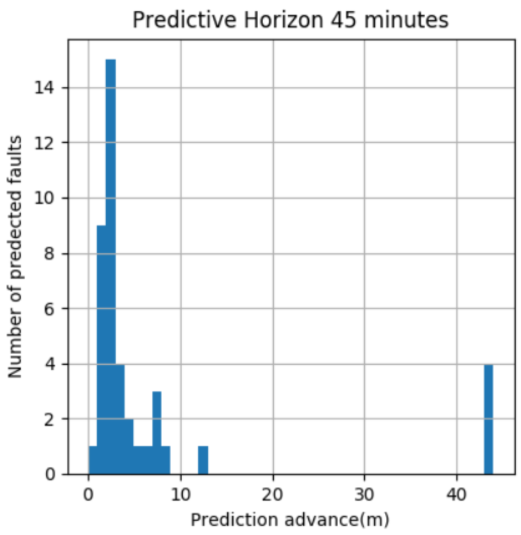

Figure 3.35. Number of predicted faults by MLP for PH equal to 45 minutes adding means and variances to the initial 12 variables of the dataset

With respect to Figure 3.21, the graph is slightly different: 4 breakdowns was predicted with an advance of 45 minutes and the majority of breakdowns was predicted 3 minutes before instead of 6 minutes.

Even if there was some changes in the graph, the situation is almost the same because most predictions are concentrated in the interval of time between 1 minutes and 10 minutes.

Below, we find the comparison between the classifiers based on some performance measures. The number of faults is 127 and the dataset balancement is 88.73 as seen with the previous results.

With respect to table 3.6, we immediatily notice the bad performances of DT, with a lower accuracy and precision and a higher number of false alarms. Also the training time has worsened; it is even higher than that of RF. RF has slighly improved. LR has slightly improced in some measures and slightly worsened in others; the traning time is very high. SVM has detected 86 faults instead of 87 and the number of false alarms has increased by 2 units. MLP has had the worst performances, especially for the number of detected faults that was 42 against the previous 88. Overall, it seems that SVM was the best.

Table 3.7. Results of classifiers for PH = 45 minutes (Class. = Classifiers, Acc. = Accuracy, Prec. = Precision, Rec. = Recall, Det. faults = Detected faults, False al. = False alarms, Tr. time = Training time

Class.	Acc. (%)	Prec.(%)	Rec.(%)	AUC(%)	Det. faults	False al.	Tr. time(sec)
DT	85.89	25.55	**18.18**	57.03	**88**	47	98.58
RF	90.72	**88.43**	13.08	**57.58**	**88**	16	**13.05**
LR	**90.76**	87.90	13.65	56.88	86	17	138.44
SVM	90.72	87.64	13.28	56.41	86	**15**	50.38
MLP	89.29	45.58	11.80	55.08	42	17	35.29

3.3.5 Trying all the 29 variables of the dataset

Until now, we have used only 12 variables of the original feature space. These variables were related to the Buss Mixer. Let us use all the 29 variables present in the original dataset and see if we can improve our results. These variables are related also to the other equipment of the J chain.

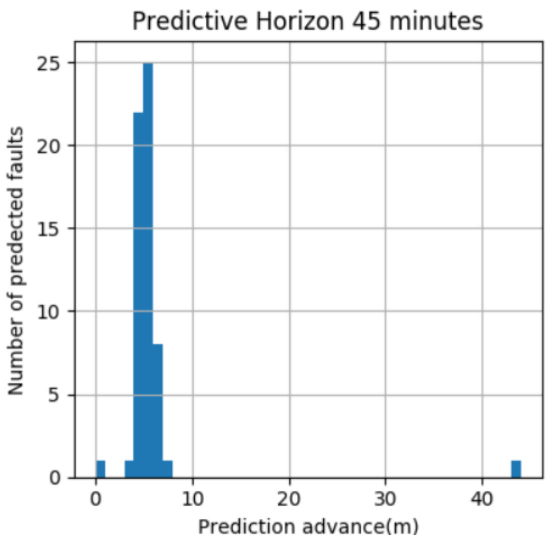

Figure 3.36. Number of predicted faults by DT for PH equal to 45 minutes using all the 29 variables of the dataset

Differently from the case in which we have used only 12 variables, using all the 29 variables changes the distribution of the number of faults predicted by DT; we have zero or few faults between 1 minute and 4 minutes and a high concentration at 5-6 minutes. Furtheremore, this time we have not been able to predict a fault with 12 minutes of advance as in Figure 3.5, but we have been able to predict one time a fault with 45 minutes of advance.

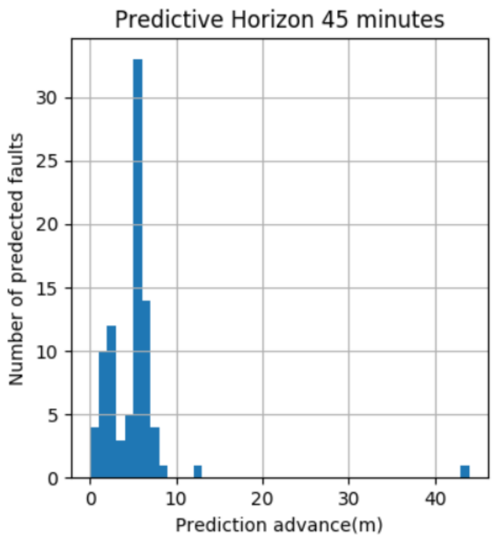

Figure 3.37. Number of predicted faults by RF for PH equal to 45 minutes using all the 29 variables of the dataset

Instead, RF has given us the same distribution. This is because RF uses different DTs that decide together and this has improved the results.

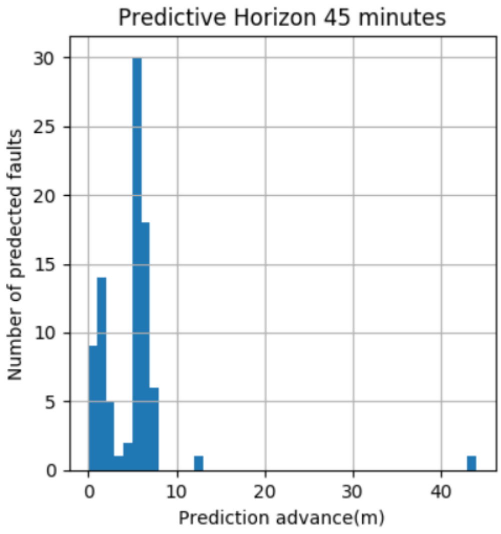

Figure 3.38. Number of predicted faults by LR for PH equal to 45 minutes using all the 29 variables of the dataset

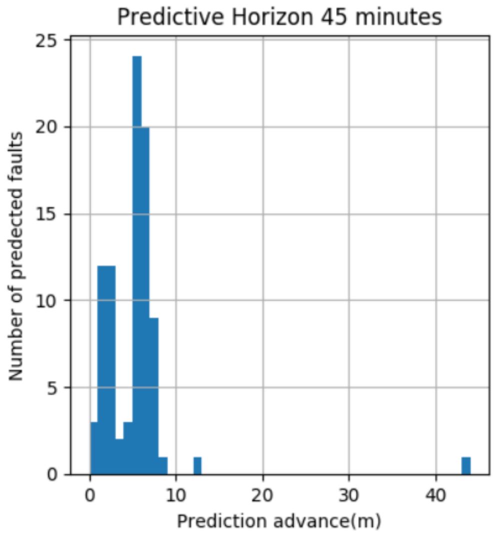

Figure 3.39. Number of predicted faults by SVM for PH equal to 45 minutes using all the 29 variables of the dataset

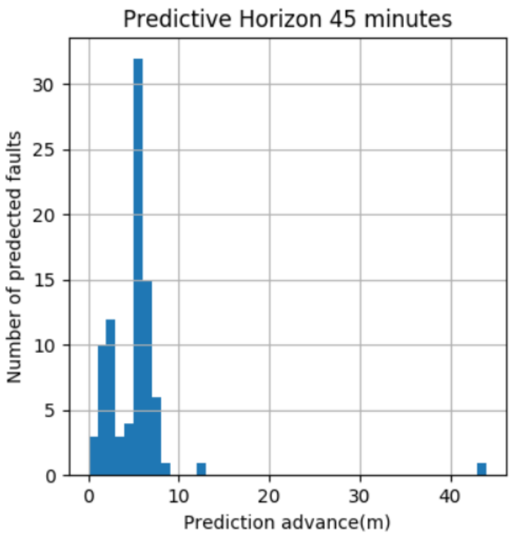

Figure 3.40. Number of predicted faults by MLP for PH equal to 45 minutes using all the 29 variables of the dataset

LR, SVM and MLP like RF give us the same results as before. In all the previous graphs, we see that all classifiers have been able to predict faults with 10 minutes of advance.

The table below shows the comparison between the algorithms and how the measures have changed with respect to the case of 12 variables.

Table 3.8. Results of classifiers for PH equal to 45 minutes using all 29 variablese of the dataset where Clas. = Classifiers, Acc. = Accuracy, Prec. = Precision, Rec. = Recall, Det. faults = Detected faults, False al. = False alarms, Tr. time = Training time

Clas.	Acc. (%)	Prec.(%)	Rec.(%)	AUC(%)	Det. faults	False al.	Tr. time(sec)
DT	90.53	85.19	11.60	55.57	59	9	**2.97**
RF	90.74	87.61	13.44	**57.36**	**88**	16	6.66
LR	90.67	87.83	12.64	56.03	87	18	49.76
SVM	90.75	**88.28**	13.40	56.42	**88**	**15**	20.78
MLP	**90.78**	88.23	**13.75**	56.81	**88**	18	33.12

In bold we have the best value for each performance measure. The first thing we notice is that DT has detected only 59 faults against 88 of the previous case.

The second thing that we notice is that the training time of all algorithms has doubled. This is obvious because of the new variables introduced. The accuracy of DT has slightly worsened together with the rest of measures. For the other classifiers, some measures have worsened, while others have improved. In both cases, the improvement or the worsening was lightweight. LR has detected 87 faults, one plus the previous case, while SVM has reached the 88 detected faults (they was 87). SVM has still the lowest number of false alarms. For the ratio false alarms/detected faults, SVM is the winner. This time, MLP had the highest accuracy and the best recall. However, there was not so much difference with SVM.

Chapter 4

Deep Neural Networks

We have already introduced DNNs in chapter 2 when we have talked about the MLP. Let us continue our discussion about them introducing some parameters to regularize the network and some techniques to enhance results.

4.1 Learning rate scheduling

Finding a good learning rate can be difficult. If we set it too high, training may diverge. If we set it too low, training will eventually converge to the optimum, but it will take a very long time. If we have a limited computing resources, we may have to interrupt training before it has converged, yielding a suboptimal solution. We can be able to find a good learning rate by training our network several times using various learning rates and comparing the learning curves. The ideal learning rate will learn quickly and converge to good solution. There are many different strategies to reduce the learning rate during training.

4.2 Avoiding overfitting through regularization

DNNs typically have a lot of parameters. As a result, the network has an big amount of freedom and can fit a variety of datasets. But this great flexibility also means that it is prone to overfitting the training set. In this section we will present some of the most popular regularization techniques for neural networks: early stopping, l_1 and l_2 regularization.

4.2.1 Early stopping

To avoid overfitting the training set, a first solution is early stopping: interrupt training when its performance on the validation set starts dropping. Although early stopping works well in practice, we can usually get higher performance by combining it with other regularization techniques.

4.2.2 ℓ_1 and ℓ_2 regularizations

We have already introduced this in chapter 2. Like we have done for simple linear models, we can use l_1 and l_2 regularization to constrain a connection weights of a neural network. One way to do this using TF is to simply add the appropriate regularization terms to our cost function.

4.2.3 Convolutional Neural Networks

This is a small degression to complete the discusion about the type of neural networks. We will give a small space also to RNN.

CNNs emerged from the study of the visual cortex of brain, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power, the amount of available training data, CNNs have managed to achieve superhuman performance on some complex visual tasks. They power image search services, self-driving cars, automatic video classification systems, and more. Moreover, CNNs are not restricted to visual perception: they are also successful at other tasks, such as voice recognition or Natural Language Processing (NLP); however, we will focus on visual applications for now.

4.2.4 TensorFlow and classification with neural networks

About TensorFlow

TF is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The architecture is flexible and allows us to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TF was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting ML and DNNs research, but the system is general enough to be applicable in a wide variety of other domains as well.

Results for the industrial case

Table 4.1 shows the number of layers used in the ANN, the number of units per layer and the performance measures which we have tried to get a longer prediction time. We have started from shallow ANNs to arrive to DNNs (10 layers). The number of units forms a sort of funnel, from the bottom layer to the top one. Moreover, the results are, in the majority of cases, similar to those obtained using classic ML algorithms.In fact, moving from a shallow ANN of one layer (the second row of the Table 4.1) to a DNN of 10 layers (the last row), the peformance measures are not too different. In the middle, there are ANNs with different depths, number of nodes per layer and different learning rates, that give similar results or bad ones (an example is the fourth row). A DNN should give us better results because incrementing the number of layers, the model becomes more complicated and sophisticated. By incrementing therefore the number of layers, we would expect better results. If we do not get better results, it is possibile that the cause are the data. From this fact, we observe that the data must be improved because they seem not to be relied to failures of the Buss Mixer. In the previous tests carried out with the classic ML algorithms, we have shown, using SVM, that one variable is more relevant than the others (see Subsection 3.3.3). The classifiers lean only on this variable to give the label to the measurements and this seems to be not sufficient. To improve the results, it would be advisable to study together with the industrial experiments if other data can be acquired from the sensors present actually in the industry or if new sensors can be installed, after having studied on a physical level which could be the variables to be monitored.

To carry out the tests, we have used the version of TF that uses also the GPU, to go faster. The GPU used was Nvidia GeForce GTX 950M, with 4 GB of memory dedicated. In the Table 4.1, we do not see layers with 1024 units because of the "ResourceExausted" error. The computer used for the experiments was not able to allocate the structure for 1024 units. In future, we can think about implementing our techniques on more powerful GPUs and using a cluster of machines.

Table 4.1. Results of predictions using ANN with different values for parameters. PH = 45 minutes and the number of faults is 127 (L. = Number of layers, L.R. = Learning rate, A. = Accuracy, P. = Precision, R. = Recall, D.F. = Detected faults, F.A. = False alarms, T.T. = Training time)

L.	Units	L.R.	A. (%)	P. (%)	R. (%)	D.F.	F.A.	T.T. (sec)
1	8	1e-2	90.65	88.08	12.44	86	20	39.59
1	8	1e-3	90.76	88.04	13.61	88	20	33.01
1	8	1e-4	90.62	87.74	12.12	87	20	37.67
1	16	1e-3	89.99	77.84	6.27	39	22	31.9
1	16	1e-4	**90.77**	87.69	**13.78**	88	70	32.33
1	32	1e-2	90.01	78.22	6.38	39	19	30.56
1	32	1e-3	90.76	88.27	13.55	88	21	30.73
1	32	1e-4	90.72	87.71	13.19	**89**	47	29
1	64	1e-2	90.01	78.28	6.40	39	**16**	33.48
1	64	1e-3	90.74	87.93	13.43	88	23	36.50
1	64	1e-4	90.77	88.04	13.66	**89**	25	33.37
1	128	1e-3	90.74	87.77	13.39	**89**	44	32.53
1	128	1e-4	90.74	87.76	13.46	88	82	35.60
1	256	1e-3	90.76	88.04	13.56	88	22	43.35
1	256	1e-4	90.75	87.70	13.51	**89**	82	39.03
1	512	1e-2	90.01	78.27	6.38	39	**16**	149.4
1	512	1e-3	90.76	88.14	13.63	88	22	146.11
1	1024	1e-3	**90.77**	**88.16**	13.63	88	26	236.87
2	512 256	1e-3	90.75	87.73	13.59	88	66	151.01
2	512 256	1e-4	90.76	88.14	13.59	88	34	150.70
3	512 256 128	1e-3	**90.77**	88.06	13.69	88	28	164.02
10	512 512 512 256 128 64 32 16 8 4	1e-3	90.75	88.08	13.64	88	19	413.32

Chapter 5

Conclusions

In this thesis, we have studied a ML approach to PC for industrial processes. Specifically, we have faced a real application namely predicting failures of a Buss Mixer, a machine used in aluminium production. The used data has been acquired in the framework of the MONSOON project. It is a time series of 29 variables. Initially, we have only used the 12 variables related to the chain of Buss Mixer in the aluminium production process, and then we have extended the experiments to 29 variables. The dataset has been modified introducing the concept of PH: when a measurement is labeled as failure, we have labeled as failure also the m previous measurement. The new modified dataset has been given to different ML classification algorithm. We have obtained predictions of failures of 10 minutes of advance. This is in line with results obtained by MONSOON analysts.

ℓ_1 penalty reveals that one variable is more significant than all the others and through it, the algorithms decide the label for a specific measurement. The fact that there is only one relevant variable suggests us to think that all the other variables are not associated with breakdowns of the Buss Mixer. We have enough data to process, but this data are not meaningful for this case.

To make the ML algorithms learn patterns better, we have added means and variances as variables of the dataset. The results have not changed. We have then considered the whole dataset which contains 29 variables and the obtained results have not substantially changed.

Finally, we have applied DL using an ANN with 10 layers; We have expected the results to improve with the increment of the number of layers; it has not been possible probably because of data; in fact, only one varibale is relevant.

Moreover, all the algorithms have predicted the breakdowns of 6 minutes of advance in the majority of times with a variable range between 1 and 10 minutes. They have also predicted a failure with an advance of 12 minutes and 45 minutes.

In order to improve our results, i.e., to increase the prediction advance in the future, we could undertake different actions:

- to acquire new data from the industrial partner that are more closely related to the breakdowns, which might require the installation of new sensors;
- to combine the actual variables to extract new features (for example, use: mean, variance, bunches of measurements). We could combine various variables to obtain other variables that could help us to predict better.
- to try DNNs: we could use DNNs trying more combinations of their parameters or use CNNs and RNNs.
- in this work, we have used one GPU that has allowed us to build ANNs with 10 layers. In the future, we could extend the work using a greater number of GPUs more powerful and build more deeper ANNs.

Bibliography

[1] General Electrics Intelligent Platforms, "The Rise of Industrial Big Data: Leveraging large time-series data sets to drive innovation, competitiveness and growth—capitalizing on the big data opportunity" Aug. 2012

[2] L. Monch, J. W. Fowler, S Dauzère-Pérès, S. J. Mason, and O. Rose, "A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations", J. Sched, vol. 4, no. 6, pp. 575-599, 2011

[3] G. A. Susto et al., "A predictive maintenance system for integral type faults based on support vector machines: An application to ion implantation", Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE), pp. 195-200, 2013

[4] G. Koksal, I. Batmaz, and M. C. Testik, "A review of data mining applications form quality improvementin manufactoring industry", Expert Syst. Appl., vol. 38, no. 10, pp. 13448-13467, 2011

[5] G. A. Susto, S. Pampuri, A. Schirru, A. Beghi, and G. De Nicolao, "Multistep virtual metrology for semiconductor manufactoring: A multilevel and regularization methods-bases approach", Computers and Operations Research, vol. 53, pp. 328-337 January 2015

[6] R. K. Mobley, "An Introduction to Predictive Maintenance", London, U.K, Butterworth-Heinemann, 2002

[7] G. A. Susto, A. Beghi, and C. De Luca, "A predictive maintenance system for epitaxy processes based on filtering and predictions techniques", IEEE Trans. Semicond. Manuf. vol. 25, pp. 638-649, 2012

[8] http://www.busscorp.com/en/anodepaste.htm

[9] A. Coates, B. Carpenter, C. Case, S Satheesh, B. Suresh, T. Wang, D. J. Wu, A. T. Ng, "Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning", International Conference on Document Analysis and Recognition, pp. 1, 2011

[10] Aurélie Géron, "Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems", 2017

[11] J. McCarthy, E. Feigenbaum "In Memoriam Arthur Samuel: Pioneer in Machine Learning", AI Magazine, vol. 11, no. 3, 1990

[12] https://www.umiacs.umd.edu/~hal/

[13] Johns Hopkins University Press, "Perspectives in Biology and Medicine", vo. 43, no. 2, pp. 193-216, 2000

[14] Guang-Bin Huang, "Cognitive Computation", vol. 7, no. 3, pp. 263–278 June 2015

[15] D. H. Hebb, "The organization of behaviour a neuropsychological theory", New York, John Willey and sons, Inc. 1949

[16] http://yann.lecun.com/exdb/mnist/

[17] C. Dan, U. Meier, J. Schmidhuber, "Multi-column deep neural networks for image classification" IEEE Conference on Computer Vision and Pattern, 2012

[18] L. Yann, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based Learning Applied to Document Recognition" Proceedings of the IEEE, vol 86, pp. 2278-2324, November 1998

[19] W. Li, M. Zeiler, S. Zhang, Y. LeCun, R. Fergus, "Regularization of Neural Network using DropConnect", International Conference on Machine Learning (ICML), 2013

[20] T. Wuest, D. Weimer, C. Irgens, K. D. Thoben, Machine learning in manufacturing: advantages, challenges, and applications, Production and Manufacturing Research, 4:1, 23-45,

[21] http://www.iiconsortium.org/press-room/09-18-17.htm

[22] https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-predictive-overview

[23] J. M. Maciejowski, Predictive Control: with Constraints, Pearson Education, 2002

[24] https://www.sysman.it/processi/iot-predictive-maintenance.html

[25] Warren S. M., W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Journal The bulletin of mathematical biophysics, Springer, vol. 5, pp. 115–133, December 1943

[26] M. Minsky, S. A. Papert, Perceptrons, MIT Press, 1969

[27] Andrew Y. Ng Feature selection, L1 vs. L2 regularization, and rotational invariance, Computer Science Department, Stanford University, Stanford, USA

[28] Michele Banko, Eric Brill, "Scaling to Very Very Large Corpora for Natural Language Disambiguation", Microsoft Research, 2001

[29] A. Halevy, P. Norvig, F. Pereira, "The unreasonable effectiveness of data", IEEE Intelligent Systems, vol. 4, pp. 8-12, 2009

[30] https://magenta.tensorflow.org/

[31] Karl J. Astrom, T. Hagglund, Advanced PID Control, ISA - The Instrumentation, Systems and Automation Society, 2006

[32] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, A. Beghi, "Machine Learning for Predictive Maintenance: A Multiple Classifiers Approach", IEEE Transactions on Industrial Informatics, vol. 11, pp. 812 - 820 2015

[33] M. G. Forbes, R. S. Patwardhan, H. Hamadah, R. B. Gopaluni, "Model Predictive Control in Industry: Challenges and Opportunities", IFAC-PapersOnLine, vol. 48, pp. 531 - 538, 2015