University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Generative Adversarial Network for Predictive Maintenance of a packaging machine

Supervisor
Prof. Alberto Testolin
University of Padova

Co-supervisor
Dr. Marco Boresta
ACT Operations Research

Master Candidate
Adriano Rasetta

Student ID
1225009

Academic Year
2021-2022

“TO THOSE WHO HAVE HELPED AND ENCOURAGED ME ALONG MY PATH”

Abstract

Generative models have been designed to discover and learn the latent structure of the input data in order to generate new samples based on the regularities discovered in the data. Starting from the first simplest models such as the Restricted Boltzmann Machines up to the Variational Autoencoders and Generative Adversarial Networks (or GAN), these models have experienced a surprising development in generating data as similar to reality as possible. The potential of these models, especially in Deep Learning, has led to the most disparate applications: generation of images, videos and music, image-to-image translations, text-to-image translation and conversion of low resolution images to high resolution, to name a few.

In this thesis work, carried out during the internship period of the Master's Degree, the main focus is on GANs, a generative model that makes use of the principles of supervised training through the use of two competing "sub models": a generator, trained to produce new realistic samples, and a discriminator, which tries to distinguish between real and generated data. Usually, when this model is employed, the focus is mainly on the role of the generator used to produce new data. In this case, however, the idea is to use the discriminator as a binary classifier in the context of Predictive Maintenance of a packaging machine. In other words, the discriminator obtained as a result of GAN training is used to classify the state of the machine as either normal or critical. After an initial pre-processing and exploration of the datasets, the results obtained are compared with other classifiers. Finally, the limits and possible developments of this approach are discussed.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ACT OR ..... Analytics Control Technology Operations Research

Adam ..... Adaptive Moment Estimation

AdamW ..... Adaptive Moment Estimation With Decoupled Weight Decay

CV ..... Cross Validation

DL ..... Deep Learning

FFNN ..... Feed Forward Neural Network

GAN ..... Generative Adversarial Network

IG ..... Information Gain

KNN ..... K-Nearest Neighbour

MCC ..... Matthews Correlation Coefficient

PCA ..... Principal Component Analysis

PdM ..... Predictive Maintenance

RBF ..... Radial Basis Function

ReLU ..... Rectified Linear Unit

SVM ..... Support Vector Machine

t-SNE ..... T-distributed stochastic neighbor embedding

1 Introduction

For some years now, the efforts in many research fields have been directed towards artificial intelligence models thanks to their ability to manage large amounts of data. The industrial field is not an exception, in fact even in this sector there has been a wide application of Machine Learning models and an important search for increasingly complex models to achieve high levels of accuracy. Thanks to the increase in computing power in the last few years, the focus has shifted more towards Deep Learning (DL) which has provided new state-of-the-art models in many fields including Computer Vision, Language Processing and Intrusion Detection System. Currently, the industrial system is moving towards a fourth revolution called Industry 4.0 which uses a combination of software, sensors and intelligent control units to improve production processes. These tools allow the automation of functions for the constant monitoring of production through the analysis of the data collected [17].

As regards the project presented in this thesis, we want to offer an exploratory contribution to the use of a generative model of DL in the industrial field, in particular in Predictive Maintenance (PdM). The model considered is a Generative Adversarial Network (GAN) adapted to a classification task in the context of anomaly detection: the model aims to identify those observations that deviate from the normal dynamics of the data, that is when the machine under investigation it is in a working state.

In Chapter 2, through a brief examination of the available literature, the framework of PdM is defined in a general way. The focus is on supervised and unsupervised models typically used in this area. Subsequently, attention is paid to some examples of adapting a GAN for PdM tasks.

The next chapter presents the project carried out during the internship period at Analytics Control Technology Operations Research (ACT OR), a math-technology company. After a brief description of the company and its research areas, the PdM problem under study is defined through the description of the packaging machine, the experiments conducted on it, the data collection and finally the objective of the project.

Chapter 4 provides a description of the datasets used and the pre-processing performed. It is an exploration phase in which the fundamental choices to solve the problem in question are discussed. Next, two dimensionality reduction techniques are applied to provide a visual representation of the data dynamics. The chapter concludes

with the description of two clustering techniques applied to the single datasets. These models are intended to provide a preliminary idea about the divisibility between the samples belonging to the normal and critical classes without the use of a label.

Chapter 5 describes all the classification techniques used in this project. The mathematical foundations of the classifiers and DL models are provided with particular attention to the architecture of the GAN and the ways in which it is adapted to the classification task required in the project.

The implementation details, the hyperparameters and the final results of each model are presented and compared in Chapter 6. In one of the paragraphs the samples of the datasets are compared with some samples generated by the generator in order to offer an additional point of view to evaluate the performance of the GAN. Then, a final comment is provided on the results obtained and on the advantages and disadvantages of using this generative model for the task considered.

Finally, in Chapter 7, there is an overview of the limitations and the possible future developments to explore. Among these we find the use of unsupervised or semi-supervised models, the use of different architectures for the GAN and a proposal for an alarm reactivity system to be adopted directly on the machine.

2

Predictive maintenance overview

With the increase in the amount of industrial data available, the interest in DL architectures for PdM has become very common, especially in optimizing maintenance tasks. Maintenance is defined by EN 13306 [18] as "the combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function". This definition implies three types of maintenance: improvement maintenance, which deals with improving reliability and safety maintaining the original functionality, predictive maintenance, carried out before failure occurs, and finally corrective maintenance, which deals with the replacement of damaged parts when the machine is not working [17].

In this project, the focus is on the implementation of a basic predictive maintenance system. This type of system, as defined by Venkatasubramanian et al. [21], should have the following properties: "quick detection and diagnosis, isolability (distinguish among different failures), robustness, novelty identifiability, classification error estimation, adaptability, explanation facility, minimal modelling requirements, real-time computation and storage handling, multiple fault identifiability". From a methodological point of view, however, PdM can be classified and defined through three approaches: physical model-based, data-driven and hybrid. The first, through the use of systems' knowledge, provides a mathematical description of the degradation level of the system. The data-driven approach provides predictions on the state of the system through continuous monitoring. It makes use of statistical methods and artificial intelligence, and is particularly suitable for complex systems, even if it becomes difficult to correlate the result of these methods with the physical meaning. Finally, the last approach is a combination of the two approaches. Data-driven PdM, today mainly based on DL models, is structured according to the following incremental steps: anomaly detection, diagnosis, prognosis and mitigation [17].

As shown in Figure 2.1, at the base there is the anomaly detection, in this phase the purpose is to determine whether the machine is working under normal conditions or not. There are three ways of approaching the problem depending on the Machine Learning task considered: the classification (class labels are available), one-class classification (only one class label is available, typically non-failure data) and clustering (unlabelled data). Diagnosis consists in understanding the nature of the anomaly identified and whether this can lead to a real failure or not.

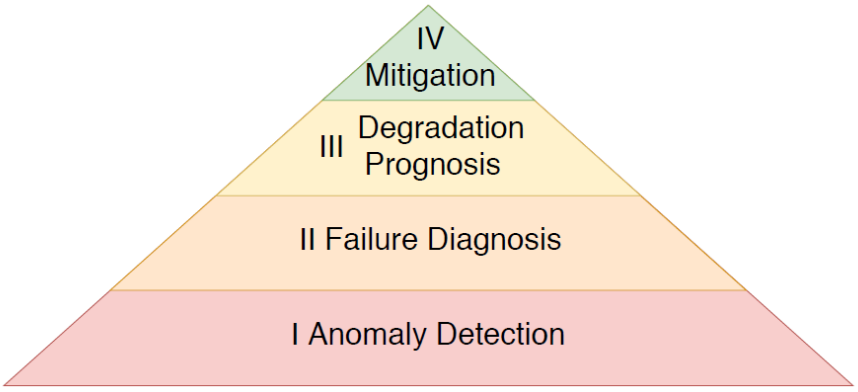


Figure 2.1: Predictive maintenance roadmap represented by a stages of a pyramid chart [17].

Once the anomaly has been identified and the diagnosis has been performed, the prognosis phase is responsible for monitoring the evolution of the degradation of the machine state and estimating the remaining time or cycles until a failure occurs. Finally, after the prediction of the remaining time, some actions are taken to mitigate the failures. At this point, all necessary corrections are made to restore the normal state of the machine [17]. This exploratory thesis project is based exclusively on anomaly detection.

From a practical point of view, the most commonly monitored components are bearings, blades, engines, valves, gears and cutting tools, instead among the types of failures that can be identified it is possible to find imbalance cracks, fatigue, abrasive and corrosion wear, rubbing, defects and leak detection. More rigorously, however, Li and Gao [10] classifies the types of system failures into three categories: component failure, environmental impact, human mistakes and procedure handling.

In order to monitor the status of a system, different techniques and sources of information are used. The most common techniques include Mechanical Ultrasound, Vibration Analysis, Wear Particle Testing, Thermography, Motor Signal Current Analysis and Nondestructive Testing, Torque, Voltage and Envelopes, Acoustic Emission and Pressure or Temperature Monitoring. Instead, on an informative level, there are the environmental and operational conditions. They represent the conditions under which a machine works. Environmental conditions are represented by external conditions such as temperature, while operating conditions are technical specifications of the process such as speed or position. Another source of information comes from the sensors of the machine. These data are collected in the form of time series in order to facilitate the identification of patterns and trends [17].

Regarding the use of a GAN in this specific area, there are not many papers in the literature. However, for this thesis, papers relating to classification and anomaly detection have also been consulted. For example, Odena et al. [12] provide the first adaptation of a GAN for a classification task through the implementation of an Auxiliary Classifier GAN (or ACGAN). The peculiarity of this model lies in the loss function consisting of one part responsible for identifying the real samples and the other responsible for identifying the correct label. ACGAN has been widely used in image generation and classification tasks, usually both the generator and the discriminator are Convolutional Neural Networks. In the context of anomaly detection, however, the AnoGAN of Schlegl et al. [16] has been one of the first models proposed. The model has the structure of a Deep Convolutional Generative

Adversarial Network (or DCGAN) [14] with the addition of an adaptation that allows mapping from samples to latent space. Through this adaptation, the anomaly score is calculated, capable of identifying the anomalous samples. A further development of this model is the Time series Anomaly detection with GAN (or TAnoGAN) [2], a model for unsupervised anomaly detection in time series data. Finally, in the specific context of the time series applied to the PdM framework, Liu et al. [11] propose a model based on long-short-term memory (LSTM) networks called LSTM-GAN. This particular implementation can solve the disadvantage of vanishing gradients and the mode collapse typical of GANs. Furthermore, it is able to identify abnormal data. This PdM model includes a predictive model for anomaly detection and a maintenance decision model. The first predicts the state of the machine and anticipates the possible faults of the machine. The second deals with organizing maintenance personnel and offering a maintenance plan.

A standard GAN has been implemented for the project discussed here. This has been adapted to the PdM classification task focusing mainly on the role of the discriminator, used in the testing phase as classifier that is independent from the generator. It is important to specify that the model as a whole is trained as an anomaly detection model as only samples belonging to the normal class are used. However, the discriminator, placed in front of a new sample, attempts to classify it as a Feed-Forward Neural Network would do in a classification task.

3 Internship project

This chapter presents the company where the internship has been carried out and the related project. In particular, the data collection, the experiments conducted on the machine and the objective of the project are described.

3.1 The company

The project presented in the thesis has been developed during the 5-month internship period at ACT OR (Figure 3.1). ACT OR is a math-technology company founded in 1996, it is specialized in providing corporate decision-making software and process control solutions. The company offers robust and innovative solutions to cut costs, improve productivity and service levels, predict trends and behaviors, improve safety, support the care of patients, reduce risks and manage crises. ACT OR work projects can be divided into five main areas:

1. Math-optimization: the company uses optimization tools to find solutions in areas such as calculating the optimal routing of vehicles, estimating the optimal planning of resources or defining the optimal quantity for the inventory.
2. Simulation: in this field discrete sequences of events are represented and studied with the use of discrete-event-simulation models. These models are able to capture the behavior of complex systems present in contexts such as supply chains, factories, distribution centers and warehouses.
3. Artificial intelligence: AI-based software solutions are offered through the implementation of Machine Learning models for a wide range of problems ranging from consumer behavior prediction to robotics and automation.
4. Forecasting: ACT OR implements demand-based forecasting methods to support decision-making in areas such as managing bookings on a particular website, the energy requirement or failures for maintenance purposes.

Figure 3.1: ACT OR logo.

5. Process and control: this area deals with the design of plant control, automation and protection, in simulation and dynamic control of industrial processes.

The project discussed here falls within the area of forecasting, in particular in the prediction of failures for maintenance purposes.

3.2 The task and the failure scenarios

This project is the result of the collaboration between ACT OR and a company in the production of packaging machines. The data collection has been carried out by mounting sensors on a packaging machine, an industrial machine designed to vacuum a series of recipes or types of food, consisting of a conveyor belt and a sealing chamber where the vacuum takes place.

The objective of the project is the implementation of a GAN for a binary classification task in the context of PdM, in short, the model must be able to distinguish between the ”normal” and ”critical” state of the machine. It already has an alarm system but only for severe problems. Furthermore, the time elapsed between the failure and the alarm is very long. The main purpose of the model is to provide a better alarm system than the current one.

During the normal operation of the machine two tamperings were carried out in order to induce the critical state and simulate two failure scenarios:

1. Vacuum pump wears out;
2. Valves wear out.

The data was collected when the machine was in a normal production state and after the induced failure. This state has been labeled as ”critic”. The two failure scenarios were simulated by an expert by opening two different taps, one for each experiment, that simulate the vacuum pump and valves wear out. The variables of the two failure scenarios were both labeled as critic without distinction between the two situations because the aim of the project is simply to identify the criticality. Future developments of the project will lead to the implementation of models able to distinguish the types of criticalities. An example of a report on the experiment conducted on May 18 is presented in Table 3.1.

Hour	Event	Machine event	Status	Machine status
8:31	Start heating	Energy consumption increases	normal	Preparing to run
8:56	Start of production	The machine starts packing	normal	Running
12:00	Stop machine	Experiment pauses	NA	Stopped
13:32	Start of production	The machine starts packing	normal	Running
13:36	Tap open	Cycle time is increased	critical	Running
14:16	Tap open	Cycle time is increased	critical	Running
15:25	Tap open	Cycle time is increased	critical	Running
15:53	Tap open	Cycle time is increased	critical	Running
16:07	Tap open	Cycle time is increased	critical	Running
16:47	Tap open	Cycle time is increased	critical	Running
16:50	Machine's alarm	The machine stops	critical	Stopped in alarm

Table 3.1: Example report for the May 18th experiment.

4

Datasets exploration

The chapter provides a detailed description of the datasets used and the pre-processing performed. Two dimensionality reduction techniques were applied to each dataset with the aim of visualizing the data through only two principal components. Furthermore, two clustering methods were implemented to identify the best number of clusters to use to represent our datasets with unsupervised learning and to observe how the normal and critic samples are divided among the clusters.

4.1 Dataset description

The company made four datasets available for the project, each with 34 variables but with a different number of observations. The variables were detected every minute, in fact every minute corresponds to an observation. The first dataset derives from the experiment that tried to simulate the failure scenario of valves wear out, while the second and third datasets simulated the vacuum pump wear out. The fourth dataset contains only normal samples. The details of the datasets are summarized in Table 4.1.

Among the 34 variables considered, one is a time variable, eight represent the data collected by the sensors placed on the machine, twenty-four are variables added through transformations of the first eight (square of each

Date	Dataset	Samples	Normal	Critic	Failure scenario
21-22-23th July	1	1523	1014	509	Valves wear out
18th May	2	370	175	195	Vacuum pump wear out
19th May	3	390	211	179	Vacuum pump wear out
26th May	4	161	161	0	NA

Table 4.1: Summary table of the datasets.

Variable name	Description
Act_Temp1	Current mold temperature
Act_Speed	Packaging cycles per minute
Act_L1Amp Act_L2Amp Act_L3Amp	3 phase actual current
CyCounter	Cycle counter
Energy	Total electricity consumption
Gas	Total gas consumption

Table 4.2: Summary table of the variables and their description.

variable, delta with one-minute earlier and two minutes earlier variables) and one represents the binary target variable `status` with normal and critic labels. A brief description of the eight variables is provided in Table 4.2

The behavior of each variable over time for each dataset is shown in Figure 4.1. The dashed line defines when the tampering has been carried out. From the graphs it is possible to observe how in datasets 2 and 3 the variables `Act_Speed` and `CyCounter` change immediately after tampering. For the other variables it is not possible to identify an immediate change.

Finally, the bar graph (Figure 4.2) shows the number of samples for each dataset and how they are distributed with respect to the variable status. It is possible to notice that the first dataset is unbalanced while the second and third datasets have equally divided classes. In order to deal with the imbalance between classes, several metrics have been taken into consideration in the evaluation of the models with particular reference to balanced accuracy.

4.2 Pre-processing

Much of the pre-processing has been done by the company, so only minimal adjustments have been made for the classification task. The steps in this phase are the following:

1. first of all, the temporal variable has been removed and the normal and critic labels have been replaced with 0 and 1 respectively;

2. subsequently a dataset has been created for training and validation with Cross Validation (CV). This step has been performed by merging datasets 1, 2 and 4 for a total of 2054 samples. For all the models, excluding GAN, a simple merging has been done. Instead, for the GAN, two datasets have been created, one containing all normal and another containing all critic samples. The details on the choice to carry out this division will be presented in the section relating to the model;

3. then, the training data has been shuffled and normalized subtracting the mean and then dividing the difference by the standard deviation. Finally, dataset 3 has been used for testing.

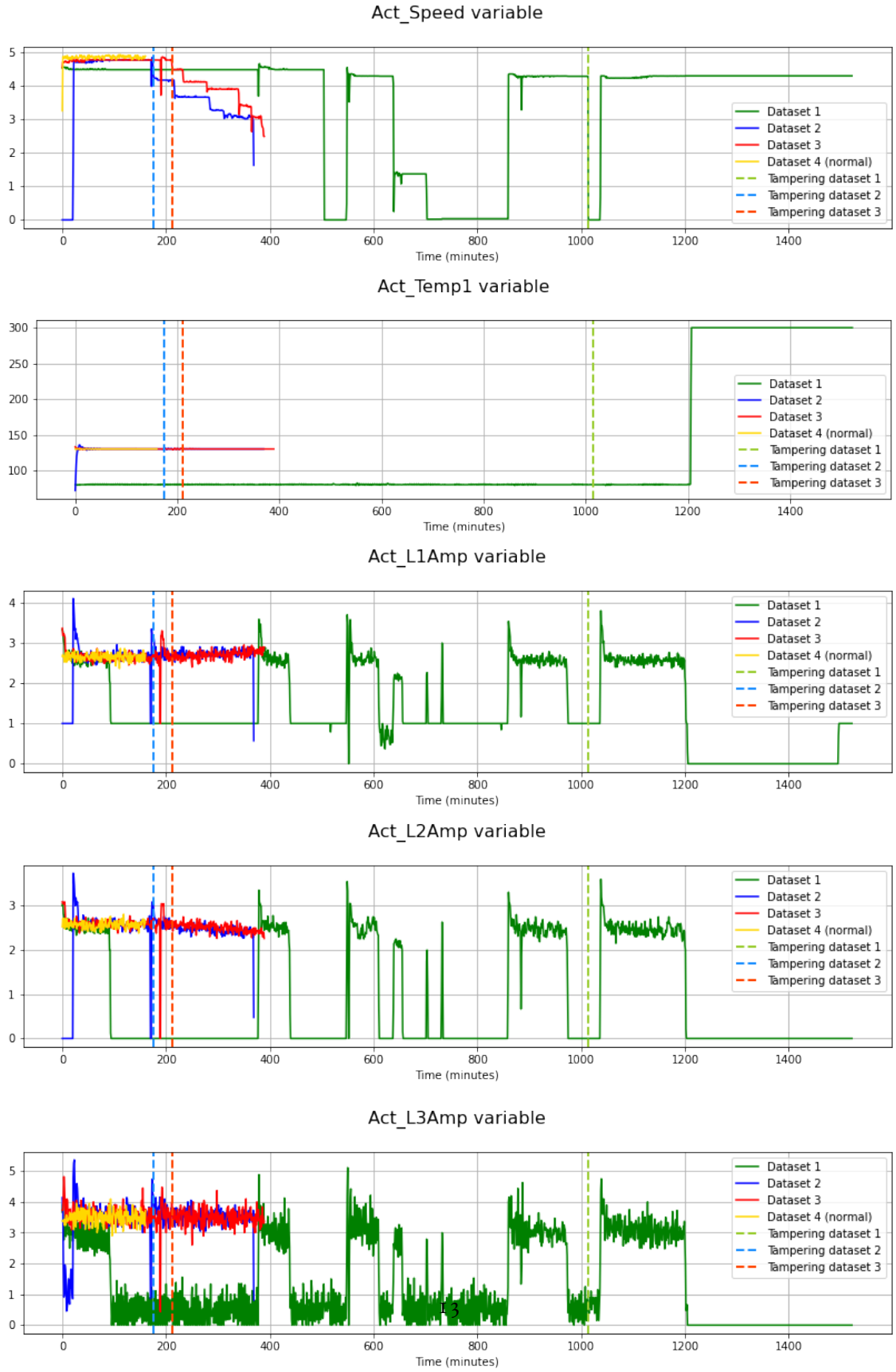


Figure 4.1: Variable over time for each dataset.

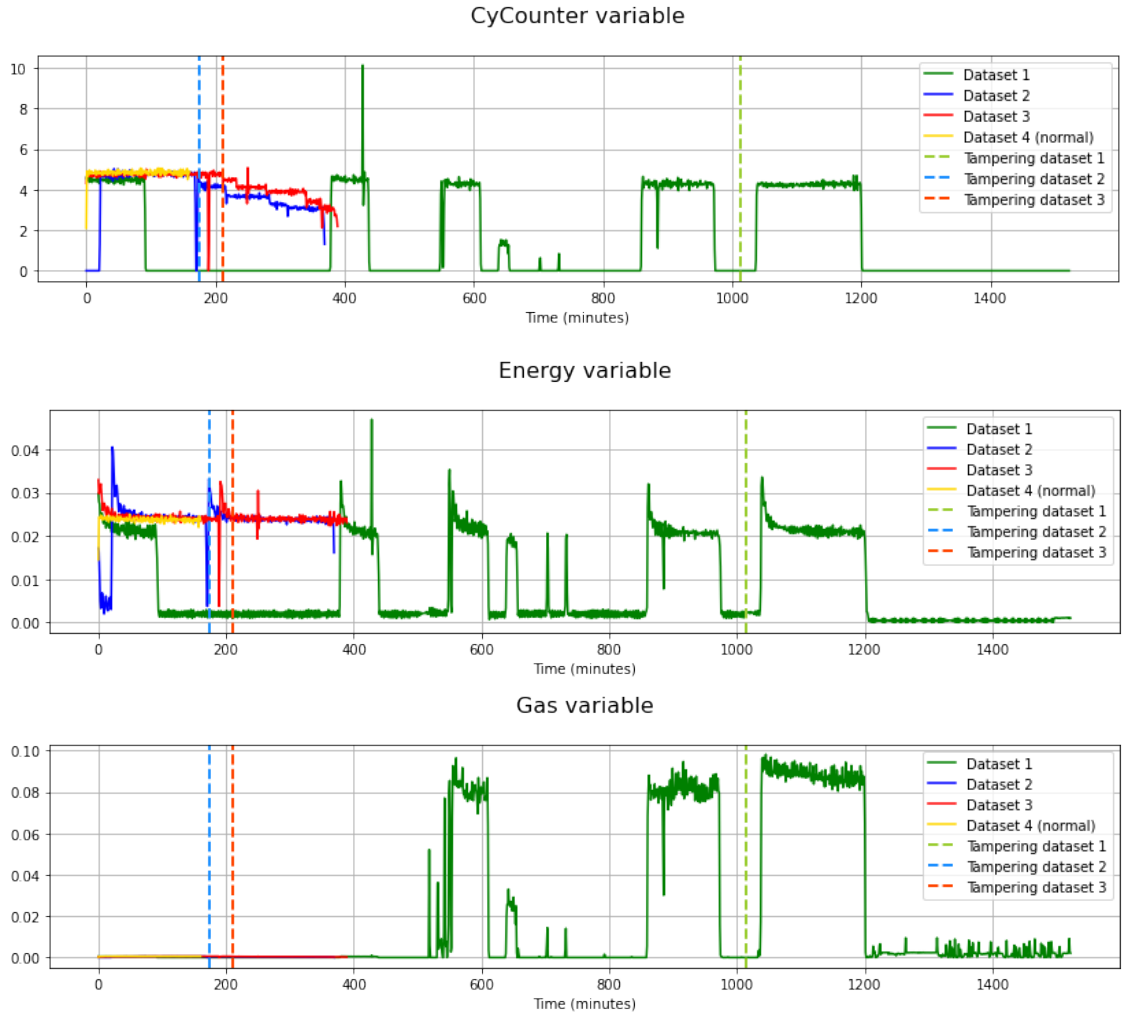


Figure 4.1: Variable over time for each dataset (cont.)

The choice to divide the data for training and testing in this way depends on the experiments performed on the machine and the objective of the project. In the training set we have a dataset for each type of experiment (dataset 1 and 2) and dataset 4 containing only normal samples. In this way the models can learn both criticality scenarios simulated on the machine. Although it is a classification task, the time component has been taken into account. In the test phase, a dataset that simulated a failure on a typical production day was used. In this way it was possible to see if the machine alarm was anticipated by the model. In other words, it would not have made sense to merge all the datasets and make a division between training and testing considering the temporal nature of the task. A limitation of the data provided by the company lies in not having a dataset for each type of failure scenario to test the model. The test was carried out with only one type of experiment. Further project developments will have to take this aspect into consideration.

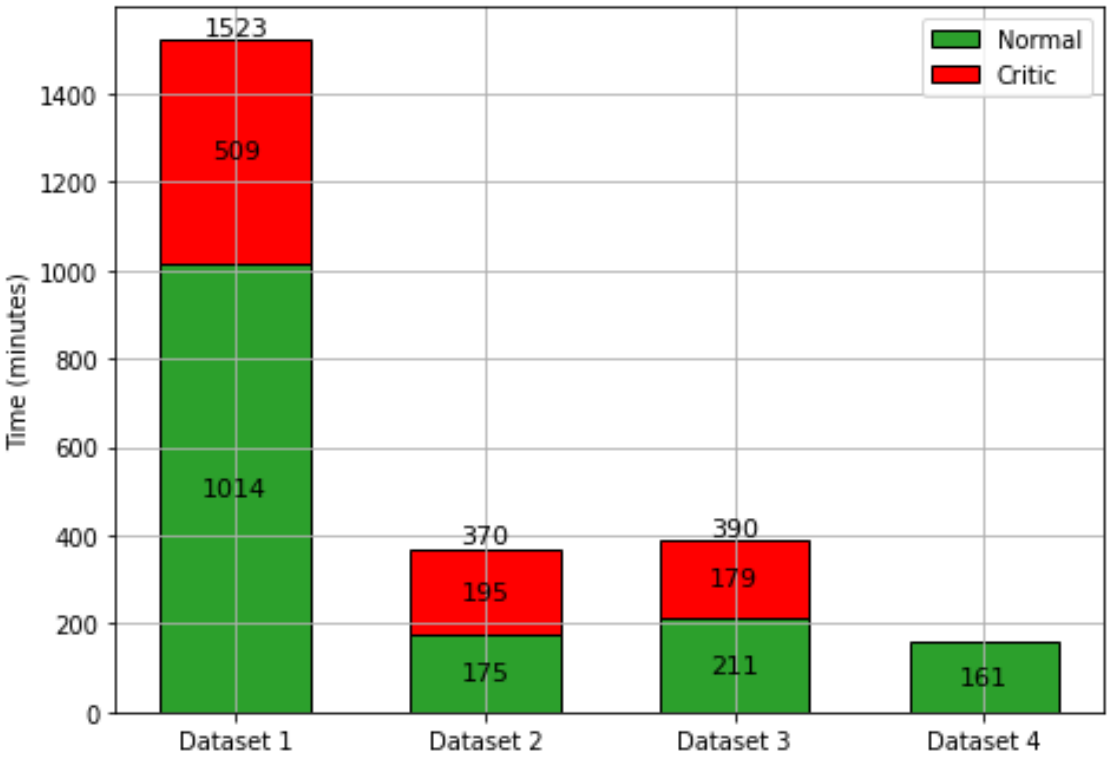


Figure 4.2: Distribution of samples with respect to the variable `status` among the datasets.

4.3 Dimensionality reduction techniques

In this part two techniques of dimensionality reduction are described and applied in order to visualize the distribution of normal and critic samples in a two-dimensional space. The goal is to understand if the use of DL models is preferable to simple classifiers. In case the obtained clusters are clearly distinguishable, the use of complex models such as a Neural Network or a GAN may not be necessary. Dataset 4 with all normal samples is excluded from this part.

4.3.1 Principal Component Analysis

Principal component analysis (PCA) is a method used for visualization and dimensionality reduction [13] [9]. Considering the linear algebra behind the method, the goal is to identify a linear combination of the original basis to filter out the noise and better re-express the dataset. In particular, if X represents the initial dataset and Y its new representation, P is the orthonormal matrix that linearly transforms X into Y according to the equation $PX = Y$. The matrix P, whose rows are the principal components of X, can be computed using the eigenvector decomposition theorem. In other words, the rows of P are the eigenvectors of the covariance matrix of X. These

new vectors are ordered according to the magnitude of their eigenvalues. From a statistical point of view, the eigenvectors determine the directions of the new feature space, and the eigenvalues explain the variance of the data along the new feature axes. Dimensionality reduction is obtained by considering only a small number of eigenvectors of the P matrix, those with the highest eigenvalues. In this way the new matrix Y has a reduced number of new independent features and ordered by explained variance.

Figure 4.3 shows the results obtained with the PCA with the first two components (approximately 60% of the variance explained) for each dataset. In the first dataset we do not have a clear distinction between classes, on the contrary in the other two it is possible to identify two clusters, here the tampering seems to have easily identifiable effects. Especially in the case of the vacuum pump wear out experiment (dataset 2 and 3), it seems possible to obtain excellent results even using simple classifiers (e.g., Logistic Regression).

4.3.2 t-distributed Stochastic Neighbor Embedding

T-distributed stochastic neighbor embedding (t-SNE) is a nonlinear dimensionality reduction technique developed by Van der Maaten and Hinton [19] for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It models each high-dimensional object in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. In particular, this technique defines a joint probability distribution p_{ij} symmetric and proportional to the similarity between the datapoints:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

where n is the number of samples and the joint probability $p_{j|i}$ is defined as

$$p_{j|i} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|x_i - x_k\right\|^2 / 2\sigma_i^2\right)}.$$

x_i and x_j are datapoints and σ_i is the variance of the Gaussian that is centered on datapoint x_i. Here, Euclidean distances between points are converted to conditional probabilities to represent similarities. Then, in order to map high-dimensional points in a low-dimensional space a second joint probability is defined as

$$q_{ij} = \frac{\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \left\|y_k - y_l\right\|^2\right)^{-1}},$$

where y_i and y_j are two points of the space with low dimensionality. For this joint probability a Student's t-distribution is used with one degree of freedom instead of the Gaussian. This choice depends on the Student's t-distribution property of better modeling the dissimilarity between distant objects through its heavier tails.

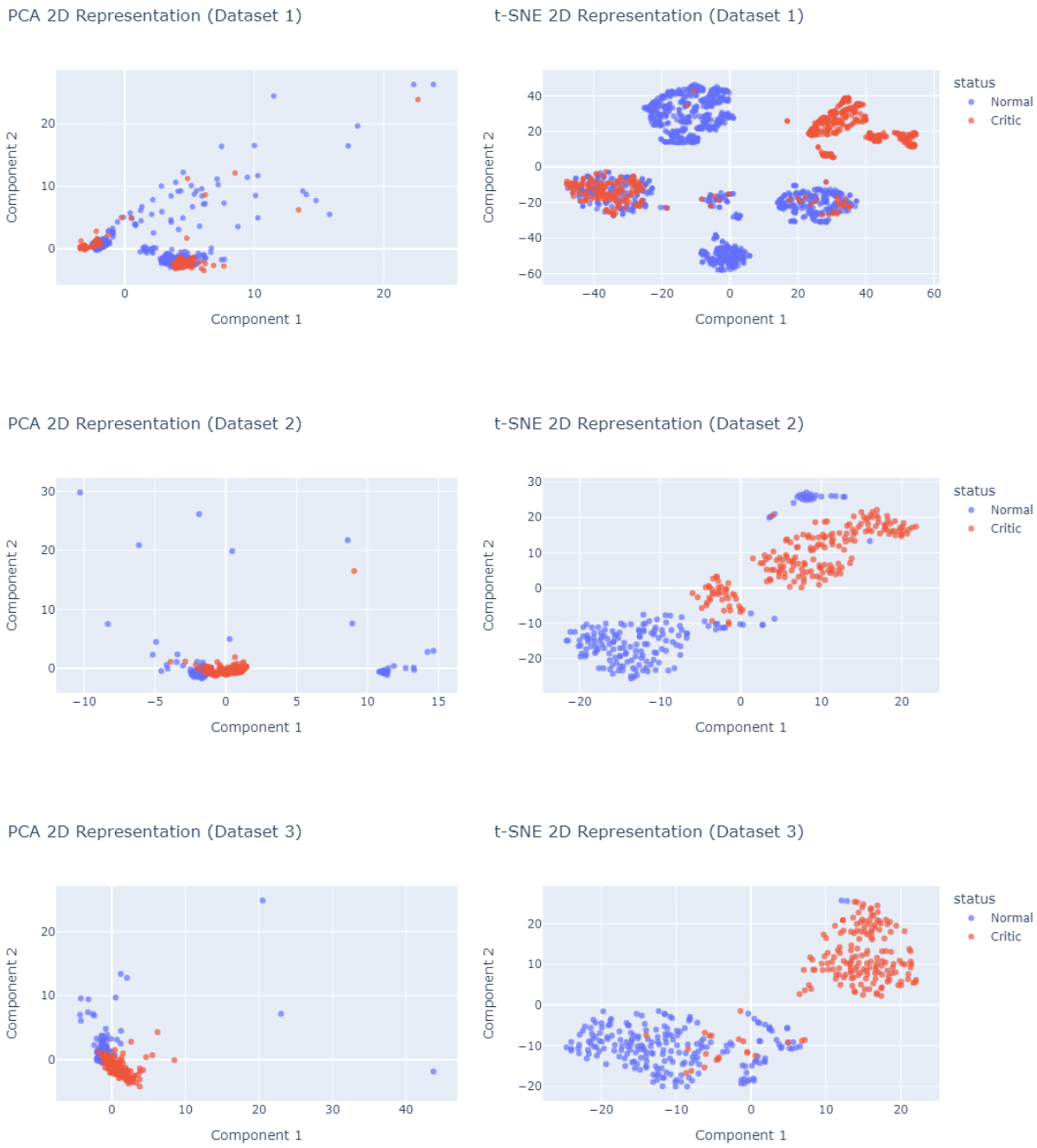


Figure 4.3: PCA and t-SNE of datasets 1, 2 and 3.

Finally, using the gradient descent, the position of the points in the low dimensional space is calculated by minimizing the Kullback-Leibler divergence between Q and P distributions:

$$KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

In conclusion, the t-SNE produces a low-dimensional map that tends to preserve the similarities between the points in the high-dimensional space. Applying this technique to the project's datasets yielded the results in Figure 4.3. In dataset 1, t-SNE is able to isolate in a clear way some clusters.However, there is one cluster containing the two overlapping classes, labeled as critic but with normal values. Probably, the samples labeled as critic just after the tampering are very similar to the normal samples, this could mean that the tampering does not have an immediate effect on the variables. As in the previous case, with dataset 2 and 3, it is possible to identify two clusters, so, the use of simple classifiers seems appropriate.

4.4 Clustering models

The main objective of this section is to identify what is the best number of clusters to represent our datasets with unsupervised learning. Again, dataset 4 with all normal samples is excluded from this part. After the application of unsupervised models, in order to evaluate how the normal and critic samples are divided, the clusters obtained have been evaluated with the V-measure metric using the labels of the `status` variable. V-measure is the weighted harmonic mean of two metrics: homogeneity and completeness. The former is satisfied when clustering assigns only those datapoints that are members of a single class to a single cluster, the latter, symmetrical to the former, is satisfied when a clustering assigns all of those datapoints that are members of a single class to a single cluster [15].

In order to get an intuitive idea, the two diagrams in Figure 4.4 show an example of perfect homogeneity and perfect completeness.

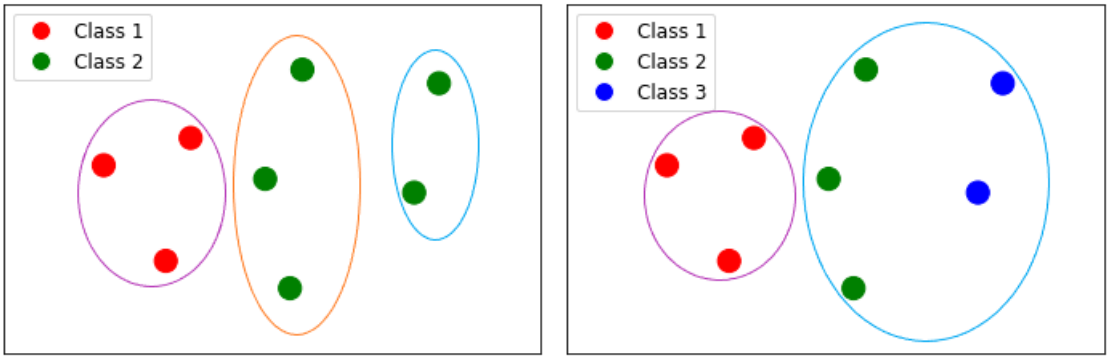


Figure 4.4: Perfect homogeneity and completeness examples. On the left, there is a perfect homogeneity, in fact in each cluster, represented by circles, the data points belong to the same class. However, there is not a perfect completeness because not all data points of the same class belong to the same cluster. On the right, the opposite case is presented. There is perfect completeness because all data points of the same class belong to the same cluster but there is no perfect homogeneity because a cluster contains data points belonging to two classes [3].

In a more rigorous manner, as described by Rosenberg and Hirschberg [15], if N represents the data points, C the classes, K the clusters and a_{ck} the number of data points belonging to the class c and cluster k, the homogeneity is given by

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases},$$

where

$$H(C \mid K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}}$$

and

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n}.$$

The completeness is defined as

$$c = \begin{cases} 1 & \text{if } H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases},$$

where

$$H(K \mid C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{k=1}^{|K|}a_{ck}}$$

and

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{n}\log\frac{\sum_{c=1}^{|C|}a_{ck}}{n}.$$

Finally, the V-measure is given by

$$V_\beta = \frac{(1+\beta)\times h \times c}{(\beta \times h) + c},$$

in which β is the weight ratio attributed to homogeneity versus completeness.

4.4.1 K-Means

K-means clustering is a method that exploits an iterative refinement technique and aims to partition the samples into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centroid), serving as a prototype of the cluster. In more detail, Hastie et al. [8] describe this method starting by defining the squared Euclidean distance as the dissimilarity measure:

$$d\left(x_i, x_{i'}\right) = \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 = \left\|x_i - x_{i'}\right\|^2,$$

then, the within-point scatter is defined by

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \left\|x_i - x_{i'}\right\|^2 = \sum_{k=1}^{K} N_k \sum_{C(i)=k} \left\|x_i - \bar{x}_k\right\|^2,$$

where $\bar{x}_k = (\bar{x}_{1k}, ..., \bar{x}_{pk})$ is the mean vector associated with the kth cluster and $N_k = \sum_{i=1}^{N} I(C(i) = k)$. In short, as defined by Hastie et al. [8], "the criterion is minimized by assigning the N observations to the K clusters in such a way that within each cluster the average dissimilarity of the observations from the cluster mean, as defined by the points in that cluster, is minimized". The iterative algorithm to solve the problem

$$C^* = \min_{C} \sum_{k=1}^{K} N_k \sum_{C(i)=k} \left\|x_i - \bar{x}_k\right\|^2$$

can be found by considering that for any set of observations S

$$\bar{x}_S = \operatorname*{argmin}_{m} \sum_{i \in S} \left\|x_i - m\right\|^2. \tag{4.1}$$

Finally, we can find C^* by solving the following optimization problem

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} \left\|x_i - m_k\right\|^2. \tag{4.2}$$

The minimization of the problem can be obtained by applying an alternating optimization procedure given by the algorithm Algorithm 4.1

Figure 4.5 shows some of the algorithm iterations for simulated data with three classes. The centroids are depicted by "O"s and the straight lines show the partitioning of points. Each sector is the set of points closest to each centroid.

Algorithm 4.1 K-means Clustering [8].

1. For a given cluster assignment C, the total cluster variance (4.2) is minimized with respect to $\{m_1, \ldots, m_k\}$ yielding the means of the currently assigned clusters (4.1).

2. Given a current set of means $\{m_1, \ldots, m_k\}$, (4.2) is minimized by assigning each observation to the closest (current) cluster mean. That is,

$$C(i) = \underset{1 \leq k \leq K}{\operatorname{argmin}} \|x_i - m_k\|^2 .$$

3. Steps 1 and 2 are iterated until the assignments do not change.

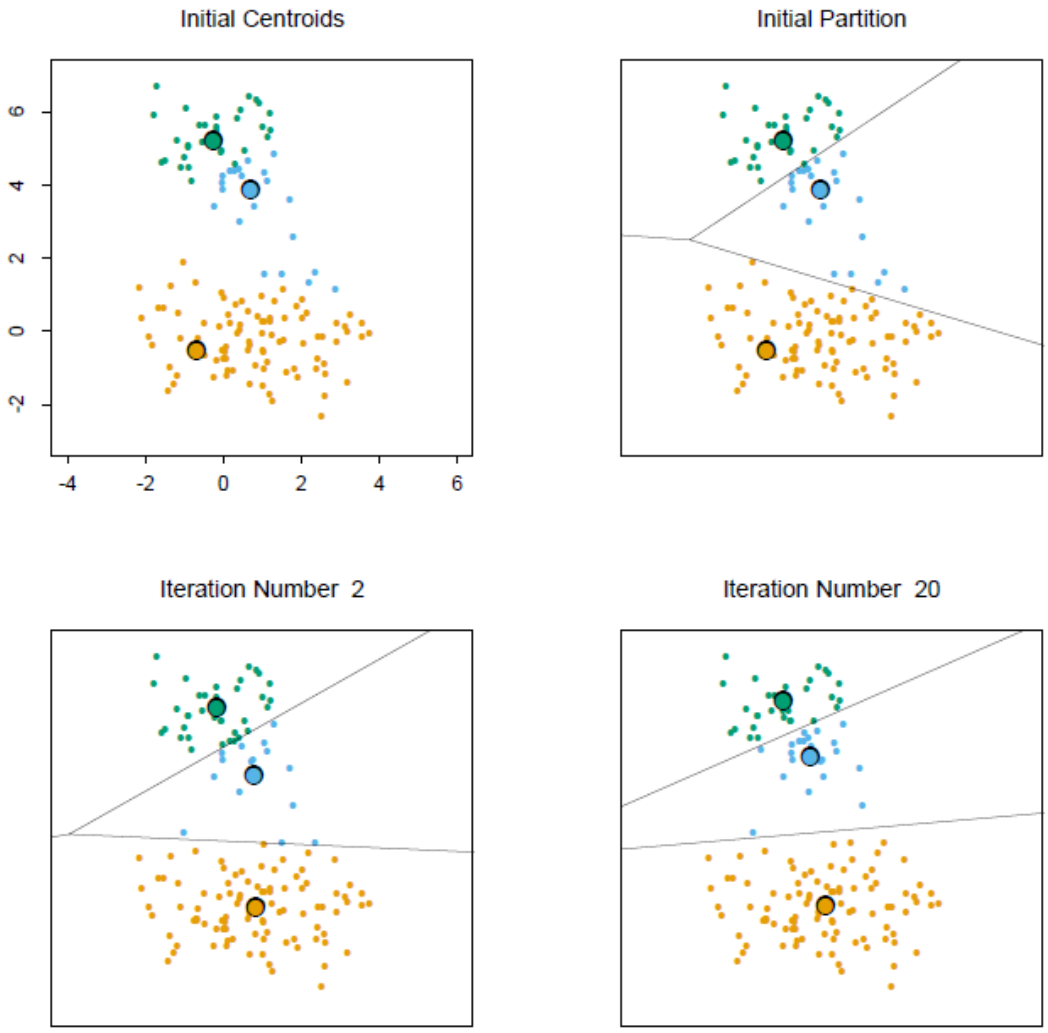


Figure 4.5: Example of successive iterations of the K-means clustering algorithm with simulated data [8].

Clusters	V-measure
2	0.0006
3	0.0042
4	0.3964

Clusters/status	1	2	Total
0 (normal)	655	359	1014
1 (critic)	343	166	509

Clusters/status	1	2	3	Total
0 (normal)	654	327	33	1014
1 (critic)	343	161	5	509

Clusters/status	1	2	3	4	Total
0 (normal)	33	0	323	658	1014
1 (critic)	5	317	161	26	509

Table 4.3: K-means clustering analysis dataset 1.

From the previous description it is clear that the most important hyperparameter is the number of clusters. For the datasets of this project, although this hyperparameter can be selected using a heuristic called elbow method, all possible clusterization with values from 2 to 10 have been evaluated. Below, there are the results with two, three and the best number of clusters.

1. The clustering analysis of dataset 1 is represented by the Table 4.3.

   There are no two clearly separated clusters. Best V-measure score is obtained with 4 clusters, theoretically this could justify the implementation of more complex models.

2. The clustering analysis of dataset 2 is represented by the Table 4.4.

   There is no clear division between clusters but with 4 clusters we have a good result, in particular with cluster 2 and 5.

3. The clustering analysis of dataset 3 is represented by the Table 4.5.

   With 3 clusters we have excellent results. It's important to note that there is a cluster with only three samples. In this case it is possible to talk about a clear distinction between classes.

4.4.2 Hierarchical Clustering

Hierarchical Clustering is a clustering method that attempts to create a cluster hierarchy. There are two strategies: agglomerative (bottom-up approach) and divisive (top-down approach). The approach used is the agglomerative

Clusters	V-measure
2	0.0841
3	0.1042
5	0.4842

Clusters/status	1	2	Total
0 (normal)	152	23	175
1 (critic)	194	1	195

Clusters/status	1	2	3	Total
0 (normal)	5	148	22	175
1 (critic)	1	194	0	195

Clusters/status	1	2	3	4	5	Total
0 (normal)	5	147	19	3	1	175
1 (critic)	1	41	0	0	153	195

Table 4.4: K-means clustering analysis dataset 2.

Clusters	V-measure
2	0.0129
3	0. 7091
4	0.6648

Clusters/status	1	2	Total
0 (normal)	3	208	211
1 (critic)	0	179	179

Clusters/status	1	2	3	Total
0 (normal)	3	208	0	211
1 (critic)	0	23	156	179

Clusters/status	1	2	3	4	Total
0 (normal)	11	198	2	0	211
1 (critic)	0	23	0	156	179

Table 4.5: K-means clustering analysis dataset 3.

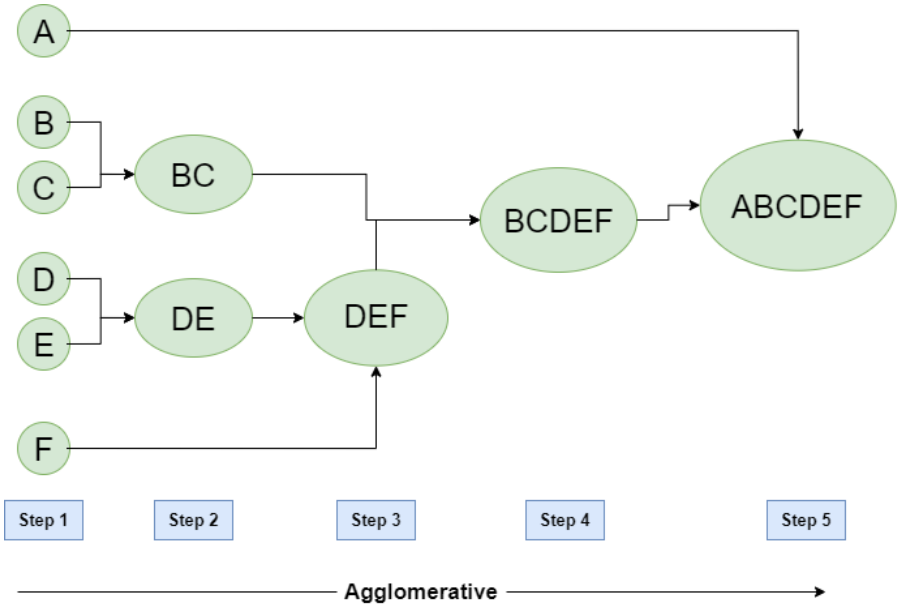


Figure 4.6: Schematic representation of Agglomerative Hierarchical Clustering [4].

one in which each observation starts in its own cluster, then the pairs of clusters are progressively joined up in the hierarchy (Figure 4.6).

In order to do this, a measure of dissimilarity between clusters called linkage must be defined. There are four main types of linkage:

- Single linkage (SL):
$$d_{SL}(G,H) = \min_{\substack{i\in G \\ i'\in H}} d_{ii'},$$
- Complete linkage (CL):
$$d_{CL}(G,H) = \max_{\substack{i\in G \\ i'\in H}} d_{ii'},$$
- Group average (GA):
$$d_{GA}(G,H) = \frac{1}{N_G N_H} \sum_{i\in G} \sum_{i'\in H} d_{ii'},$$
- Ward (W):
$$d_W(G,H) = \frac{N_G \times N_H}{N_G + N_H} \|m_G - m_H\|^2.$$

G and H represent two groups, $d_{ii'}$ is the pairwise observation dissimilarities where i belongs to G and i' to H. N is the number of points in a cluster and m represents the center of the cluster.

For the purposes of this project, to find the best combination of hyperparameters a Grid Search has been performed with:

1. Number of clusters: range from 2 to 10;

Clusters/status	1	2	3	4	Total
0 (normal)	0	38	322	654	1014
1 (critic)	315	5	161	28	509

Table 4.6: Hierarchical Clustering analysis dataset 1.

Clusters/status	1	2	3	4	5	Total
0 (normal)	5	4	147	18	1	175
1 (critic)	1	0	46	0	148	195

Table 4.7: Hierarchical Clustering analysis dataset 2.

Clusters/status	1	2	3	Total
0 (normal)	208	3	0	211
1 (critic)	21	0	158	179

Table 4.8: Hierarchical Clustering analysis dataset 3.

2. Distance type to compute the linkage: Euclidean and Manhattan;
3. Linkage criterion to evaluate the distances between sets of observations: "Ward", "Complete", "Group average" and "Single".

In the Table 4.6, Table 4.7 and Table 4.8 the best results are presented.

1. The best hyperparameters for the clustering of dataset 1 are: 4 clusters, Euclidean distance and Ward linkage criterion. The V-measure is 0.3898. The distribution of the samples in the clusters is shown in Table 4.6. Also, in this case there are no two clearly separated clusters. Theoretically this could justify the implementation of more complex models.
2. The best hyperparameters for the clustering of dataset 2 are: 5 clusters, Euclidean distance and Ward linkage criterion. The V-measure is 0.4598. The distribution of the samples in the clusters is shown in Table 4.7. The best number of clusters is still 5 but with a lower V-measure than that obtained with K-means.
3. The best hyperparameters for the clustering of dataset 3 are: 3 clusters, Euclidean distance and Ward linkage criterion. The V-measure is 0.7253. The distribution of the samples in the clusters is shown in Table 4.8. As in the case of K-means, with this dataset we get the best results with a clear distinction between classes.

5 Methods

5.1 Classification techniques

The project task consists of a binary classification to determine the status of the packaging machine. Considering the availability of the `status` label, the learning of the model is supervised. In a more formal way, let x_i be the feature vector of the ith samples and y_i the respective target value. The feature vectors take values in X, a subset of R^k, where k is the number of variables. Instead, the target values belong to Y, which in the case of a binary classification, is a subset of R^2.Considering this formalism, the learning phase of the model tries to find a function f that associates each vector in X with its respective target value in Y. Obviously, f is unknown, so the goal of the model becomes to find a function h, belonging to the hypotheses space H, which best approximates f

$$h \approx f : X \to Y$$

Once a class of models has been selected, to best adapt the model to the specific problem, it is necessary to fine-tune the hyperparameters. Among the strategies to find the best combination of hyperparameters we find the Grid Search and the Random Search. The first considers a list of values for each hyperparameter and tries to find the best set of hyperparameters by trying all possible combinations. However, faced with a large number of hyperparameters, this strategy becomes time prohibitive. In this case the Random Search becomes a valid alternative. Unlike the previous approach, this strategy randomly extracts only some of the possible combinations. This is a particularly effective strategy when some hyperparameters are more important than others.

For the purposes of this project, a Random Search has been used for all models, with the exception of the Decision Tree for which a Grid Search has been used. In the case of the GAN model, for computational cost reasons, a manual selection of the hyperparameters has been made without applying a Grid or Random Search.

These strategies have been used in combination with a k-fold Cross Validation: a resampling procedure to

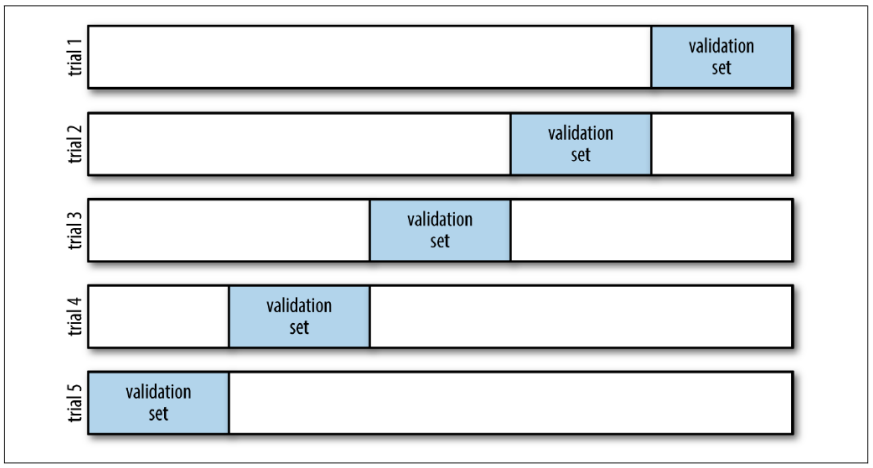


Figure 5.1: Schematic representation of 5-fold cross validation [20].

evaluate a model offering a less optimistic estimate of the model skill than a simple train/test split. In particular, the dataset is divided into k groups, in turn each group is used as a validation set, while the other groups are used as a training set. The ability of the model is evaluated by averaging the results obtained on the validation sets. For all the models implemented in this thesis, a 5-fold CV (Figure 5.1) has been used in order to have a good compromise between computational cost and evaluation of the robustness.

Before defining the metrics used, it must be specified that the samples belonging to the normal class (labeled with 0) are identified as actual negative, those belonging to the critic class (labeled with 1) are actual positive. The possible outcomes for a binary classification task are the following:

- True Positives (TP): the model correctly predicts the positive class;
- True Negatives (TN): the model correctly predicts the negative class;
- False Positives (FP): the model incorrectly predicts the positive class;
- False Negatives (FN): the model incorrectly predicts the negative class.

Starting from these definitions, each model has been evaluated considering different metrics:

$$\begin{aligned}
\text{accuracy} &= \frac{TP+TN}{TP+FP+TN+FN}, \\
\text{precision} &= \frac{TP}{TP+FP}, \\
\text{sensitivity/recall} &= \frac{TP}{TP+FN}, \\
\text{specificity} &= \frac{TN}{TN+FP}, \\
\text{balanced accuracy} &= \frac{\text{sensitivity} + \text{specificity}}{2}, \\
F_1\text{-score} &= 2 \times \frac{\text{precision} \times \text{sensitivity}}{\text{precision} + \text{sensitivity}},
\end{aligned}$$

$$\text{Matthews Correlation Coefficient (MCC)} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}.$$

However, the reference metric used in model selection was the balanced accuracy. This choice is due to the fact that the classes are unbalanced. Using accuracy would have benefited the models inclined to predict the most present class, leading to false positives, that is, classifying the critical samples as normal.

The following sections describe some classifiers used as a comparison for GAN model. Finally, the GAN architecture and its adaptation to the classification task are described.

5.1.1 Logistic Regression

The basic model in a classification task is certainly the Logistic Regression. The concept behind this model begins by defining the procedure behind a simple linear regression model

$$h_\theta(x_i) = \theta^{\mathrm{T}} x_i,$$

where, as defined above, x_i defines the feature vector of sample i and θ the parameter vector. Now it is necessary to define a decision rule for the binary classification task: if $h_\theta(x)$ is larger than a certain threshold θ_0, than the model predicts class 1 as output, otherwise class 0. In order to obtain the Logistic Regression model, the next step is to apply the logistic function to the vector product $h_\theta(x)$:

$$h_\theta(x_i) = \frac{1}{1 + e^{-\theta^{\mathrm{T}} x_i}}.$$

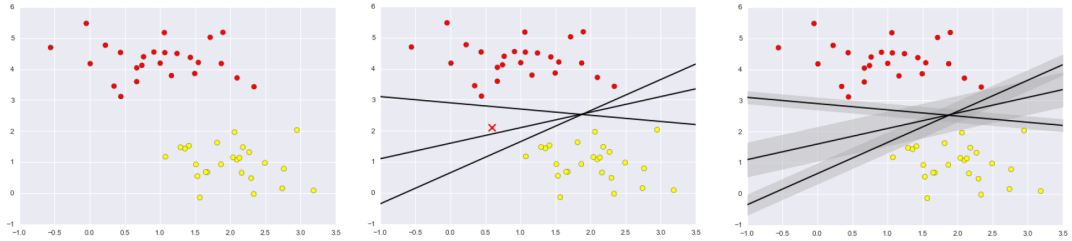
The model output takes values in the interval [0, 1] and it can be interpreted as the probability of sample i to belong to class 1. The learning phase for estimating the vector of parameters θ requires the definition of a loss function. Typically, the Cross Entropy loss is used

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \mathrm{y}_i \cdot \log\left(h_\theta(x_i)\right) + (1 - \mathrm{y}_i) \cdot \log\left(1 - h_\theta(x_i)\right),$$

therefore, the problem to be solved is the minimization of this loss function with respect to θ. In order to minimize $J(\theta)$ different algorithms can be used, for example the Coordinate Descent algorithm or some variants of the Gradient Descent such as the Stochastic Average Gradient Descent.

5.1.2 Support Vector Machine

The Support Vector Machines (SVM) are a class of supervised algorithms for both regression and classification. Clearly, this paragraph will describe the application in the second case. In order to understand how the model works, it is necessary to begin by providing a simple example of classification with two distinct classes in a two-dimensional space as shown in Figure 5.2 (a).



(a) (b) (c)

Figure 5.2: (a) Graphical representation of a dataset with two distinct classes in a two-dimensional space.(b) Three perfect linear discriminative classifiers for our data.(c) Visualization of "margins" within discriminative classifiers [20].

Although it is easy to find a model that can draw a hyperplane, in this case a straight line, between the two classes, a problem arises: there are multiple lines that can divide the two classes (Figure 5.2 (b)). Therefore, a new sample in the test set would be labeled with different labels depending on the dividing line considered, as in the case of the red "X". One way to overcome this limitation is to use Support Vector Machines. The idea is to use a dividing line with a margin of a certain thickness from the closest points (Figure 5.2 (c)).

Hence, the original problem now becomes to find the line that maximizes the margin. Formally, given a training set T with two classes $A = \{x_i : (x_i, y_i) \in T, y_i = 1\}$ and $B = \{x_i : (x_i, y_i) \in T, y_i = -1\}$, the separation margin M of the hyperplane $H = \left\{x \in \mathbb{R}^k : \theta^T x + \theta_0 = 0\right\}$ is the minimum distance between samples in the training set T and the hyperplane H. Its value is

$$M(\theta, \theta_0) = \min_{x_i \in A \cup B} \left\{ \frac{\left|\theta^T x_i + \theta_0\right|}{\|\theta\|} \right\}.$$

It is important to note that only the data samples close to the hyperplane define the best separation margin. These points are called support vectors, hence the name of the model.

At this point the problem can be written in this way

$$\begin{aligned} \max_{\theta, \theta_0} \quad & M(\theta, \theta_0) \\ \text{s.t.} \quad & \theta^T x_i + \theta_0 \geq 1, \quad x_i \in A, \\ & \theta^T x_i + \theta_0 \leq -1, \quad x_i \in B. \end{aligned}$$

This is a convex quadratic minimization problem with $\frac{1}{2}||\theta||^2$ as objective function.

So far, the case in which classes are linearly separable has been discussed, which is usually uncommon. Now, we will explore the case in which the classes are separable but not linearly, as in the example in the Figure 5.3 (a). In order to solve this new problem, it is necessary to combine SVM with the kernels: the data samples are projected into higher-dimensional space through a kernel function K defined as

$$K(x_i, x_j) = \varphi(x_i)^t \varphi(x_j),$$

where x_i and x_j are two samples and φ is the projection function. The following are the most used kernel types:

- Linear Kernel:
$$K\left(x_i, x_j\right) = x_i^t x_j;$$

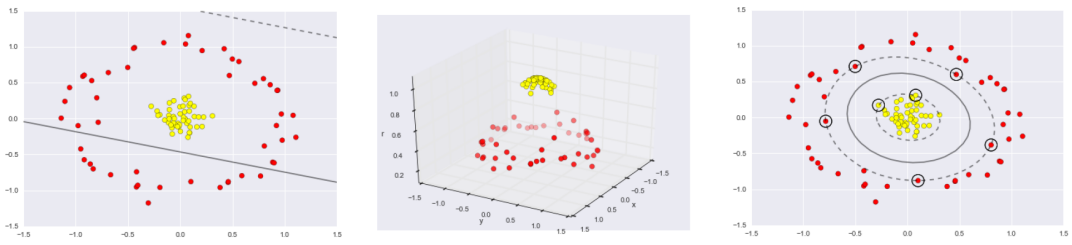
- Polynomial Kernel:
$$K\left(x_i, x_j\right) = \left(\gamma x_i^t x_j + r\right)^d, \gamma > 0, r \geq 0, d \in \mathbb{N}, d \geq 1;$$

- Gaussian Kernel, also known as Radial Basis Function (RBF):
$$K\left(x_i, x_j\right) = \exp\left(-\gamma \left\|x_i - x_j\right\|^2\right), \gamma > 0;$$

- Sigmoid Kernel:
$$\tanh\left(\gamma \cdot x_i^T x_j + r\right), \gamma > 0.$$

γ represents the intensity of the kernel used, in fact for small values of γ, the model tends to behave like a linear SVM, instead with large values the radius of the area of influence of the support vectors only includes the support vector itself.

Figure 5.3 (b) and (c) show how an SVM with kernel is able to separate a non-linear dataset by increasing the dimensionality.



(a) (b) (c)

Figure 5.3: (a) With nonlinear data, a linear classifier cannot divide the classes.(b) Adding a third dimension to facilitate the division of classes.(c) Example of how a kernel SVM can fit the data [20].

In order to complete the description of this class of models, it is necessary to describe the more general case in which the data samples are not separable even with the help of kernels. In this case the SVM is implemented by adding a tunable factor, usually denoted by C, which "softens" the margin allowing some data points to go beyond the margin if this allows for a better fit. For large values of C, the margin is less permeable, on the other hand, with small values, the margin can include some points. The new minimization problem becomes

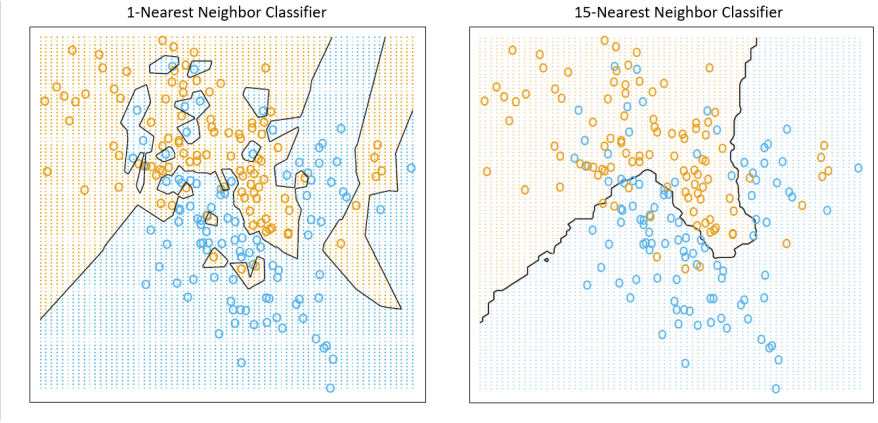


Figure 5.4: Example of K-Nearest Neighbors with k equal to 1 and 15 [8].

$$\begin{aligned} \min_{\theta,\theta_0,\xi} \quad & \frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & \theta^\top x_i + \theta_0 \geq 1 - \xi_i, \quad x_i \in A, \\ & \theta^\top x_i + \theta_0 \leq -1 + \xi_i, \quad x_i \in B. \quad (i = 1, \ldots, n), \end{aligned}$$

where ξ_i measures how much a data point can violate the margin.

5.1.3 K-Nearest Neighbors

The model classes seen so far require a parameter optimization process. K-Nearest Neighbors (KNN) on the other hand, falls into the class of non-parametric models in which there is no parameter estimation. With this class of models given a query point x_i, the k training points closest in distance to x_i are searched, the label of x_i is assigned through the majority of the votes of the k neighbors. In this case, there are two hyperparameters to consider: the distance metric and the value of k. Regarding the former, among the main metrics there are Euclidean, Manhattan, Chebyshev and Minkowski. The number of neighbors, on the other hand, controls the degree of smoothing of the decision boundary, in fact for small values of k many small areas are produced, while with large values a few large areas. The Figure 5.4 shows an example of classification of a two-dimensional dataset with two values of k.

The main disadvantage of this class of models is to calculate a number of distances equal to the number of data points in order to find the nearest k points. In this way, the computational cost can become unaffordable. In order to overcome this limit, a non-exhaustive search of the closest neighbors is carried out through tree-based search structures which offers an approximate solution.

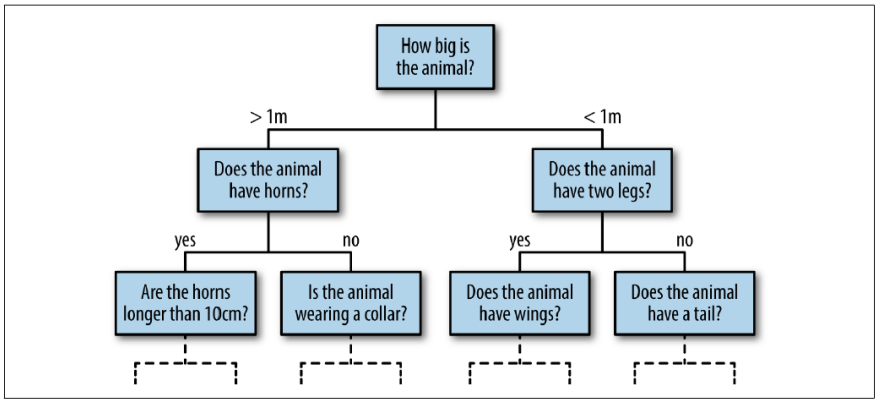


Figure 5.5: Example of a binary Decision Tree for classifying animals [20].

5.1.4 Decision Tree and Random Forest

Decision Trees are another supervised non-parametric model that tries to predict the class of a target variable from simple decision rules. These rules, which define the model, recursively partition the dataset through the values assumed by the variables until a certain condition is satisfied. The assignment of the label of a new sample takes place by making it go through the sequence of rules that define the model until it reaches the end of the Decision Tree. At this point, the label assigned to the new sample is chosen through a majority voting system among all the labels of the training samples that followed the same sequence of rules.

An example of a Decision Tree for classifying animals might be like the one shown in Figure 5.5.

A fundamental element in the implementation of the algorithm is the criterion by which the variables are selected at each split step. One of the most common is Entropy, an index that measures the degree of homogeneity of a certain set:

$$\text{Entropy}(S) = \sum_{c \in C} -p(c) \cdot \log_2(p(c))$$

here, S represent the set of training sample, C is the set of classes in S, and $p(c)$ is the fraction of elements in S belonging to class c. In this case, Entropy is used to define how homogeneous the set of samples is with respect to the target variable before the next split. In simple terms, we try to choose the decision rule that is best able to classify the samples with respect the target variable. This is done through a quantity belonging to the information theory, the Information Gain (IG). This quantity defines the change in Entropy of the set S when it is divided through the variable a.

$$IG(S, a) = \text{Entropy}(S) - \text{Entropy}(S \mid a) = \text{Entropy}(S) - \sum_{T \in \mathcal{T}} p(T)\,\text{Entropy}(T),$$

where $Entropy(S|a)$ is the Entropy computed as a weighted sum of T that is the list of subsets obtained after the splitting. Here, $p(T)$ indicates the fraction of data samples of S that after the split belongs to T. The idea is to compare the information gain values for each variable, excluding the variables already used previously. The

highest value obtained provides the variable to use to divide the current set.

Another quantity to define the choice of the variable to perform the splitting is the Gini Index:

$$\text{Gini}(S) = 1 - \sum_{c \in C} p(c)^2.$$

The value assumed by this index is equal to zero when all the samples belong to the same class, that is in the case of maximum homogeneity. On the other hand, when the classes are heterogeneous, the index value is greater than zero.

In conclusion, Decision Trees provide several advantages, here are some of them:

- Easy interpretation possibility of being viewed.
- The cost in the forecast is logarithmic in the number of data points used to train the tree.
- It does not require large data preparation such as normalization and creation of dummy variables.
- It is possible to manage both numeric and categorical variables.

However, they also have some disadvantages:

- Decision Trees can create overly complex trees unable to generalize on new data, a problem called overfitting. Some countermeasures reside in pruning, in setting the minimum number of samples required at a leaf node or in setting the maximum depth of the tree.
- These models can be unstable. Small variations in the data can cause the model to generate completely different trees. By using Decision Trees within an ensemble method this problem can be limited.
- In the case of unbalanced datasets, the model generates biased trees. In order to avoid this problem, it is necessary to balance the dataset.

As mentioned previously, ensemble methods can be used to improve the performance of this model class. One of these is Bootstrap Aggregating, also called Bagging. With this approach, B subsets are randomly extracted with repetition from the training set and a Decision Tree is fitted on each of them. The final output is defined by the value of the most frequent label among the B outputs obtained.

A development of this approach consists in the use of multiple Decision Trees but decorrelated to each other. In short, the Decision Trees are fitted on the B subsets with the difference that in this case a random subset of m variables is used to fit the models. This approach is called Random Forest.

5.2 Deep Learning models

In the following section, two DL models will be described: a Feed Forward Neural Network and a Generative Adversarial Network. For the first model the main concepts will be described, while for the second a more detailed description will be offered.

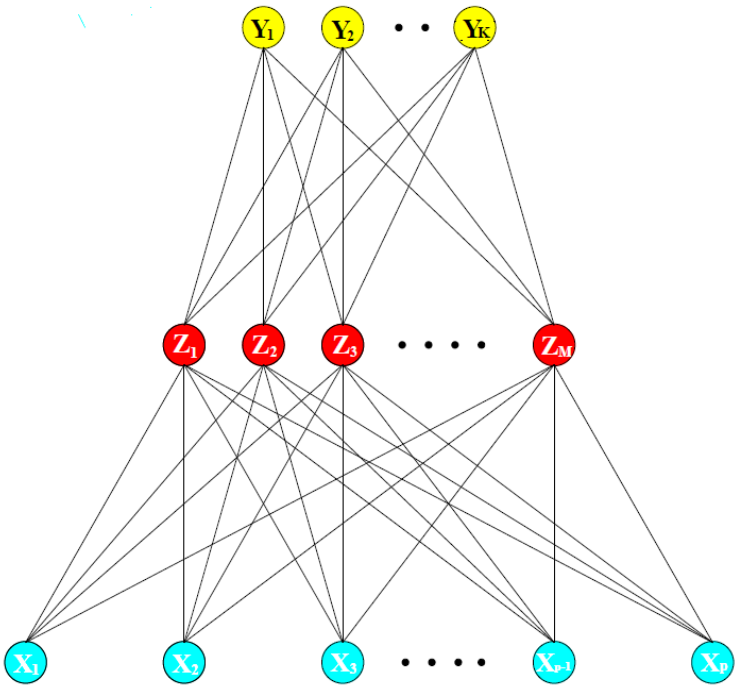


Figure 5.6: Schematic representation of a single hidden layer FFNN [8].

5.2.1 Feed Forward Neural Network

A Feed Forward Neural Network (FFNN) is the type of Neural Network with the simplest architecture. It has a layered structure made up of small individual units called neurons. The learning occurs by alternating a forward and a backward step.

The forward step begins with the first layer neurons taking the feature vector as input and producing the scalar product to which a differentiable function called activation function is applied. This is repeated for all subsequent layers taking the output obtained from the previous layer. These layers are called hidden layers. The last layer, called output layer, provides the final result.

The backward step takes place through a Gradient Descent Algorithm called Backpropagation. It updates the weights of the Neural Network through the backward propagation of the final output error compared with the ground truth. From this derives the name of the algorithm. The Figure 5.6 provides a schematic representation of a FFNN.

5.2.2 Generative Adversarial Network

A Generative Adversarial Network is a generative model introduced by Goodfellow et al. in 2014 [7] that exploits the principles of supervised training through the use of two competing sub models. They are usually DL models, and in the simplest case they are FFNN. One of the sub models, the generator, is trained to produce new realistic

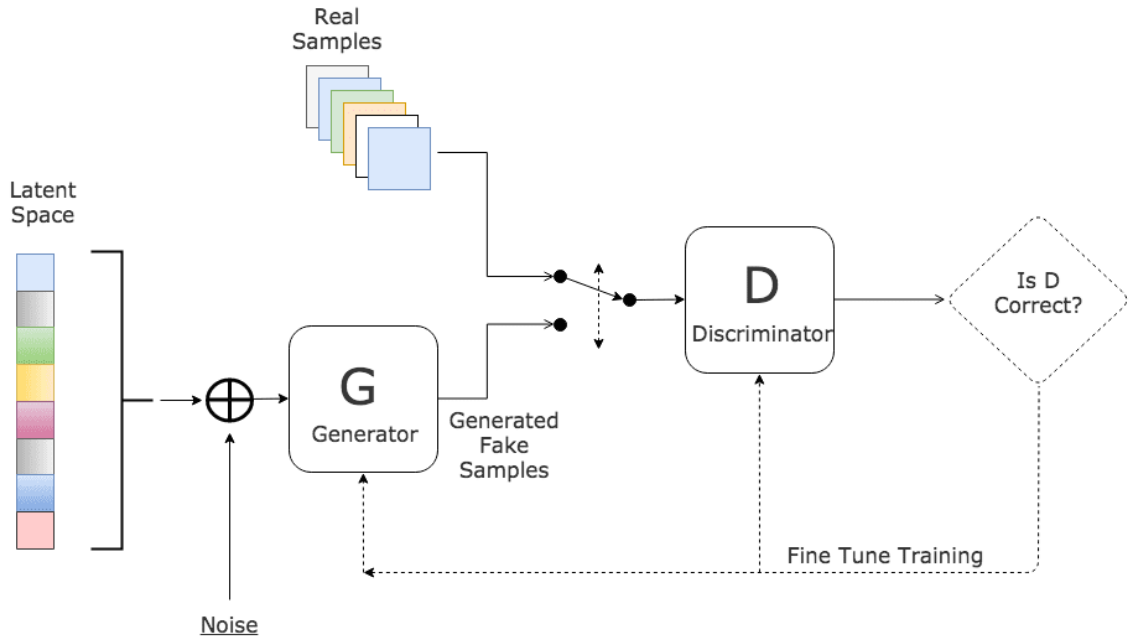


Figure 5.7: Schematic representation of the architecture of a GAN [5].

samples, while the other one, the discriminator, tries to distinguish between real (data) or fake (generated). The model training occurs concurrently in a zero-sum game until an equilibrium state is reached where discriminator provides correct outputs half the time and generator produces realistic samples.

The input of the generator, called latent space, is a vector randomly extracted from a Gaussian distribution of fixed length, the output instead has the same dimensionality as the real samples. The input of the discriminator is a sample, real from the data or generated by the generator, while the output is a binary class label of real or fake (generated).

The Figure 5.7 shows the basic architecture of a Generative Adversarial Network.

Formally, to describe the learning process it is necessary to define some concepts. For what concerns the generator, $p_z(z)$ is noise distribution which provides the input to the generator which has the purpose of learning the p_g distribution on the x data. G is a differentiable function represented by a Neural Network with parameters θ_g. So, $G(z, \theta_g)$ represents the mapping to data space.

The discriminator, also represented by a Neural Network and defined by $D(x, \theta_d)$, outputs a value that indicates the probability that a sample is real or generated given a real or generated input.

The two models play a two-player minimax game with the following value function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))].$$

The discriminator D tries to maximize the probability of assigning the correct label to both training examples x and samples from $G(z)$, simultaneously the generator G tries to minimize the probability that D assigns the "fake" label to $G(z)$.

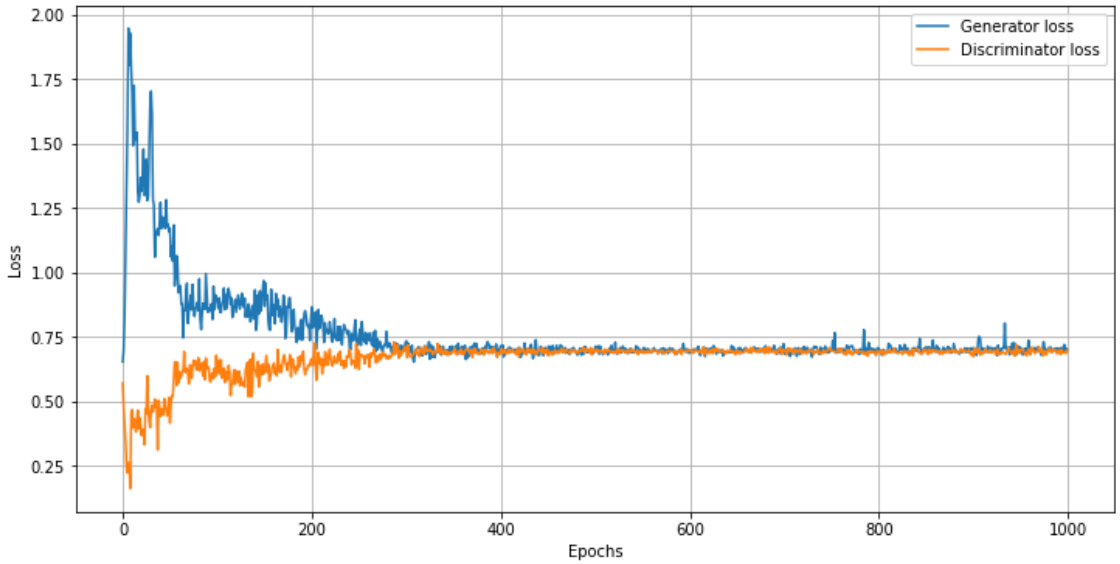


Figure 5.8: Example of equilibrium between generator and discriminator.

Given the nature of learning, it is possible to apply Backpropagation: θ_d parameters of discriminator D are updated by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right]$$

Instead, θ_g parameters of generator G are updated by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

The game ends when a state of equilibrium is reached, a saddle point with the achievement of the minimum with respect the strategy of one player and the maximum with respect the strategy of the other one (Figure 5.8). In other words, the generator produces samples very similar to the real data and the discriminator predicts "real" or "fake" with probability 0.5.

It is interesting to note that to reach the state of equilibrium it is possible to train the discriminator and generator by modifying the weights by alternating one step for each model (at each epoch the weights of the two networks are updated once), or to train one of the two models several times for each epoch (for example, train the discriminator 3 times and the generator 1 time) [6].

Returning to the project, the model must be adapted to a classification task. The basic idea is to train the model only with normal samples in order to induce the discriminator to learn the distribution underlying the variables of the standard operation of the machine. The generator has the task of generating realistic normal samples and the discriminator must distinguish the generated samples from the real normal ones. After the training phase the discriminator is detached from the model becoming a simple classifier. With a dataset containing normal and

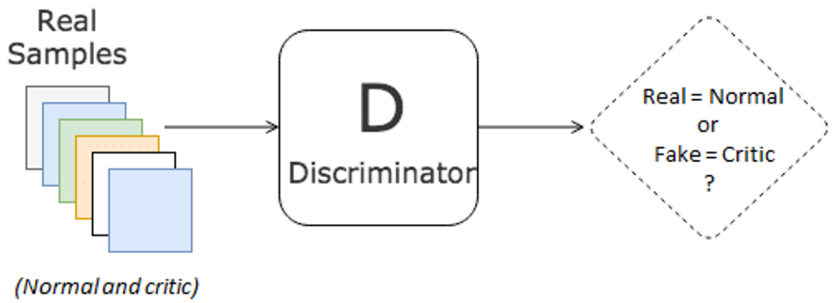


Figure 5.9: Schematic representation of the discriminator during testing phase.

critic samples, the classifier should identify as real the samples with normal dynamics and as fake (generated) those belonging to a non-normal distribution, namely critic.

The Figure 5.7 represents a schematization of the model during the training phase, while the Figure 5.9 during the validation and testing phase.

It is good to remember that for this model the datasets were merged in order to obtain one dataset with all normal and another with all critic. So, for the training and evaluation part, the CV was adapted to the model in the following way: the training folds included only normal samples, instead the validation fold included all the remaining normal samples in the "all normal dataset" and all critic ones. Finally, the test was carried out on dataset 3 as in the case of the other models.

6 Experiments and results

In this chapter we will resume some implementation details of Chapter 4 on pre-processing and the hyperparameters used in classifiers and DL models will be listed. One of the paragraphs will be dedicated to the samples generated by the GAN. Although the project task does not put any focus on the generator, this part can offer a further interpretative tool on the results obtained. Finally, a comparative and interpretative comment on the results will be offered.

6.1 Implementation details

As mentioned above, the datasets provided by the company are four: three were used for training and validation through CV, the fourth for testing, after the final training on the three initial datasets. A summary of the sample division is provided in the following Table 6.1.

For all models, excluding the GAN, the dataset for training and validation is unique. For GAN, on the other hand, two datasets were used, one with all normal, the other with all critic. The Table 6.2 and Table 6.3 list the

	Normal	Critic	Failure scenario
Label	0	1	NA
Proportion	1561 (63,87%)	883 (36,13%)	NA
Training and validation set (CV)	1350 (65,73%)	704 (34,27%)	Valves wear out and Vacuum pump wear out
Test set	211 (54,10%)	179 (45,90%)	Vacuum pump wear out

Table 6.1: Summary of the sample division and failure scenarios.

Model	Hyperparameter
Logistic Regression	inverse of regularization strength
	norm of the penalty
	maximum number of iterations
Support Vector Machine	kernel type
	kernel coefficient γ
	regularization parameter
K-Nearest Neighbors	number of neighbors
	distance metric
Decision Tree	criterion to measure the quality of a split
	splitter
Random Forest	criterion to measure the quality of a split
	number of trees

Table 6.2: List of hyperparameters of the classifiers.

tuning hyperparameters used in the Random Searches, with the exception of the Decision Tree and the GAN, in which in the first case a Grid Search was used, while in the second a manual selection.

The values of the selected hyperparameters are shown in the Appendix.

6.2 Results

This paragraph shows the results of the models selected in the CV and those on dataset 3 used as a test set. In order to carry out the final test, the best models were trained again on the dataset used for CV. It is important to remember that the reference metric considered in the choice of the model is balanced accuracy.

The results of the best models selected with CV are shown in the barplot (Figure 6.1). All the models reach good performances, however the GAN does not reach levels of balanced accuracy comparable with the other models.

As for the test, the Table 6.4 shows how all the models are able to reach performances around 93% in balanced accuracy, while GAN obtains a balanced accuracy around 65%. Furthermore, the high specificity indicates the absence of false positives, namely false alarms. Very important aspect for an alarm system.

In order to better understand the results obtained from an alarm system perspective, it is necessary to consider the output over time. From the horizontal (Figure 6.2)it is possible to see that all the models are able to predict the alarm of the machine well in advance.

All the models are able to predict the alarm after a few minutes of tampering, the GAN shows less good and stable results but as the machine approaches the alarm it identifies the critical situation.

Model	Hyperparameter
Feed Forward Neural Network	number of hidden layers
	number of hidden neurons
	epochs
	type of activation function
	optimizer
	learning rate
	L2 regularization coefficient
	dropout probability
Generative Adversarial Network	number of hidden layers
	number of hidden neurons
	epochs
	number of trainings of each model for each epoch
	type of activation function
	optimizer
	learning rate
	L2 regularization coefficient
	dropout probability (only for discriminator)

Table 6.3: List of hyperparameters of Feed Forward Neural Network and GAN (when not specified, the hyperparameter refers to both the discriminator and the generator).

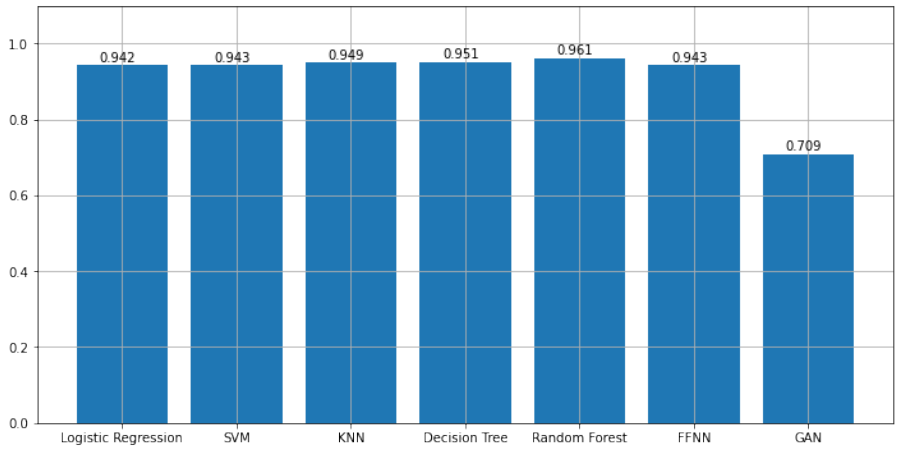


Figure 6.1: Results of the best models selected with CV considering balanced validation accuracy.

Model	Bal. accuracy	Accuracy	F1 score	Precision	Recall	Specificity	MCC
LR	**0.936**	**0.941**	0.931	**1.0**	0.872	**1.0**	**0.886**
SVM	0.927	0.933	0.921	**1.0**	0.855	**1.0**	0.872
KNN	**0.936**	**0.941**	0.931	**1.0**	0.872	**1.0**	**0.886**
DT	**0.936**	**0.941**	**0.932**	0.994	**0.877**	0.995	**0.886**
RF	**0.936**	**0.941**	0.931	**1.0**	0.872	**1.0**	**0.886**
FFNN	0.930	0.936	0.924	**1.0**	0.860	**1.0**	0.877
GAN	0.646	0.674	0.455	0.981	0.296	0.995	0.420

Table 6.4: Metrics of all models on the test set.

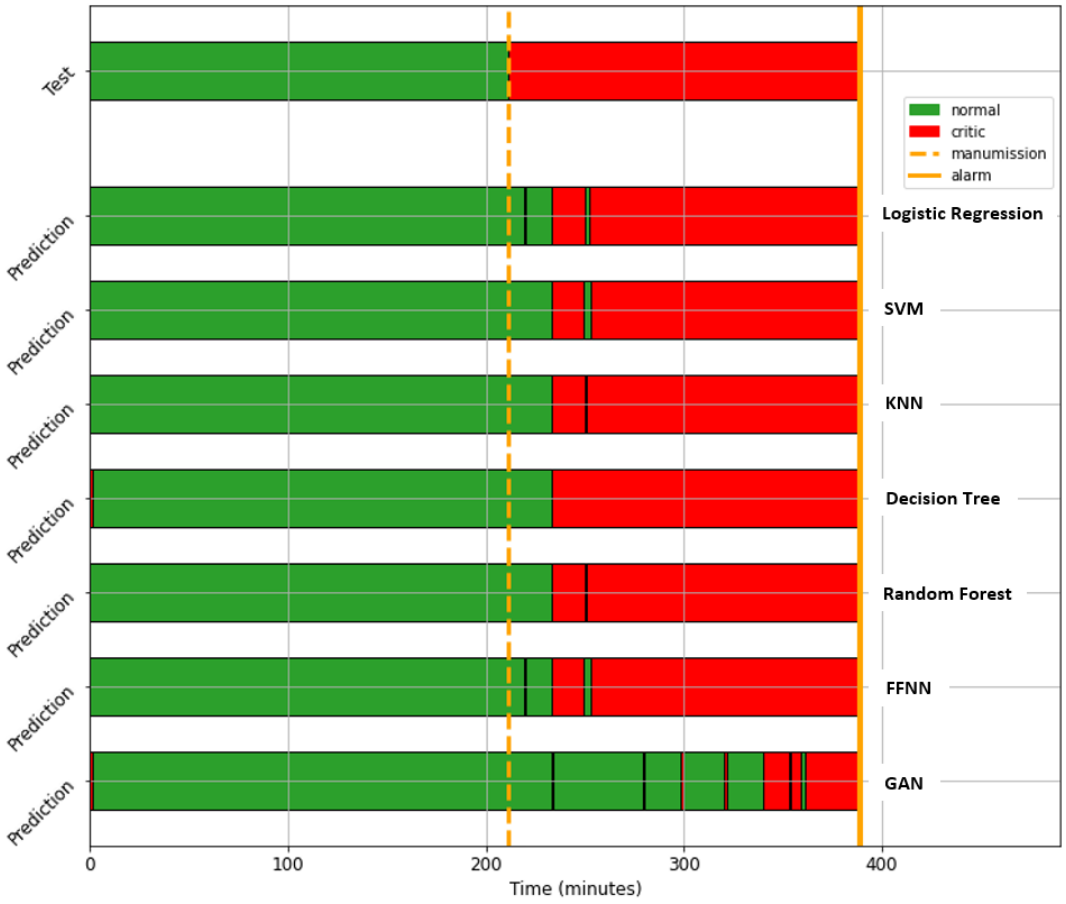


Figure 6.2: Performance of the models on the test set over time.

6.3 Generated samples

Another way to evaluate the ability of the GAN to adapt to the data is to evaluate the samples generated by the generator. Although producing realistic samples is not the main purpose of the project, observing the quality of the generated samples can be useful to understand the results obtained and the limitations of this model. The Figure 6.3 represent the PCA with the two main components and the 2D t-SNE of the training data and some samples generated through the generator.

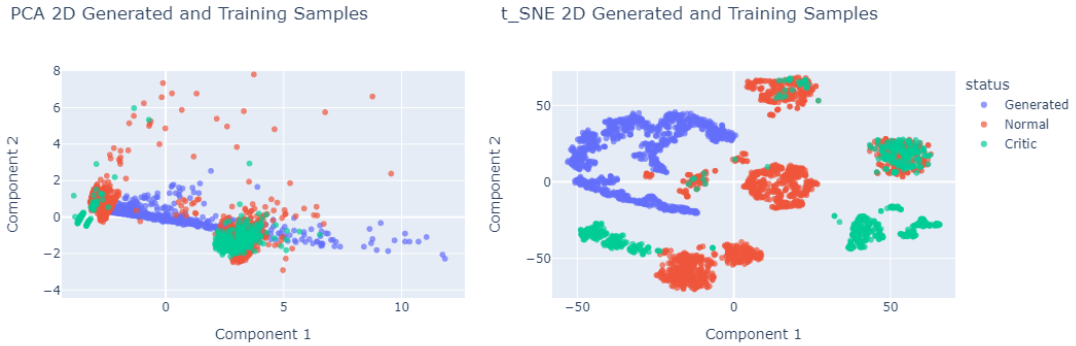


Figure 6.3: Two-dimensional PCA and t-SNE representations of the training set and some samples generated by the model.

Although there is no clear distinction between the two classes, in both cases the generated samples are quite close to normal ones. In the PCA it is possible to see how the samples tend to be very close to the largest cluster with normal samples. However, the quality of the generated samples is not that high.

6.4 Final comment

The implementation of less complex and supervised models is preferable both in terms of performance and time. However, as clustering and GAN have shown, unsupervised models can offer good performance just by monitoring the machine without carrying out tampering-induced experiments. This results into savings for the company and the possibility of collecting a substantial amount of data through simple monitoring.

Regarding the GAN, there are some interesting considerations. Usually, this model is not used for classification, but it can be adapted for that purpose. In addition, it can be trained with normal data without performing tampering-induced experiments. However, there are some disadvantages with this model:

- Selecting hyperparameters to find the balance between generator and discriminator may not be straightforward.
- It takes a long time to train compared to other models used in this project.
- Ideally, GANs are designed to be used with datasets with many variables and many samples (e.g., image datasets). In this case, additional data would be needed to justify the use of a GAN.

In conclusion, the idea of adapting a complex DL model such as a GAN can only be justified under suitable conditions. In this particular case it would be appropriate to invest in more conventional statistical models such as SVM or Random Forest. Finally, taking into account how data are collected for this kind of project, the use of unsupervised and semi-supervised models should be explored in order to limit the costs of experiments performed on the machine.

7 Conclusion

This thesis aims to provide an exploratory contribution to the use of a generative DL model in the context of predictive maintenance. In particular, the main task is to apply a Generative Adversarial Network to a classification task for the identification of critical states in which there is a risk of damage to the machine during the production process. So, the goal is to provide an alarm system capable of signaling the onset of a criticality with a certain advance compared to the system of the machine. This can also be seen as an anomaly detection task since the identification of these critical states occurs as a contrast to the samples belonging to the normal operating state of the machine.

The data provided by the company did not require particular treatment before being processed. Most of the choices made on the datasets concerned the ways in which they were combined and used for training and testing. At an exploratory level, two dimensionality reduction techniques were used in order to visualize the distribution of the samples. In this way it was possible to have a preliminary idea of whether or not to use DL models or simple classifiers. This analysis anticipated the relative simplicity of the task, as confirmed by the use of unsupervised models.

In view of these results, the GAN was compared with some classifiers that have shown significantly better performance. Therefore, it seems that the available data does not justify the use of complex models, confirming the fact that complex models do not necessarily perform better than simple ones.

However, decent results have also been obtained with GAN, demonstrating the possibility of adapting this type of model to a PdM task like this one. Furthermore, considerations must also be made on how the data are collected in the experiments conducted on the machine. Using a model that can only train with normal samples, as in the case of GAN, could provide a considerable advantage over supervised models. The need to induce critical issues has a cost both in terms of production and the risk of damage to the machine.

Regarding the limitations encountered in this project, they lie mainly in the datasets. Firstly, large datasets are needed to justify the use of a DL model, in fact even with simple classifiers it is possible to obtain very satisfactory results. Secondly, only one dataset is available on the valves wear out experiment. Another dataset from this

experiment would be needed to be able to test the models on both experiments conducted. The tests presented above were carried out exclusively with a dataset deriving from the vacuum pump wear out experiment. In the future, unsupervised and semi-supervised models should be explored in more detail. Also, some variants of the GAN should be explored, such as the Wasserstein-GAN [1] or Auxiliary Classifier GAN, in order to improve the performance of the generator, useful for creating new realistic normal samples. Regarding the critic samples, the implemented models distinguish only between normal and critic, it would be very useful to distinguish between types of criticalities (multiclass classification). A possible solution starting from the GANs could be to implement a GAN for the normal/critic distinction and a supervised model that classifies the type of critic identified by the GAN. It would also be interesting to compare the current time step classification approach with models for predicting subsequent steps, such as the Recurrent Neural Networks. Finally, the alarm reactivity system should be implemented. Given the consecutive increase in criticality alerts close to the machine stop, one possibility could be the triggering of the alarm after a certain number of consecutive alerts of the model.

In conclusion, further developments and experiments with sufficiently large datasets capable of justifying the use of a Generative Adversarial Network are needed, but it seems possible to adapt this approach to the type of task considered in this thesis.

Appendix

Other implementation details and hyperparameters

This section provides some technical details regarding the implementation and hyperparameters of each model selected during the evaluation phase through the CV.

The implemented code was entirely written in Python using the free online platform Google Colaboratory based on Jupyter; these are notebooks made up of cells containing text, images or code. One of the reasons behind this choice for the implementation of the project lies in the ease of use, in fact it is not necessary to configure the device to use the packages of Machine Learning and Deep Learning available on Python. Google Colaboratory also offers a free GPU service, which is very useful in the context of Deep Learning.

The main libraries used in the project were: `Numpy`, `Pandas`, `SciPy`, `Seaborn` and `Matplotlib`, instead for the implementation of the models, `Scikit-learn` for the classifiers and `PyThorc` for the Deep Learning models. Furthermore, some classes and auxiliary functions have been implemented to transform data, define, train and carry out the CV of Deep Learning models (FFNN and GAN) and create some graphs.

The Table A.1 and Table A.2 show the hyperparameters selected for the classifiers, the FFNN and the GAN through the CV.

Model	**Hyperparameter**	**Value**
Logistic Regression	inverse of regularization strength	4.237
	norm of the penalty	*L1*
	maximum number of iterations	377
Support Vector Machine	kernel type	*RBF*
	kernel coefficient γ	*Scale*
	regularization parameter	0.991
K-Nearest Neighbors	number of neighbors	18
	distance metric	*L1*
Decision Tree	criterion to measure the quality of a split	*Gini*
	splitter	*Best*
Random Forest	criterion to measure the quality of a split	*Entropy*
	number of trees	93

Table A.1: Hyperparameters of classifiers.

Model	Hyperparameter	Value
Feed Forward Neural Network	number of hidden layers	2
	number of hidden neurons	1024
	epochs	50
	type of activation function	*LeakyReLU (negative slope=0.2)*
	optimizer	*AdamW*
	learning rate	0.0017
	L2 regularization coefficient	0
	dropout probability	0.7
Generative Adversarial Network (Discriminator)	number of hidden layers	4
	number of hidden neurons	1024, 512, 256, 128
	epochs	500
	number of trainings of each model for each epoch	6
	type of activation function	*ReLU*
	optimizer	*Adam*
	learning rate	0.001
	L2 regularization coefficient	0008
	dropout probability	0.95
Generative Adversarial Network (Generator)	number of hidden layers	4
	number of hidden neurons	64, 128, 256, 512
	epochs	500
	number of trainings of each model for each epoch	1
	type of activation function	*LeakyReLU (negative slope=0.2)*
	optimizer	*AdamW*
	learning rate	0.0015
	regularization coefficient	0.002

Table A.2: Hyperparameters of FFNN and GAN.

Additional charts

Here are some charts, called dendrograms, obtained during the exploration of the dataset with Hierarchical Clustering. Each dataset, with the exception of dataset 4, has its own dendrogram (Figure A.1, Figure A.2 and Figure A.3) and relative clustering table (Table A.3, Table A.4, Table A.5; tables already shown in subsection 4.4.2.). In addition, the graph with the losses of the generator and discriminator (Figure A.4) and the confusion matrix of GAN (Figure A.5) and best models (Figure A.6) on the test set are provided.

Clusters/status	1	2	3	4	Total
0 (normal)	0	38	322	654	1014
1 (critic)	315	5	161	28	509

Table A.3: Hierarchical Clustering analysis dataset 1.

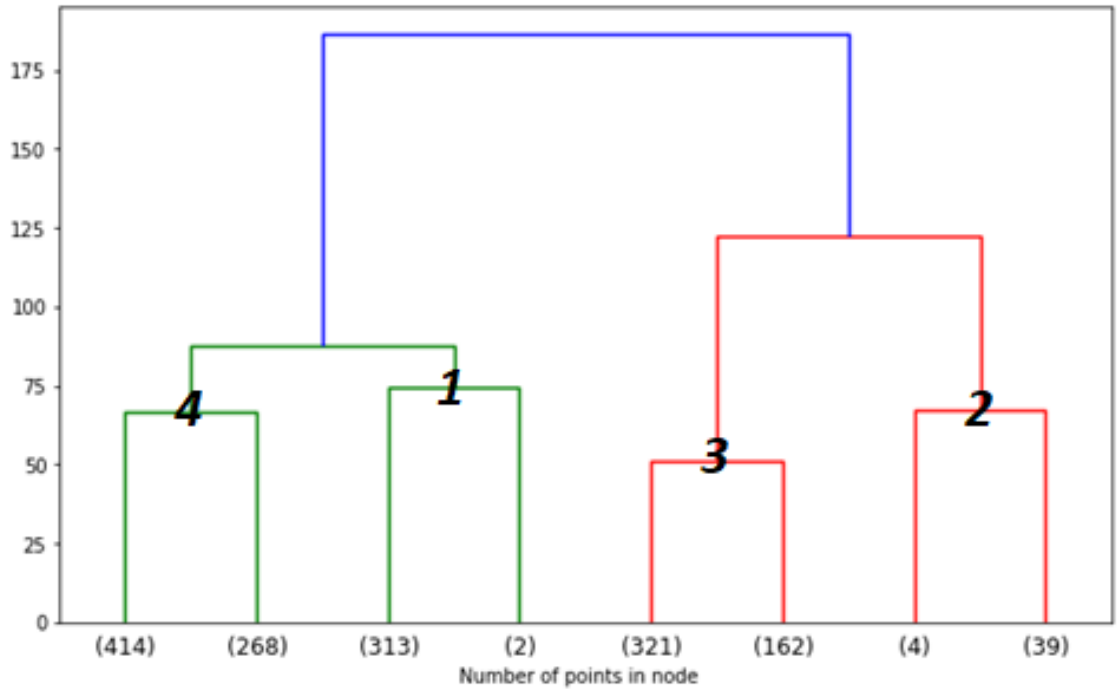


Figure A.1: Dendrogram of dataset 1. The numbers in bold represent the number of the cluster in which they have been inserted (see related table).

Clusters/status	1	2	3	4	5	Total
0 (normal)	5	4	147	18	1	175
1 (critic)	1	0	46	0	148	195

Table A.4: Hierarchical Clustering analysis dataset 2.

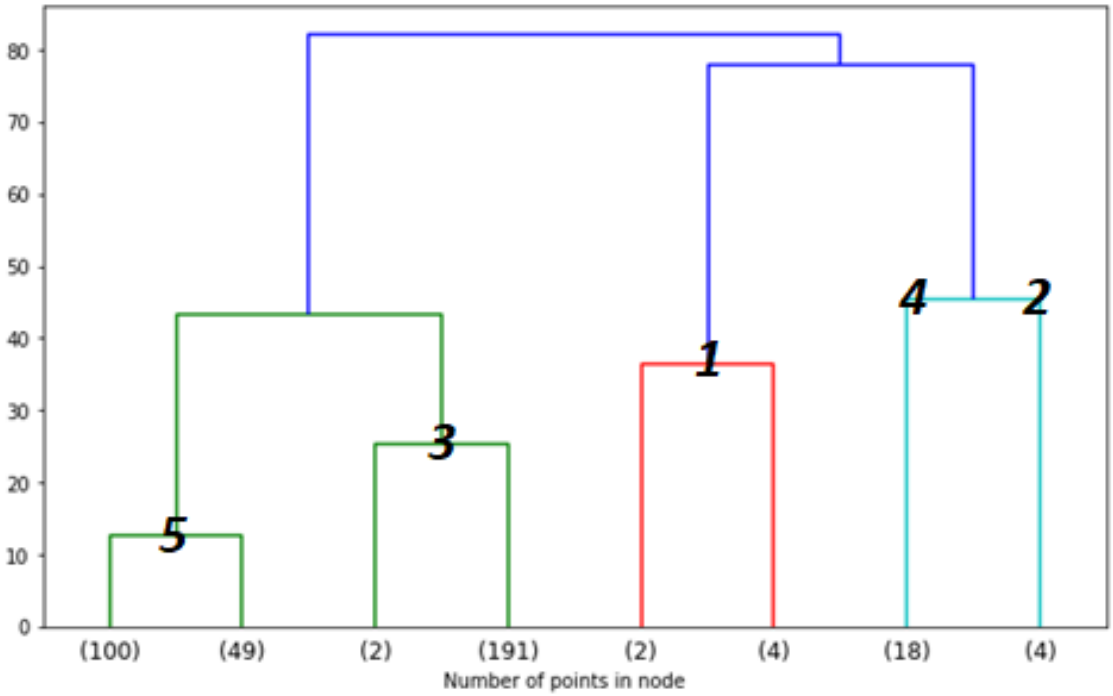


Figure A.2: Dendrogram of dataset 2. The numbers in bold represent the number of the cluster in which they have been inserted (see related table).

Clusters/status	1	2	3	Total
0 (normal)	208	3	0	211
1 (critic)	21	0	158	179

Table A.5: Hierarchical Clustering analysis dataset 3.

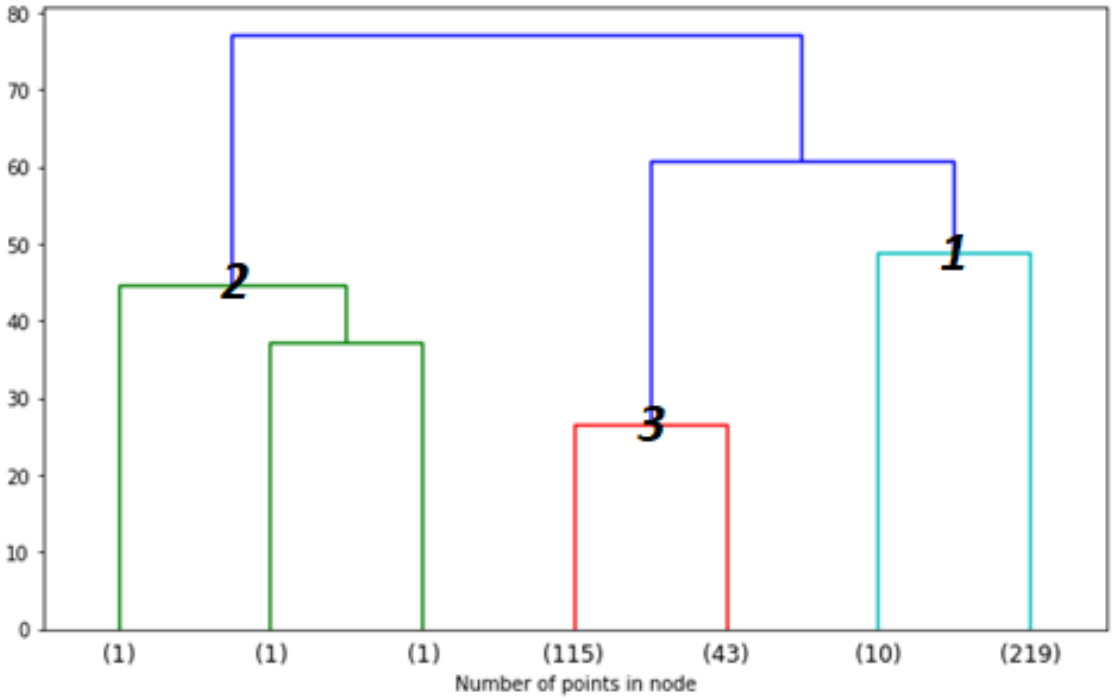


Figure A.3: Dendrogram of dataset 3. The numbers in bold represent the number of the cluster in which they have been inserted (see related table).

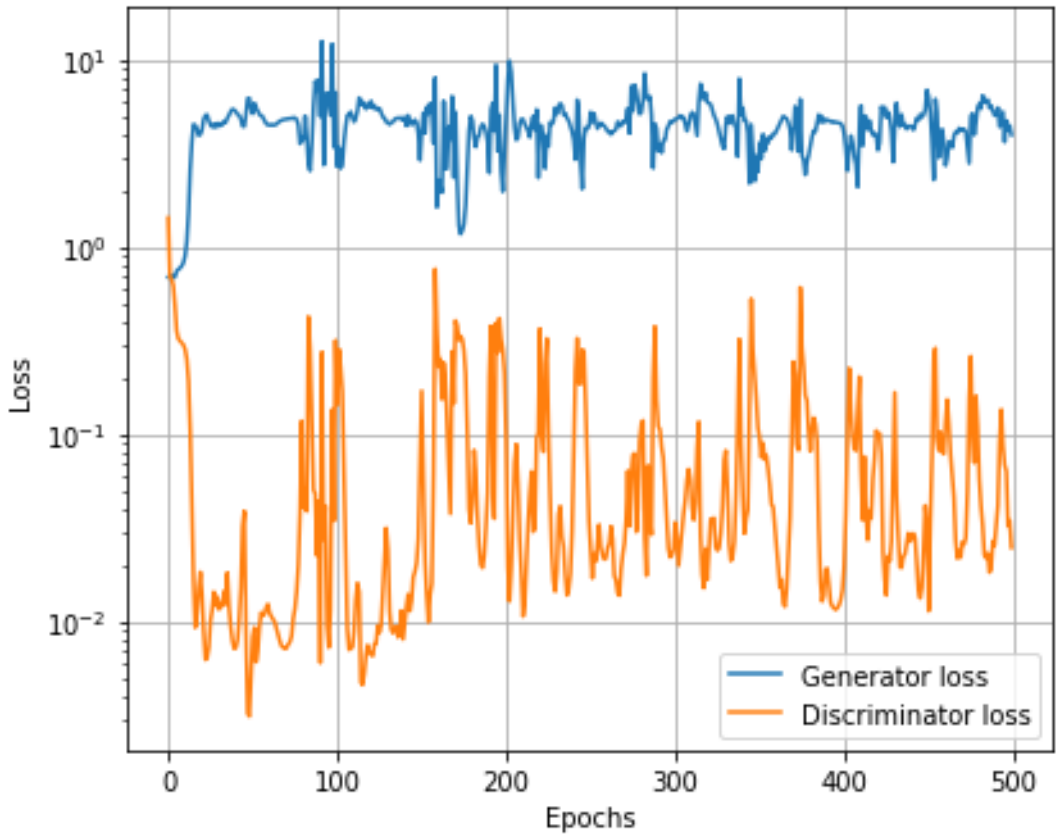


Figure A.4: Generator and discriminator losses in the final training.

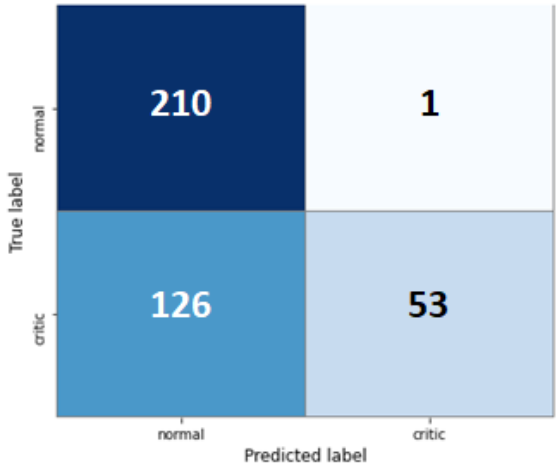


Figure A.5: GAN confusion matrix on the test set.

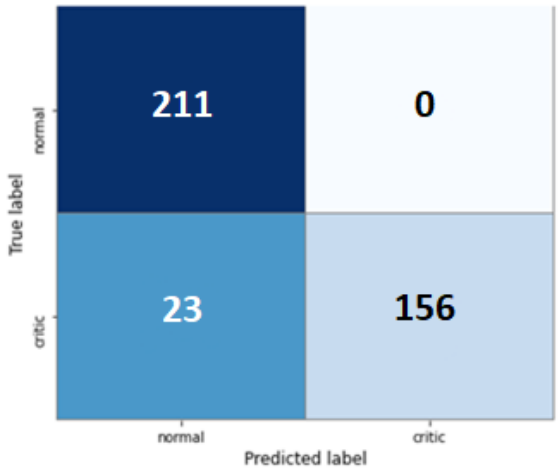


Figure A.6: Best models (Logistic Regression, KNN, Decision Tree and Random Forest) confusion matrix on the test set.

References

[1] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (pp. 214–223).: JMLR.org.

[2] Bashar, M. A. & Nayak, R. (2020). Tanogan: Time series anomaly detection with generative adversarial networks. *arXiv*, abs/2008.09567.

[3] GeeksforGeeks (2019). Ml | v-measure for evaluating clustering performance. Available at: `https://www.geeksforgeeks.org/ml-v-measure-for-evaluating-clustering-performance/`. Accessed: 2022-5-30.

[4] GeeksforGeeks (2020). Hierarchical clustering in data mining. Available at: `https://www.geeksforgeeks.org/hierarchical-clustering-in-data-mining/`. Accessed: 2022-5-30.

[5] Gharakhanian, A. (2017). Generative adversarial networks – hot topic in machine learning. Available at: `https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html/`. Accessed: 2022-6-2.

[6] Goodfellow, I. (2017). Nips 2016 tutorial: Generative adversarial networks.

[7] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. In *Advances in neural information processing systems* (pp. 2672–2680).

[8] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

[9] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24, 498–520.

[10] Li, D. & Gao, J. (2010). Study and application of reliability-centered maintenance considering radical maintenance. *Journal of Loss Prevention in the Process Industries*, 23(5), 622–629.

[11] Liu, C., Tang, D., Zhu, H., & Nie, Q. (2021). A novel predictive maintenance method based on deep adversarial learning in the intelligent manufacturing system. *IEEE Access*, 9, 49557–49575.

[12] Odena, A., Olah, C., & Shlens, J. (2016). Conditional image synthesis with auxiliary classifier gans. *arXiv*, abs/1610.09585.

[13] Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11), 559–572.

[14] Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In Y. Bengio & Y. LeCun (Eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[15] Rosenberg, A. & Hirschberg, J. (2007). V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* (pp. 410–420).: Association for Computational Linguistics.

[16] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., & Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *arXiv*, abs/1703.05921.

[17] Serradilla, O., Zugasti, E., & Zurutuza, U. (2020). Deep learning models for predictive maintenance: a survey, comparison, challenges and prospect. *arXiv*, abs/2010.03207.

[18] UNI-EN 13306:2018 (2018). *Maintenance. Maintenance terminology*. UNI Ente Italiano di Normazione, Milano, Italy.

[19] van der Maaten, L. & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605.

[20] VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media, 1 edition.

[21] Venkatasubramanian, V., Rengaswamy, R., Yin, K., & Kavuri, S. N. (2003). A review of process fault detection and diagnosis. *Computers and Chemical Engineering*, 27(3), 293–311.

Acknowledgments

First of all, I would like to thank my parents, Elia, Nicholas (“Lu citl”), grandparents, aunt and uncle for the trust and support shown in these years away from home.

I thank all the roommates, former roommates, former university colleagues and friends, with particular regard to Alessandro (”il Socio”), Piergiorgio, Raffaele and Marianna with whom I shared the victories and defeats that accompany university life and beyond.

I thank my university colleagues from the ”We has” group for all the help and availability provided. A special thanks goes to Alessandro, colleague and friend always ready for a valid help.

I would like to thank my supervisor Alberto Testolin and my tutor Marco Boresta for the support and availability provided during the internship and the thesis.

Finally, a final thanks goes to all those who have contributed, for better or for worse, to my growth.