Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Electrical, Electronics and Communication Engineering

(38^{th}cycle)

AI-Based Novelty Detection for Industrial and Robotic Applications

By

Umberto Albertin

******

Supervisor:

Prof. Marcello Chiaberge

Doctoral Examination Committee:

Prof. Giuseppe Tommaso Costanzo, HEIG-VD

Dr. Gabriele Gualco, ETEL SA

Politecnico di Torino

2026

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Umberto Albertin
2026

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to ...

Acknowledgements

This work has been developed with the contribution of the Politecnico di Torino Interdepartmental Centre for Service Robotics (PIC4SeR). This thesis is part of the project PNRR-NGEU which has received funding from the MUR – DM 351/2022.

Abstract

Current applications in both industry and academia often attempt to implement anomaly detection and predictive maintenance solutions without a thorough understanding of fault modes or their behavior in real systems. Similarly, in robotics, sensor data are frequently assumed to be perfectly reliable and are trusted uncritically when performing tasks such as navigation or localization. However, this assumption can lead to inaccurate predictions and unreliable system behavior.

This thesis addresses these limitations by introducing and analyzing several techniques based on the Novelty Detection paradigm. The work is structured into two main parts. The first part focuses on industrial applications, where different novelty detection algorithms are evaluated for bearing monitoring. In this context, an unsupervised machine learning benchmark is also conducted to assess how models commonly proposed in the literature perform in real-world industrial scenarios.

The second part of the thesis is dedicated to the application of novelty detection in the robotics domain, with a particular focus on localization. Specifically, this work investigates the reliability of Ultra-Wideband range measurements obtained from multiple antennas under varying environmental conditions. Novelty detection is employed to identify unreliable sensor measurements and improve the robustness of localization pipeline.

A key design principle underlying the proposed methods is simplicity and computational efficiency. The models presented in this thesis are intentionally lightweight and easy to interpret, making them accessible even to non-expert users. Moreover, their low computational requirements enable deployment on embedded systems, thereby supporting edge computing applications.

Overall, the contributions of this thesis demonstrate how novelty detection techniques can surpass existing state-of-the-art approaches in specific application domains. In addition, this work clarifies the fundamental conceptual differences

between anomaly detection and novelty detection, two terms that are often used interchangeably despite referring to distinct problem formulations.

Contents

Chapter 1

Introduction

Nowadays, companies and research institutions strive to develop methods for identifying the conditions that lead to failures in complex systems such as machines, robots, and vehicles. These methods fall under the increasingly widespread concept of predictive maintenance, which has gained attention in the industrial environment.

The concept of predictive maintenance is studied by the scientific community due to its usefulness in detecting and anticipating system failures. It is often confused with the notion of condition monitoring, although the two terms describe different approaches. Condition monitoring refers to the continuous observation of specific signals within a system. When one of these monitored signals—or a mathematical transformation thereof—exceeds a threshold defined by domain experts, the algorithm triggers an alert indicating that the threshold has been surpassed. The user should then inspect the system, identify the cause of the anomaly, resolve the issue, and restore normal operation. Predictive maintenance differs from condition monitoring in its ability to provide temporal forecasts of system degradation. While condition monitoring detects the onset of a problem only when a threshold is exceeded in real time, predictive maintenance estimates when a failure is likely to occur in the future. This forecast is commonly referred to as the Remaining Useful Life (RUL), which could represent the estimated number of days, hours, or operational cycles a system can continue functioning before a failure occurs. The concept of condition monitoring could be associated with reactive maintenance, a maintenance strategy in which components are replaced only after a failure has already taken place, as described in [1].

In recent years, a great deal of research has focused on improving RUL prediction through sophisticated algorithms, aiming to achieve incremental gains in prediction accuracy. However, the big problem faced by companies today is not the lack of algorithms, but rather the scarcity of high-quality data. This issue is particularly evident when a system or component is newly designed: insufficient operational data makes it difficult to train models able to reliably estimate the components' RUL. Under such circumstances, it is also difficult to set meaningful condition monitoring thresholds to distinguish between normal and faulty operating conditions. Although domain experts define preliminary thresholds (i.e., datasheet characteristics), these values are often either too high or too low to capture anomalies in the system. To address this lack-of-data problem, a new research direction has emerged in recent years: novelty detection.

Novelty detection enables the identification of novelties or deviations from a nominal condition without requiring large datasets. Such algorithms are particularly valuable when no historical data exist—for example, when a new machine is just designed and built, and no prior signals are available to define signal forecasting for a predictive maintenance algorithm. Novelty detection, therefore, represents a compromise between the benefits of a predictive maintenance framework and the practical limitations imposed by limited data availability.

The key distinction between predictive maintenance and novelty detection lies in the underlying assumption about the data. Predictive maintenance relies on extensive datasets to forecast the RUL, whereas novelty detection simply requires a well-defined nominal condition, such as the first operational hours of a new system. Although novelty detection does not predict RUL, instead it could provide an estimate of what might be termed the remaining time to alert [2], based on deviations from the nominal behavior. Another advantage of novelty detection is its high degree of flexibility. If the algorithm identifies a condition as novel but domain experts determine that it does not correspond to a fault, this new condition can be incorporated into the set of normal behaviors. Over time, this allows the system to accumulate a richer dataset that can be used to develop a future robust predictive maintenance model. In this sense, novelty detection can be regarded as a precursor to predictive maintenance: it supports the early understanding of a system's behavior, helps validate sensor configurations, and facilitates the forecasting of future RUL estimations.

As of 2025, the year in which this thesis is written, one of the primary challenges in novelty detection lies in identifying models that are sufficiently precise to determine when operating conditions deviate from their nominal state. The significant issue for this class of algorithms remains the false positives, thus the distinction between nominal and novel conditions. Recent developments in the field have increasingly adopted Artificial Intelligence (AI) techniques to classify system behavior as either normal or novel. Following current trends in the state of the art, researchers continue to design more sophisticated models to achieve higher accuracy while simultaneously minimizing false positive rates.

During these years, the main challenge concerned the development of models that are as simple and lightweight as possible, both in terms of usability and computational demand. The objective was to provide companies and non-expert users with a tool that allows them to experiment with and evaluate novelty detection techniques without the complexity typically associated with state-of-the-art algorithms. In other words, my intention was to offer accessible methods that can serve as practical alternatives to more sophisticated approaches.

The content of this thesis is organized into three main sections. Section 2 presents the theoretical background necessary to understand the methodologies analyzed in the works developed throughout these years. Sections 3 and 4 contain the most representative and significant contributions of my academic path during the PhD. The works in Section 3 focus on applications in industrial environments. Here, simple Machine Learning approaches are examined, discussing how novelty detection can be implemented using lightweight models. The works in Section 4 explore a more recent and emerging research direction within novelty detection, specifically related to the robotics domain. One of the key advantages of novelty detection is its flexibility across different application fields. This second research area, while not directly tied to industrial applications, investigates a more academic problem: the identification of problematic zones that affect robot localization in indoor environments. This work demonstrates how novelty detection methodologies can extend beyond traditional industrial maintenance tasks, offering valuable tools for analyzing and interpreting environmental conditions in robotics. The motivation for exploring this new field of application lies in my interest in extending previous studies related to novelty detection. Robotics presents numerous open challenges, with localization being one of the most prominent and difficult to address due to the variety of environmental and operational factors involved. Particularly compelling is the adaptability of the

novelty detection paradigm to an entirely different application domain. This shift has allowed me to work across both industrial and academic contexts, thereby gaining exposure to a diverse range of methodologies, problems, and research perspectives.

The remainder of this thesis is structured as follows: Section 5 presents the conclusions and offers a critical reflection on the work carried out during these years. Finally, Section 6 outlines future research directions that, hopefully, will contribute to further advancements in the field of novelty detection.

Chapter 2

Theoretical Analysis

This chapter presents the fundamental concepts necessary to understand the works presented in Sections 3 and 4. In more detail, this section is divided into a preliminary presentation of how a Novelty Detection technique works, followed by a discussion of the techniques used to develop it, with an in-depth analysis of the Artificial Intelligence approaches.

2.1 Novelty Detection

Novelty Detection is a technique that has been developing in recent years and aims to identify differences between a nominal and a novel condition. Often, this type of architecture is modeled as a binary classifier (i.e., an architecture capable of discerning only two classes: the normal class and the novel class).

Formally, in a binary classification problem, given a dataset $X = \{(x_i, \omega_i) \mid x_i \in \mathbb{R}^D, i = 1, \ldots, N\}$, where each sample x_i has dimension D and is associated with a label $\omega_i \in \{0, 1\}$, the goal is to identify the correct class for a given input vector [3].

Let h denote the binary decision function of a generic architecture:

$$\omega = h(x|X) : \mathbb{R}^D \to [0, 1] \tag{2.1}$$

Often this kind of architecture uses a score to estimate if the input vector belongs to a class or to the other. Most of the time, this score is a number that is the

representation of a difference between the nominal input and the actual one. Indeed, it is possible to define a threshold k that is the number used for discerning the two classes. Calling the scoring function g, the two possible situations can happen:

$$g(x) \leq k \rightarrow 0 \tag{2.2}$$

$$g(x) > k \rightarrow 1 \tag{2.3}$$

The challenge consists of defining a model or architecture able to estimate the score, in order to obtain a reliable quantity for comparison. Clearly, an appropriate choice of the threshold k is also crucial for reducing the false positive rate. [3] classifies Novelty Detection algorithms into five main categories:

- Probabilistic approach estimates the density of the normal class, identifying novel samples in low-density regions.
- Distance-based methods rely on clustering models and assume that normal data form compact clusters, whereas novel samples lie far from them.
- Reconstruction-based methods employ machine learning or deep learning models trained exclusively on normal data, identifying novel samples through a high reconstruction error.
- Domain-based methods attempt to describe the normal data distribution while excluding high-density regions.
- Information-theoretic methods use measures such as entropy to quantify the normality of the data.

In this work, the reconstruction-based approach will be analyzed, using both machine learning and deep learning models, which will be discussed in the next section.

2.2 Artificial Intelligence

The term Artificial Intelligence (AI) is deeply integrated into modern language. In many cases, it is used improperly to refer to any application that employs complex

neural network models, such as ChatGPT or Google Gemini. However, behind these simple terms lie a wide range of complex concepts that have been developed over the last century. Artificial Intelligence is the most general term used to describe systems capable of making decisions autonomously. Within this broad domain, two important subfields are commonly identified: machine learning and deep learning. Furthermore, deep learning can be considered a subset of machine learning. Consequently, Artificial Intelligence represents the most general domain, including machine learning and deep learning. Figure 2.1 illustrates this hierarchical relationship through a Venn diagram.

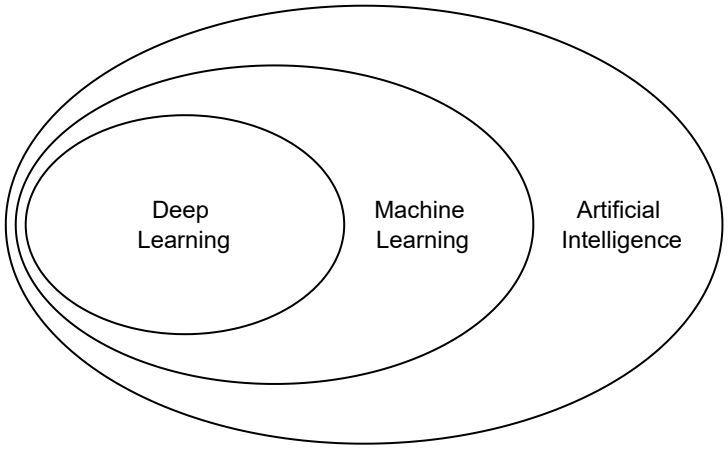


Fig. 2.1 Venn diagram of the artificial intelligence domain. Artificial Intelligence (AI) is a broad term that encompasses both Machine Learning and Deep Learning as subfields within it.

As reported in [4, 5], Artificial Intelligence (AI) is a general term used to describe the ability of a system to perform tasks that typically require the human brain, such as speech recognition and decision-making. Machine Learning (ML) is a subfield of AI that enables systems to learn how to perform tasks from historical data. Deep Learning (DL) extends the machine learning paradigm by employing large-scale datasets and more complex architectures. By leveraging vast amounts of data, deep learning methods are able to address highly complex tasks, such as image and speech recognition, object detection, and many others. There is a long and rich history behind the concepts of AI and DL. Historically, three major waves of deep learning have been identified, as reported in [6]: cybernetics in the 1940s–1960s, connectionism during the 1980s–1990s, and the current wave, commonly referred to

as deep learning, which began around 2006. One of the earliest models associated with the first wave of cybernetics was proposed by McCulloch and Pitts [7]. This model represented one of the first attempts to emulate the behavior of the human brain. It could recognize only two categories, acting as a binary classifier that produced either a positive or a negative output. The decision was made by manually setting a suitable set of weights in a linear function that yielded one of the two possible outputs. Subsequently, Rosenblatt [8, 9] introduced the perceptron, the first model capable of automatically learning weights from input data. Around the same period, Widrow and Hoff developed the ADALINE (Adaptive Linear Element) model [10], which could predict continuous-valued outputs rather than binary categories. Today, these approaches are collectively referred to as linear models. Notably, ADALINE was the first model to employ the Stochastic Gradient Descent (SGD) algorithm to automatically learn model parameters for estimating a real-valued output. Modern deep learning algorithms still rely on variants of SGD as the predominant training strategy. One of the main limitations of linear models was later highlighted by Minsky and Papert [11], who demonstrated that such models are unable to learn the XOR function. This discovery marked the first major decline in research interest in neural networks, often referred to as the first "AI winter" in the deep learning field.

Neuroscience has been a significant source of inspiration for deep learning, but it has not served as its primary guiding framework due to the extreme complexity of the human brain. A vast number of neurons are involved in decision-making processes, and, to date, understanding how the brain operates in detail remains highly challenging. As a result, it is currently infeasible to construct a comprehensive mathematical model capable of fully emulating brain functionality [12].

As shown in [13], most mammalian brains appear to rely on a single underlying algorithm to perform a wide variety of tasks. This insight opened the possibility of unifying previously fragmented research efforts, which had traditionally focused on solving isolated tasks. Although research areas remain distinct today, many research groups now work across multiple domains, combining expertise to emulate the integrated processing capabilities of the brain, as suggested by the studies of von Melchner.

The history of deep learning, starting from its roots in neuroscience, can be summarized through the following key developments. In the 1980s, Fukushima introduced a novel architecture called the Neocognitron for image processing, which

laid the foundation for modern convolutional neural networks. The original Cognitron, also developed by Fukushima [14], was highly complex and aimed at closely emulating biological brain functions.

Subsequently, a simplified model of the neuron, known as the Rectified Linear Unit (ReLU), was introduced by combining ideas proposed by several researchers [15–18]. This activation function, which is analyzed in detail in Section 2.8, significantly contributed to the practical success of deep learning architectures.

In the 1980s, the connectionism movement emerged within the broader context of cognitive science. Cognitive science initially gained popularity through the study of symbolic reasoning. However, translating symbolic reasoning into symbolic models proved difficult to implement using neural networks. Touretzky and Hinton [19] were among the first connectionist researchers to study the work of psychologist Donald Hebb [20]. During this period, the concept of distributed representation was introduced [21]. The core idea behind this concept is that each input should be represented by a set of features distributed across multiple units rather than by a single symbolic entity.

Other significant discoveries of this era include the backpropagation algorithm, introduced by Rumelhart [22] and further developed by LeCun [23], which remains one of the primary methods for training neural networks. Additionally, Hochreiter and Schmidhuber introduced the Long Short-Term Memory (LSTM) architecture [24], which is widely used today for Natural Language Processing (NLP) tasks.

The current wave of deep learning began when Hinton demonstrated that deep neural networks could be effectively trained using a strategy known as greedy layer-wise pretraining [25]. This breakthrough solidified the term deep learning as it is understood today, highlighting the ability to train networks with significantly more layers than in previous waves. Initially, this wave focused on improving generalization performance when only limited amounts of training data were available. In contrast, contemporary research increasingly revisits very deep architectures to fully exploit the availability of large-scale datasets.

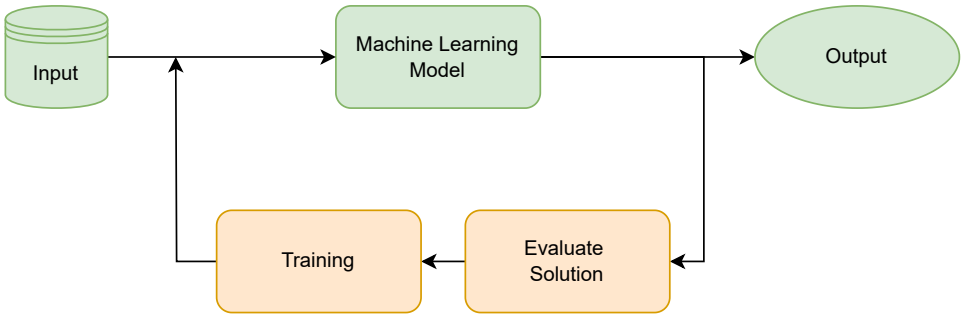


Fig. 2.2 Machine Learning workflow.

2.3 Machine Learning

Machine Learning (ML) is the science of programming computers so that they can learn from data [26]. Machine learning is a paradigm that can be applied to both simple and complex tasks. For example, consider the problem of recognizing words in a natural language. Each language contains a vast vocabulary, with numerous accents, pronunciations, and linguistic variations. In such cases, it is extremely difficult to design a deterministic mathematical algorithm capable of correctly recognizing every word.

Machine learning provides an effective alternative, thanks to its ability to learn patterns and features directly from input data. As stated, a machine learning model is able to learn from historical data. While it cannot directly produce outputs for situations it has never encountered, various mathematical techniques enable forecasting and generalization by exploiting the latent features extracted during training. Figure 2.2 illustrates the general workflow of the machine learning paradigm.

Today, the scientific community identifies four different types of Machine Learning: supervised, semi supervised, unsupervised learning, and reinforcement learning algorithms. There are many differences between these types of machine learning that depends on the label of the task. The label or class, corresponds to the output of the machine learning model. The label can assume different shapes, it could be a regression or a integer number, identifying the category. In the following paragraphs it will be analyzed the first three machine learning paradigms, leaving out the Reinforcement Learning because a bit far from the scope of the works presented in the next part.

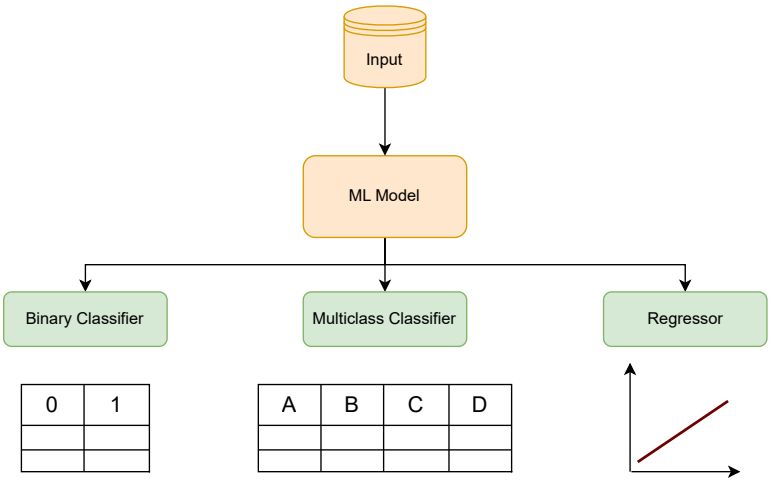


Fig. 2.3 Three types of machine learning tasks: binary classifier (producing two possible labels), multiclass classifier (producing one label among multiple classes), and regressor (producing a continuous output value following the learned data pattern)

2.3.1 Supervised Learning

When labeled data are available, the learning paradigm is referred to as supervised learning. In this type of machine learning, the training dataset includes labels provided by the user. Supervised learning is commonly used to perform three main types of tasks (Figure 2.3): binary classification, multiclass classification, and regression. A binary classifier is trained using only two labels and, based on the input data, assigns one of the two classes as output. Multiclass classification extends this concept by allowing the model to predict one among multiple classes observed during training; such models are often more complex than binary classifiers. Finally, regression models differ from classification models in that they output a continuous value rather than a discrete class label. By learning the relationship between input features and continuous targets, regression models can perform forecasting, even for input–output combinations not explicitly observed during training.

2.3.2 Unsupervised Learning

The unsupervised learning paradigm differs fundamentally from supervised learning. In this case, no labels are predefined by the user; instead, the model attempts to learn autonomously from the data. Unsupervised learning is often associated with

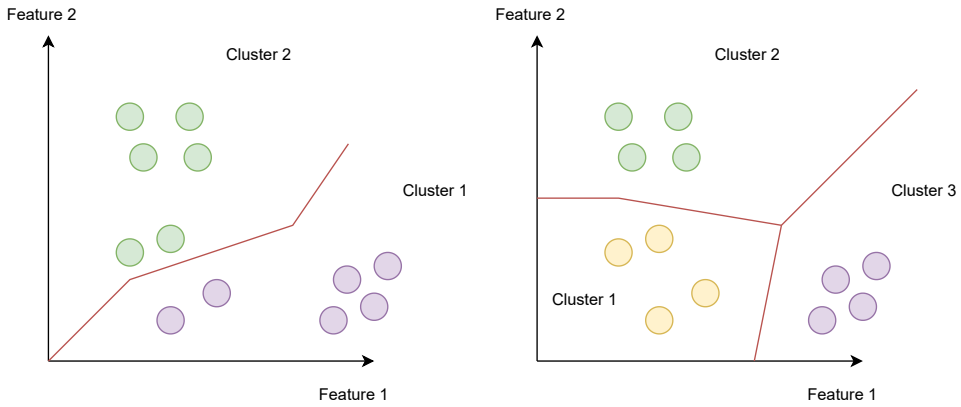


Fig. 2.4 Example of unsupervised machine learning based on clustering. Two possible cluster configurations.

the concept of clustering, as many unsupervised models aim to organize input data by exploiting the underlying patterns inherent in the data itself. In this context, the model partitions the data into multiple groups across potentially high-dimensional feature spaces. Data points that are similar according to multidimensional similarity measures tend to aggregate into groups known as clusters. Clustering algorithms are among the most common techniques in unsupervised machine learning. However, deep learning has also introduced unsupervised approaches that do not explicitly rely on clustering, but instead learn more complex latent representations, as discussed in Section 2.8. Unsupervised learning methods are often used as a precursor to supervised learning. Once meaningful clusters are identified within a dataset, labels can be assigned to the data, enabling the subsequent application of supervised algorithms. An example of an unsupervised learning algorithm is illustrated in Figure 2.4.

As shown in figure 2.4, the user can select one of the n possible configurations. In general, there is no predefined number of clusters that must be chosen. In principle, the number of clusters could be set equal to the number of input samples; however, such a choice is meaningless from the application point of view. Instead, the user should select a number of clusters that is appropriate and meaningful for the specific application.

2.3.3 Semi-Supervised Learning

The final paradigm is semi-supervised learning, which represents a trade-off between supervised and unsupervised learning. Two main interpretations of this paradigm exist: the first focuses on reducing the cost and time associated with the labeling process by leveraging a large amount of unlabeled data together with a small subset of labeled samples. The second interpretation is more closely related to the nature of the available data. In many applications, and particularly in Novelty Detection, explicit labels are not available to the learning algorithm, even though the user possesses implicit knowledge about the data. As described in the previous Section 2.1, novelty detection models are typically trained using nominal data only. While the model itself is agnostic that the training data represent the nominal condition, this information is known to the user. For this reason, such approaches can also be regarded as a form of semi-supervised learning.

2.3.4 Curse of Dimensionality

One of the main challenges in machine learning is the curse of dimensionality. This term was first introduced by Richard Bellman in his seminal works [27, 28]. The concept refers to a common problem in machine learning that arises from the dimensionality of the input data used for training. As the dimensionality of the input space increases, the number of samples required for the learning grows rapidly. In particular, the relationship between the number of features and the amount of training data increases exponentially. Several approaches have been proposed to mitigate this issue. Among them, the two most widely used strategies are dimensionality reduction and feature selection. Dimensionality reduction consists of techniques that reduce the dimensionality of the data by compressing the original high-dimensional information into a lower-dimensional representation, where each new dimension is a combination of the original features. One of the most well-known methods in this category is Principal Component Analysis (PCA). Feature selection, on the other hand, also aims to reduce the dimensionality of the input data, but does so by selecting the most informative features and discarding the less relevant ones.

2.3.5 Main Challenges

When dealing with machine learning, two main aspects must be carefully considered: the input data and the model design. The challenges associated with data handling are related to the quality and information content of the data itself. If the data used for training are not representative or are insufficient, the training process will be ineffective. It is crucial that the amount of data is tailored to the dimensionality of the feature space, in order to mitigate the effects of the curse of dimensionality (i.e. applying data reduction techniques). Moreover, the training data must be representative of the underlying problem domain for generalization purposes. If this condition is not satisfied, the model may fail when tested on previously unseen samples (i.e. generalization problems).

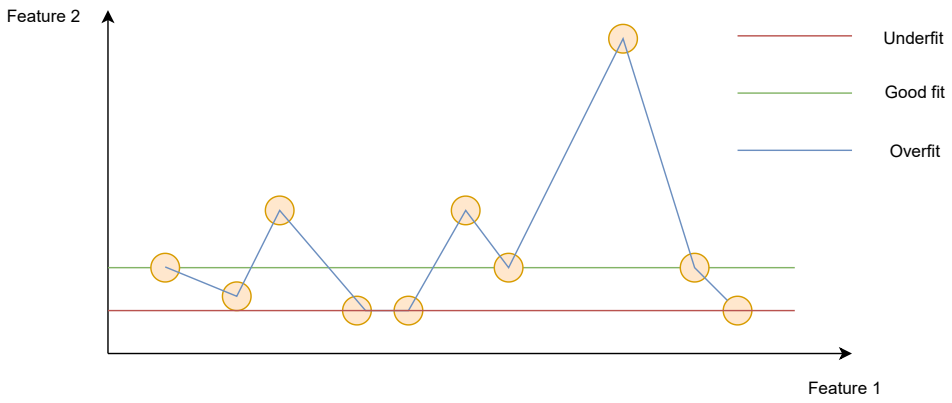


Fig. 2.5 Difference between underfitting and overfitting based on model complexity. The green line represents a good fit, the red line indicates underfitting, and the blue curve represents overfitting.

With regard to model design, two undesirable conditions may arise in machine learning: overfitting and underfitting. These two phenomena represent opposite extremes. Overfitting occurs when a model is excessively complex relative to the amount or quality of the training data. In this case, the model learns the training dataset almost perfectly, producing highly accurate predictions on the training samples, but fails to generalize to unseen data, often yielding incorrect outputs. Underfitting, on the other hand, represents the opposite situation, in which the model is too simple to capture the underlying patterns and hidden features of the input data. As a result, the model produces overly generic predictions, often returning similar outputs even when the inputs are substantially different. A graphical illustration of

these two problems is shown in Figure 2.5. As observed, the overfitted model closely follows the training data trend but lacks generalization capability, while the red curve corresponds to an underfitted model that fails to learn meaningful features from the data. Possible solutions to mitigate overfitting include increasing the amount of training data, thereby better capturing the diversity of the dataset, or applying techniques such as cross-validation and bootstrapping. Cross-validation consists of splitting the dataset into several subsets and repeatedly training and validating the model on different partitions, whereas bootstrapping is a resampling method that generates multiple datasets by sampling from the original data with replacement. A desirable trade-off is represented by the green curve, which illustrates a model achieving a good balance between these two characteristics.

2.4 Machine Learning models

In this section, a recap of the main machine learning models is presented, focusing on those adopted and described in the following chapter dedicated to the PhD works. Each model is introduced from a theoretical perspective, with the aim of explaining its underlying principles and functionality, progressing from simpler models to more complex ones.

2.4.1 Training, Testing and Validation dataset

The first step in setting up a machine learning problem is to divide the dataset into three distinct parts: the training, testing, and validation sets. The training set is used to train the algorithm and adjust its parameters. The testing set is dedicated for evaluating the performance of the trained model on unseen data. Finally, the validation set is sometimes used during training to monitor the model's performance and detect potential overfitting or underfitting over time.

2.4.2 Optimization Cost Functions

One of the main elements of Machine Learning is the ability of a system to automatically learn the parameters of a model for prediction. In technical terms, the quantities that are updated at each training step to improve the fit to the training data

of the model are called hyperparameters. To achieve this, every machine learning algorithm relies on several metrics that are necessary to monitor the training process and to assess the current state of the model. Focusing on supervised approaches, the most commonly used metrics are based on the error between the model output and the ground-truth labels of the dataset. In the literature, many different types of error measures are proposed. The simplest one is the standard error, computed as the difference between the predicted output and the true label of the data:

$$e(X, f) = \sum_{i=1}^{m} \hat{y} - y \tag{2.4}$$

where X is the input vector of the training dataset, f is the prediction function of the model, $\hat{y}$ is the predicted label of the model while y is the real label of the dataset, and e is the error between the two. This is a very basic metric; however, more complex and informative metrics have been proposed in the literature, such as the following:

$$mse(X, f) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2.5}$$

this metric is known as the Mean Squared Error (MSE), where m is the number of instances over which the error is computed, and the remaining notation is the same as in equation 2.4.

$$rmse(X, f) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2} \tag{2.6}$$

that is the Root Mean Squared Error (RMSE), where the notation is the same for the equation 2.5.

$$mae(X, f) = \frac{1}{m} \sum_{i=1}^{m} |f(x_i) - y_i| = \frac{1}{m} \sum_{i=1}^{m} |\hat{y}_i - y_i| \tag{2.7}$$

This metric is known as the Mean Absolute Error (MAE). During training, these metrics define the cost function to be minimized in order to reduce the difference between the true labels and the predicted ones. A fundamental question then arises: how can we find the best solution for choosing the model hyperparameters in a

specific application? Most of the time, finding the optimal solution is not realistically feasible; however, it is possible to obtain a sufficiently good solution by means of the Gradient Descent optimization algorithm.

Basically, the idea of an optimization algorithm is to iterate different combination of hyperparameter in order to reach the best cost function. For example, in a supervised learning, one of the cost functions could be one of the errors presented before, thus the optimization algorithm tries to reduce the error as much as possible. The idea behind the Gradient Descent is to find a direction of the change of the hyper-parameters in order to find the minima of the cost function selected. Often, the initialization of the hyper-parameters is random at the beginning, then every step, there is an update that gives the direction of the changes finding the minima optimizing the cost function. The main problem is that, a complex model does not have only a global minima, but they can have also a lot of local minima that mislead the best hyper-parameters. To enhance this behaviour is possible to set a learning rate that is a parameter able to configure the amount of the step of the change. A larger learning rate can converge faster but can also do not converge at all. On the opposite hand a lower learning rate can converge for sure to a local minima but if the step is not enough it will stay in the local minima without leaving it and misleading it into a global one. In figure 2.6 is possible to see a representation of the Gradient Descent algorithm.

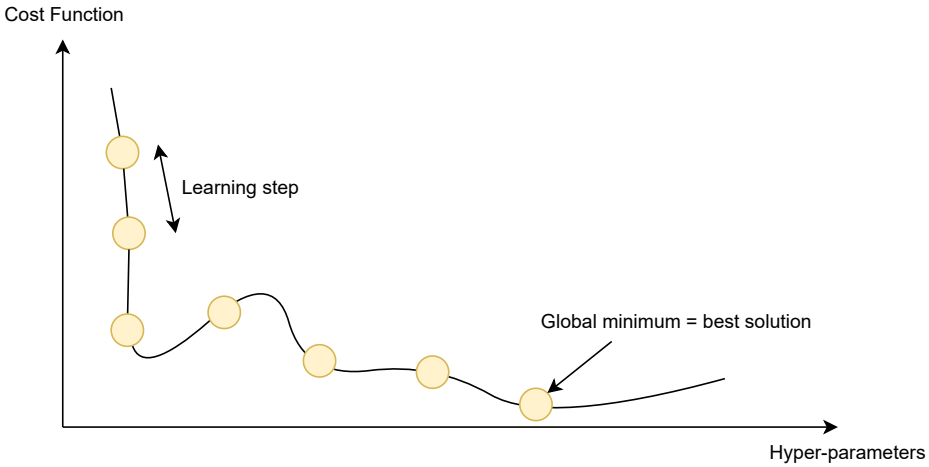


Fig. 2.6 Gradient Descent algorithm representation.

In general, the goal of an optimization algorithm is to iteratively explore different combinations of model hyperparameters in order to minimize a given cost function. For instance, in supervised learning, a cost function may correspond to one of the error metrics introduced previously; therefore, the optimization algorithm aims to reduce the error as much as possible.

The idea behind the Gradient Descent algorithm is to identify the direction in which the model parameters should be updated in order to reach a minimum of the selected cost function. Typically, the parameters are randomly initialized at the beginning of the training process, and at each iteration they are updated according to the gradient of the cost function, which provides the direction of steepest descent.

A major challenge is that complex models usually exhibit not only a single global minimum, but also multiple local minima, which may prevent the algorithm from converging to the optimal solution. To address this issue, a learning rate is introduced, which controls the magnitude of each parameter update step. A large learning rate may lead to faster convergence, but it can also cause instability or divergence. Conversely, a small learning rate generally ensures convergence; however, if the step size is too small, the algorithm may become trapped in a local minimum and fail to reach the global one.

In the literature, three main variants of the Gradient Descent algorithm are commonly distinguished:

- *Batch Gradient Descent*: this variant computes the gradient using the entire training dataset at each iteration. The main drawback of this approach is its high computational cost, since processing the full dataset at every update step can be extremely time-consuming, especially for large-scale datasets.
- *Stochastic Gradient Descent (SGD)*: unlike Batch Gradient Descent, this approach updates the parameters by computing the gradient using only a single randomly selected training instance at each iteration. This randomness introduces noise into the optimization process, which can help the algorithm escape local minima, but may also lead to instability. A common technique used to mitigate this issue is simulated annealing, which gradually adapts the learning rate during training. The main advantage of SGD is its low computational cost, making it suitable for large datasets and scenarios with limited hardware resources.

- *Mini-batch Gradient Descent*: this method represents a compromise between Batch Gradient Descent and Stochastic Gradient Descent. At each iteration, a small randomly selected subset (mini-batch) of the training data is used to compute the gradient. This approach reduces the noise compared to SGD while remaining computationally efficient, since only a portion of the dataset is processed at each step, rather than the entire dataset as in Batch Gradient Descent.

On the other hand, unsupervised machine learning algorithms do not operate in the same manner, as no ground-truth labels are available; consequently, an error between predicted and true labels cannot be computed.

In such cases, the scientific community has developed alternative evaluation metrics that are more closely related to clustering algorithms. It is well known that most unsupervised learning methods rely on clusters that are formed directly from the latent features of the data. In this context, clustering evaluation metrics are commonly classified as follows:

- *Cohesion*: also referred to as intra-cluster distance, it measures the similarity among data points within the same cluster, indicating whether the cluster is dense or sparse.
- *Separation*: also known as inter-cluster distance, it measures how well different clusters are separated from each other.

A graphical representation of these two metrics is shown in Figure 2.7.

In a formal way it is possible to show that the cohesion is:

$$Cohesion(x_i) = CHS(x_i) = \frac{1}{|n_i| - 1} \sum_{x_i \neq y_i} d(x_i, y_i) \tag{2.8}$$

while the separation between two clusters is:

$$Separation(x_i, x_j) = SEP(x_i, x_j) = min_{j \neq i} \frac{1}{|n_j|} \sum d(x_i, x_j) \tag{2.9}$$

where i and j are two clusters with n_i and n_j points respectively, x_i is a point of the cluster i and x_j is a point of the cluster j, and $d(x_i.y_i)$ is the distance between

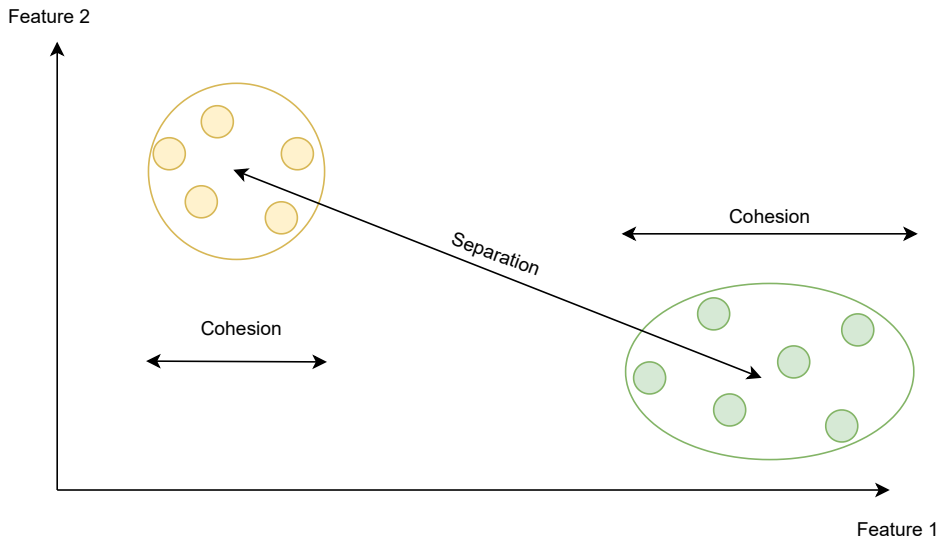


Fig. 2.7 Cohesion and Separation metrics in a clustering algorithm

two point inside the same cluster i, while $d(x_i, x_j)$ is the distance between a point of the cluster i and a point of the cluster j. The distance can assume the shape of an Euclidean distance or a Manhattan distance. In literature then, there is a metric that tries to condense the cohesion and the separation metrics, called Silhouette score [29], with the following shape:

$$\begin{aligned} Silhouette(x_i) = \frac{SEP(x_i) - CHS(x_i)}{max(SEP(x_i), CHS(x_i))} \quad if \quad C_i > 1 \quad \rightarrow \\ \rightarrow \begin{cases} 1 - \frac{CHS(x_i)}{SEP(x_i)} \quad if \quad CHS(x_i) < SEP(x_i) \\ 0 \quad if \quad CHS(x_i) = SEP(x_i) \\ \frac{SEP(x_i)}{CHS(x_i)} - 1 \quad if \quad CHS(x_i) > SEP(x_i) \end{cases} \end{aligned} \tag{2.10}$$

where $CHS(x_i)$ is the cohesion while $SEP(x_i)$ is the separation. The Silhouette coefficient takes values in the range $[-1, +1]$. A value close to $+1$ indicates that the data point is well matched to its assigned cluster and far from neighboring clusters. A value close to 0 suggests that the data point lies near the boundary between two clusters, while a value close to -1 indicates that the data point may have been assigned to an incorrect cluster.

2.5 Supervised Machine Learning models

In this section, the most commonly used algorithms in current research will be analyzed, providing the theoretical foundations necessary to better understand chapters 4, 5 dedicated to the proposed works.

2.5.1 Linear Regressor

Although it may seem counterintuitive, a simple linear function represents one of the most basic machine learning models, as it is capable of performing predictions based on the input data used to estimate its parameters. The mathematical formulation of a simple linear regressor is given by:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta \bullet x \tag{2.11}$$

where y is the predicted value, n is the number of features, x_i is the i^{th} feature value and θ_j is the j^{th} model parameter. Thus, the key idea is to find the right parameters' values in order to obtain a multidimensional linear function able to give an output that is satisfying with respect to the ones used for the training.

2.5.2 Polynomial Regression

There exists a related model that differs from the previous one in that it is capable of representing not only linear relationships, but also nonlinear ones, such as quadratic, cubic, and higher-order polynomial functions.

$$y = \theta_0 + \theta_1 x^2 + \theta_2 x^3 + ... + \theta_n x^n \tag{2.12}$$

In practice, data may exhibit patterns in the feature space that are not strictly linear; therefore, more complex models are often required to capture such relationships. As shown in Figure 2.8, there are situations in which a linear regressor is not sufficient to properly model the data distribution across all feature dimensions.

In this example, the data distribution clearly follows a quadratic trend when represented in a two-dimensional feature space. Consequently, a linear regressor

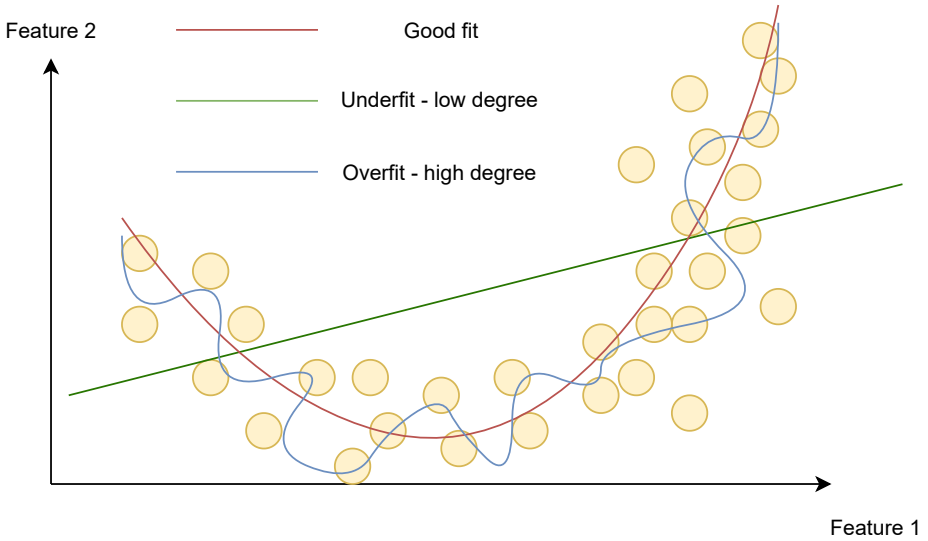


Fig. 2.8 Difference between low-, high-, and optimal-degree polynomial regression

fails to accurately estimate the target values, leading to the phenomenon known as underfitting.

Conversely, overfitting occurs when a high-degree polynomial model fits the training data perfectly but fails to generalize well to unseen data during inference. In this scenario, the best compromise between model complexity and generalization capability is achieved by a quadratic polynomial function.

The models analyzed so far represent only the model component in the machine learning framework. As discussed previously, overfitting is a common issue when using these relatively simple models. In particular, high-degree polynomial models can overfit the training data very quickly, leading to poor predictive performance on unseen data. A straightforward way to regularize such models is to reduce the degree of the polynomial. However, the literature proposes additional modeling approaches that incorporate regularization mechanisms to mitigate overfitting more effectively. These models will be discussed in the following section.

2.5.3 Ridge Regression

Basically, this kind of model is very similar to a Linear Regressor, with the addition of a regularization term in the cost function used for the training. The equation is the following:

$$J(\theta) = MSE(\theta) + \alpha \sum_{i}^{n} \theta_i^2 \tag{2.13}$$

where the regularization term is $\alpha \sum_i^n \theta_i^2$, that tries to force the weights of the model to be small but also fitting the data distribution. Then MSE is always the Mean Squared Error computed by using the hyper-parameters θ_i.

2.5.4 Lasso Regression

Lasso regressor is another regularized version of Linear Regressor, but with a different cost function with respect to the Ridge regression. In this case the cost function follows this equation:

$$J(\theta) = MSE(\theta) + \alpha \sum_{i}^{n} |\theta_i| \tag{2.14}$$

A key characteristic of the Lasso regressor is its ability to automatically perform feature selection by shrinking the coefficients of less important features toward zero. This property promotes sparse solutions, effectively reducing model complexity. In contrast to Ridge Regression, which admits a closed-form solution as well as iterative optimization methods such as Stochastic Gradient Descent (SGD), the cost function of the Lasso regressor does not have a closed-form solution and is therefore typically optimized using iterative methods, such as SGD, to obtain a sufficiently accurate solution.

2.5.5 Elastic Net

Last but not least among regularized models is the Elastic Net. The Elastic Net can be interpreted as a trade-off between the Lasso and Ridge regressors, as its

regularization term combines the penalties used in the two previous models. In this case, the cost function is defined as follows:

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i}^{n} \theta_i^2 \tag{2.15}$$

where the terms are the same as in the previous models, with the addition of a mixing parameter r. When $r = 0$, the Elastic Net reduces to Ridge Regression, whereas when $r = 1$, it becomes equivalent to Lasso Regression. It is worth noting that Lasso regression can exhibit problematic behavior when the number of features exceeds the number of samples or when some features are strongly correlated. In such cases, Elastic Net is often preferred, as it mitigates these issues by combining the advantages of both Ridge and Lasso regularization.

2.5.6 Logistic Regression

Logistic Regression is another model that is commonly used in machine learning. This model outputs a probability indicating whether a given sample belongs to a particular class. In the case of a binary classifier, if the predicted probability is greater than 50%, the instance is assigned to class 1, whereas if the probability is less than 50%, the sample is assigned to class 0. Similarly to linear regression, Logistic Regression computes a weighted sum of the input features plus a bias term. This quantity is then passed through a nonlinear activation function. The resulting function is defined as follows:

$$p = \sigma(x^T \theta) \tag{2.16}$$

where, θ is always the hyperparameter vector and x is the samples vector, and:

$$\sigma = \frac{1}{1 + exp(-t)} \tag{2.17}$$

where t is the logit. Then, a binary classification could be explained as:

$$y = \begin{cases} 0 & if \quad p < 0.5 \\ 1 & if \quad p \geq 0.5 \end{cases} \tag{2.18}$$

The cost function for a Logistic Regression, also called log loss, is the following:

$$J(\theta) = -\frac{1}{m}\sum_{i}^{m}[ylog(p) + (1-y)log(1-p)] \tag{2.19}$$

where y denotes the target probability, while p represents the probability estimated by the model. Also in this case, no closed-form solution exists for directly computing the optimal model parameters. However, by appropriately selecting the learning rate, it is possible to reach the global minimum of the cost function, since it is convex and therefore admits a unique global minimum. An interesting extension arises when more than two classes must be predicted. In this case, the model is referred to as Softmax Regression, which is described in the next paragraph.

2.5.7 Softmax Regression

This model differs from Logistic Regression in that it is specifically designed for multiclass classification tasks. Given a sample x, Softmax Regression computes a score for each class and estimates the probability that the sample belongs to each class. Formally:

$$s_k(x) = x^T \theta^{(k)} \tag{2.20}$$

where each class has its own parameter vector $\theta^{(k)}$ and s_k is the score for each class k. Thus the probability function is:

$$p = \sigma(s(x)) = \frac{exp(s_k(x))}{\sum_j^K exp(s_j(x))} \tag{2.21}$$

where K is the number of classes and $\sigma(s(x))$ is the estimated probability that x belongs to the class k.

This type of model is often trained using a cost function that differs from those previously introduced. The idea is to employ a loss function that emphasizes the class associated with a given sample by increasing its predicted probability while decreasing the probabilities assigned to the other classes. To achieve this, the Cross-Entropy cost function is commonly used, as it measures how well a predicted

probability distribution matches the target class distribution. The cost function is defined as follows:

$$J(\theta) = -\frac{1}{m}\sum_{i}^{m}\sum_{k}^{K} y_k log(p_k)) \tag{2.22}$$

where y_k is the target probability, p_k is the estimated probability. If the problem is a binary classification ($K = 2$), the cost function assumes the same shape as the log loss already explained in Equation 2.19.

2.5.8 Support Vector Machines

Support Vector Machines (SVMs) are powerful machine learning models that are widely used in modern applications. They are capable of performing both linear and nonlinear classification, particularly on small and medium-sized datasets. Thanks to their versatility, SVMs can also be applied to regression tasks. Considering a binary classification problem, the idea behind SVMs is to find the optimal decision boundary that separates one class from the other. Specifically, an SVM seeks the hyperplane that maximizes the margin, defined as the distance between the decision boundary and the nearest data points, known as support vectors. This formulation is referred to as hard-margin classification. Figure 2.9 provides a graphical representation of SVM-based classification. However, in many practical scenarios, a perfectly separating hyperplane does not exist. In such cases, the model allows for a limited number of classification errors while still maximizing the margin, leading to the soft-margin classification formulation.

The main problem behind this theory is that often the data are not linearly separable, thus a linear margin cannot be found. To overcome this issue, an effective strategy consists in mapping the original features into a higher-dimensional feature space, where the data may become linearly separable. In this mapping, the new features are typically nonlinear combinations of the original ones. While explicitly constructing such feature mappings can be computationally expensive, this problem is addressed by the so-called kernel trick. This technique enables SVMs to operate in high-dimensional feature spaces without explicitly computing the transformed features, by instead relying on a kernel function that implicitly computes inner products in the transformed space. Several kernel functions are commonly used in

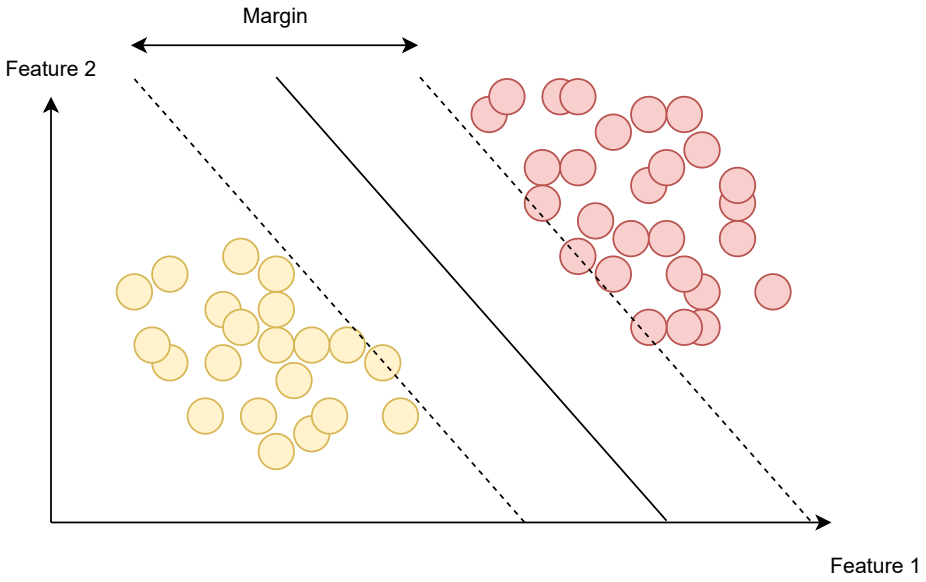


Fig. 2.9 Support Vector Machine Classifier example

practice, including the linear, polynomial, and Gaussian (or Radial Basis Function, RBF) kernels. In particular, the Gaussian kernel is often employed when the data are highly nonlinearly separable. In this case, a regularization parameter γ is introduced to control the influence of individual data points and to regulate the smoothness of the decision boundary. As mentioned earlier, SVM can also be employed for regression tasks. In this case, the objective differs from classification: instead of maximizing the margin to separate classes, the goal is to fit as many data points as possible within a predefined margin, while penalizing margin violations. In Support Vector Regression (SVR), the width of this margin is controlled by a hyperparameter ε. Once the regression function is fitted to the data, it can be used as a standard regressor, similarly to the models discussed in the previous sections.

2.5.9 Decision Trees

Decision Trees are a technique widely used in the data mining and machine learning context. They can be represented as a flowchart-like structure composed of nodes, branches, and leafs. Each internal node corresponds to a decision, each branch represents the outcome of that decision, and each leaf corresponds to the final

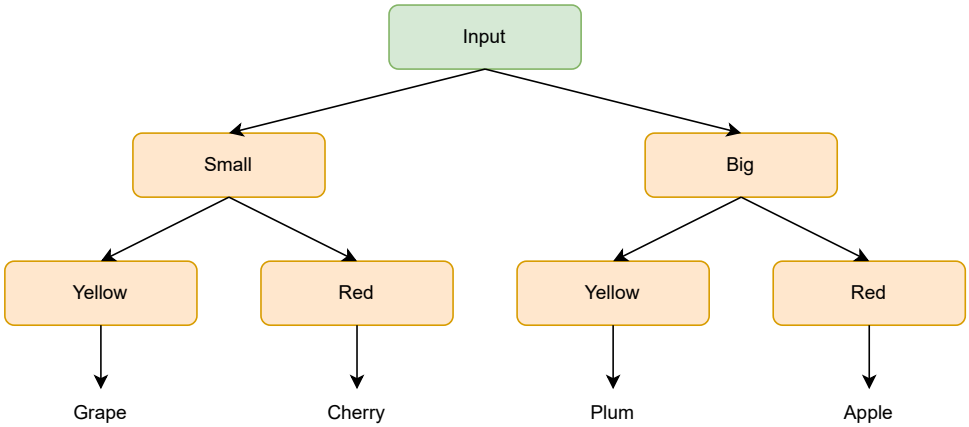


Fig. 2.10 Example of a Decision Tree flowchart

prediction. The hierarchical organization of nodes and branches forms a structure resembling a tree, which is the origin of the name of this model. An example of a Decision Tree architecture is shown in Figure 2.10.

One of the main advantages of Decision Trees is that they can be used for both classification and regression tasks. Depending on the type of problem being addressed, different criteria can be employed to evaluate whether a tree has been trained properly. In the case of regression, the Mean Squared Error (MSE) is commonly used as a cost function to assess the quality of the model. The main limitation of using a decision tree for regression is the granularity of its output values: due to its tree-based structure, it cannot model gradual and continuous patterns smoothly, but instead generates "discretized" predictions determined by the leaves. For classification tasks, a widely adopted metric to evaluate the quality of the splits is the Gini impurity, which is defined as follows:

$$G_i = 1 - \sum_{k}^{n} p_{i,k}^2 \tag{2.23}$$

where $p_{i,k}$ denotes the proportion of instances belonging to class k among the training samples at node i. This metric quantifies the impurity of each node. A high Gini impurity indicates that a node contains a mixture of different classes, meaning that the classification at that node is poorly defined. Conversely, the optimal configuration is achieved when the Gini impurity is close to zero, indicating that the node predominantly contains instances from a single class and that the classification

is well performed. This metric is often compared with the entropy of a node, which is defined as follows:

$$H_i = -\sum_{k,p_{i,k}\neq 0}^{n} p_{i,k} log_2(p_{i,k}) \tag{2.24}$$

The entropy metric is interpreted in a manner analogous to the Gini impurity: a value close to zero indicates that the node is pure, meaning that most of the samples belong to a single class and that the classification is well defined. There is no strict theoretical preference for choosing one metric over the other; however, from a computational standpoint, the Gini impurity is less expensive than entropy, as it does not require the evaluation of logarithmic functions. One of the main advantages of Decision Trees is their high level of interpretability, since the tree structure generated by the algorithm can be easily visualized and analyzed. Moreover, implementations of Decision Trees typically involve a limited number of hyperparameters. These hyperparameters allow control over the depth and complexity of the tree, thereby enabling practitioners to manage overfitting and underfitting of the model.

2.5.10 Random Forest

The Random Forest model is an ensemble learning model that uses different decision trees to take a single decision.

Random Forest combines two fundamental techniques known as Bootstrap Aggregating (bagging) and the Random Subspace method. The former generates multiple subsets of the training data by sampling with replacement, which are then used to train individual decision trees. The latter randomly selects a subset of features at each split, thereby increasing diversity among the trees. This model is widely used due to its intrinsic ability to reduce overfitting. By aggregating multiple decision trees, Random Forest preserves the advantages of individual Decision Trees while significantly improving generalization performance. Similarly to Decision Trees, Random Forest can be applied to both classification and regression tasks. In classification problems, the final prediction is obtained through a majority voting mechanism among the individual trees, where the class receiving the highest number of votes is selected as the final output. The reliability of the prediction generally increases with the number of trees that agree on the same class. For regression tasks, the final prediction is

computed by averaging the outputs of all the trees in the ensemble. One important limitation of random forests in regression is their poor extrapolation ability: since predictions are obtained by averaging the outputs of decision trees built on observed training values, the model cannot naturally extend trends beyond the range of the training data. A graphical illustration of the Random Forest architecture is shown in Figure 2.11.

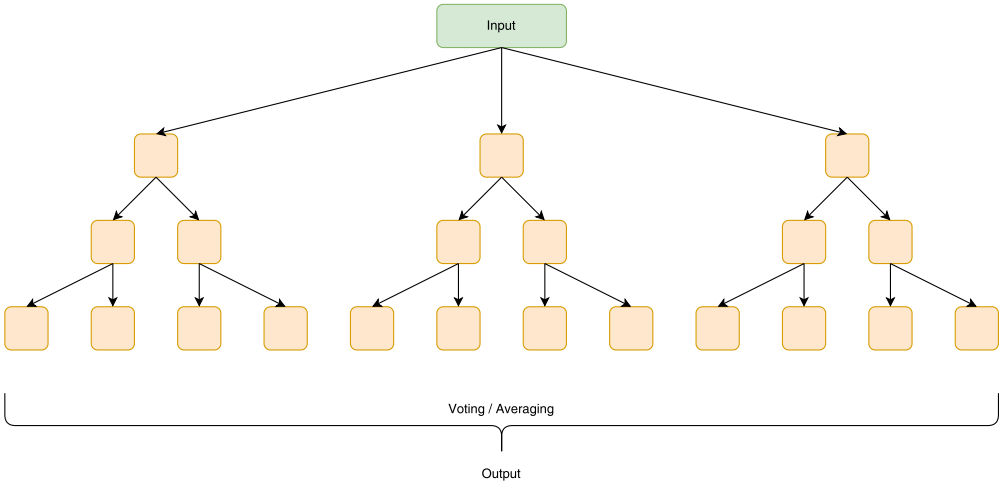


Fig. 2.11 Random Forest example

2.6 Dimensionality Reduction techniques

As previously mentioned, the curse of dimensionality poses significant challenges in machine learning systems, affecting computational complexity, training time, and the risk of overfitting. Consequently, the scientific community has developed various methods and techniques to reduce the number of features in a dataset, retaining only those that are most informative. One of the most widely used algorithms for feature extraction is Principal Component Analysis (PCA).

2.6.1 PCA

This feature extraction method is one of the simplest and most widely used techniques for dimensionality reduction. Its core idea is to project the input samples into a

new feature space, where the original features are combined in such a way that the most relevant information is preserved in a lower-dimensional representation. In practice, this is achieved by identifying the principal components of the training dataset, which correspond to the directions of maximum variance in the data. To compute these principal components, a commonly used technique is the Singular Value Decomposition (SVD). SVD decomposes the training data matrix into three matrices, one of which contains the principal components of the dataset:

$$X = U\Sigma V^T \tag{2.25}$$

$$V = \begin{pmatrix} | & | & & | \\ c_1 & c_2 & \dots & c_n \\ | & | & & | \end{pmatrix} \tag{2.26}$$

where V contains the unit vectors that define the principal components of the training set. Given a dataset with dimensionality d, it is possible to identify a number of principal components less than or equal to d. Dimensionality reduction is achieved by projecting the original data onto the hyperplane spanned by the first principal components. The objective is to select a hyperplane that preserves a predefined proportion of the total variance, which is considered sufficient to retain the most salient information in the data. In practice, a common criterion is to select the smallest number of principal components that preserve more than 90% of the variance of the original dataset. Once the number of dimensions has been selected, the transformed data can be represented using the new features defined by the chosen hyperplane. A graphical representation of the PCA component selection process is shown in Figure 2.12, which follows the rule:

$$X_d = XW_d \tag{2.27}$$

where the W_d is the matrix contains the first principal components selected from the V matrix. X_d is the new dataset with a lower dimension.

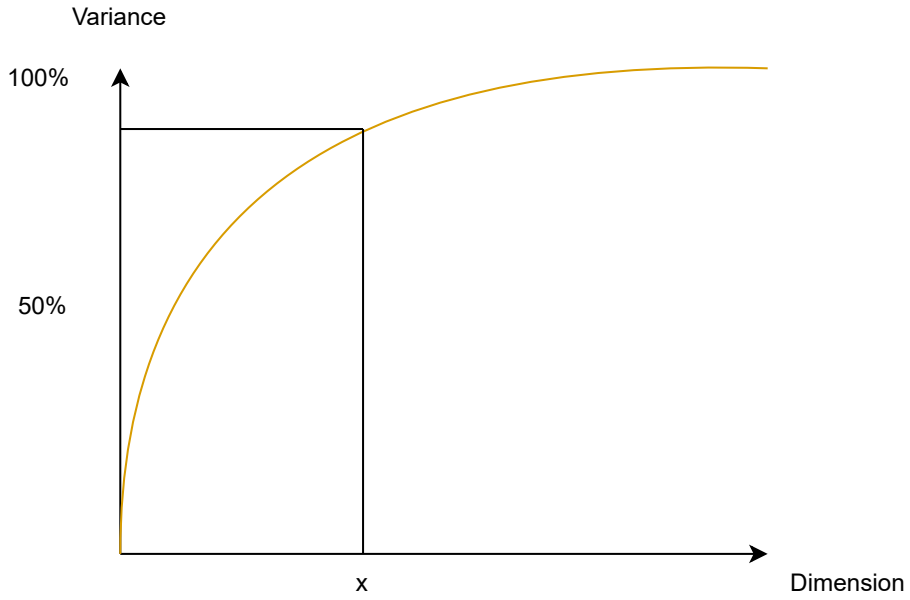


Fig. 2.12 Dimension selection based on the cumulative variance

2.6.2 Other Dimensionality Reduction Techniques

In the literature, several other techniques have been proposed to reduce the number of features in the original dataset. One commonly used method is Linear Discriminant Analysis (LDA), which identifies the most discriminative directions in the feature space and uses them to define a projection hyperplane, in a manner similar to PCA. Unlike PCA, LDA is a supervised technique, as it takes class labels into account to maximize class separability. Another approach, which is discussed in more detail in the following section, is the undercomplete Autoencoder. This is a deep learning model trained to reconstruct its input data. In this setting, the autoencoder is designed with a hidden layer of lower dimensionality than the input, forcing the model to learn a compressed representation of the data. The activations of this hidden layer can then be used as lower-dimensional feature representations.

2.7 Unsupervised Machine Learning models

So far, only supervised machine learning models have been analyzed, in order to introduce techniques that operate when the labels of a dataset are known. In this section, the main models used in the subsequent part of this work—namely, unsupervised machine learning models—are presented. These models are employed when dataset labels are not known a priori. In such cases, the goal is to identify structure in the data by relying solely on the intrinsic features of the samples. These approaches are commonly referred to as clustering algorithms, as they aim to group data points into clusters based on feature similarity. The resulting clusters can then be exploited for exploratory analysis or, in some applications, for downstream classification tasks.

2.7.1 K-Means

One of the most widely used algorithms in unsupervised machine learning is K-Means. This technique is conceptually simple and aims to partition a dataset into a predefined number of clusters by assigning each data sample to the nearest cluster centroid. An example of the K-Means clustering process is shown in Figure 2.13.

In the illustrative example shown, it is relatively easy to identify the cluster associated with each data point; however, this is not generally the case. In particular, due to the curse of dimensionality, a graphical representation such as the one shown is often not feasible. As a result, it is not possible to visually assess whether a clustering algorithm is performing well. For this reason, quantitative evaluation metrics are required. The K-Means algorithm begins by initializing the centroids of a predefined number of clusters, which are typically selected manually. The algorithm then performs a sequence of iterations to progressively refine the positions of the centroids, with the goal of improving cluster separation. At each iteration, a metric known as inertia is computed to evaluate the quality of the clustering and to guide the centroid updates. The inertia metric is defined as follows:

$$I = \sum_{k}^{K} \sum_{x \in C_k} ||x - \mu_k|| \tag{2.28}$$

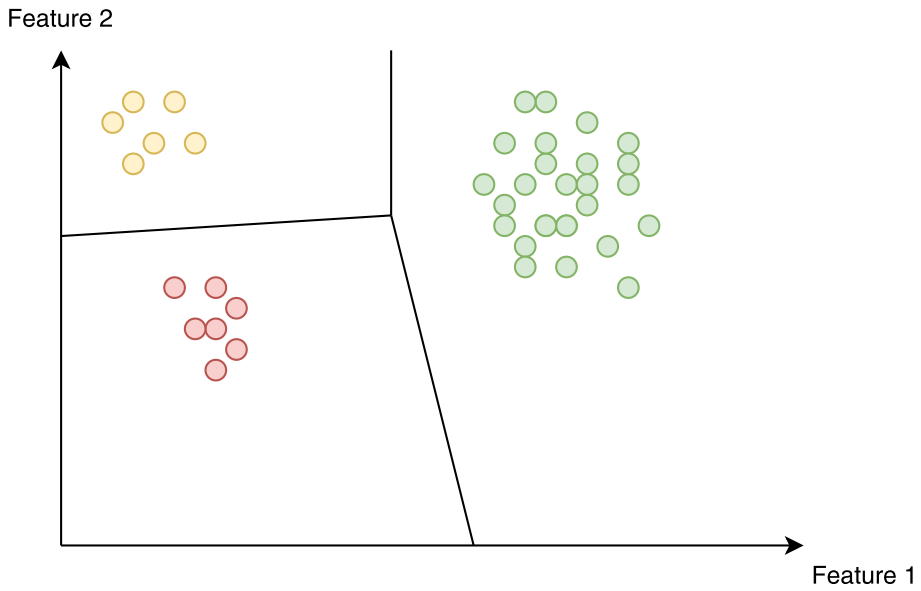


Fig. 2.13 KMeans model example

where K denotes the number of clusters, C_k represents the set of data points belonging to the k-th cluster, x is a data point in C_k, and μ_k is the centroid (mean) of cluster C_k. Inertia corresponds to the sum of the squared Euclidean distances between each data point and the centroid of its assigned cluster. A low inertia value indicates that the chosen number of clusters provides a compact and well-defined partition of the data. A key challenge in K-Means clustering is therefore the selection of an appropriate number of clusters for a given application. When the data dimensionality is low, such as in the two-dimensional example shown in Figure 2.13, selecting the number of clusters can be relatively straightforward. In that case, three clusters clearly provide a good separation of the data. However, as the dimensionality increases, visual inspection becomes impractical. For instance, with datasets containing hundreds of features, it is no longer possible to rely on graphical representations to determine the optimal number of clusters. To address this issue, a commonly used heuristic known as the elbow method has been proposed in the literature. This method consists of plotting the inertia value on the y-axis as a function of the number of clusters on the x-axis. The point at which the curve exhibits a noticeable change in slope—resembling an elbow—indicates a suitable number of clusters that achieves a good trade-off

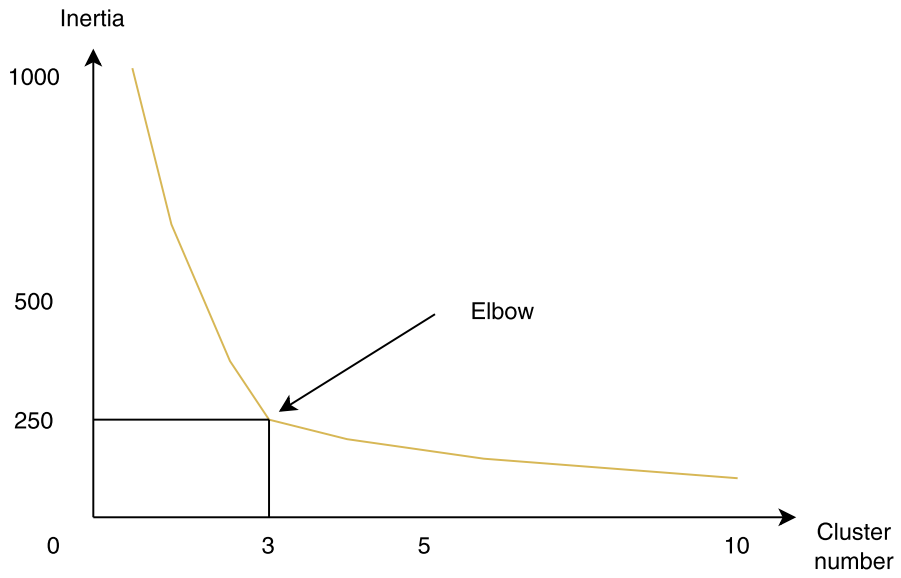


Fig. 2.14 KMeans elbow technique

between cluster compactness and model complexity. An example of the elbow method is shown in Figure 2.14.

This technique provides a practical method for selecting the number of clusters. Its main advantage is the low computational cost; however, a significant drawback is the lack of an objective criterion for identifying the elbow point in the curve, as its interpretation is often subjective. A more objective approach for estimating the optimal number of clusters is provided by another metric known as the Silhouette score. The formulation of the Silhouette score is given by:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{2.29}$$

where $a(i)$ denotes the cohesion, defined as the mean intra-cluster distance of sample i, and $b(i)$ represents the separation, defined as the mean distance between sample i and the points in the nearest neighboring cluster, excluding its own cluster. The Silhouette coefficient takes values in the range $[-1, +1]$. A graphical representation of the Silhouette score is shown in Figure 2.15.

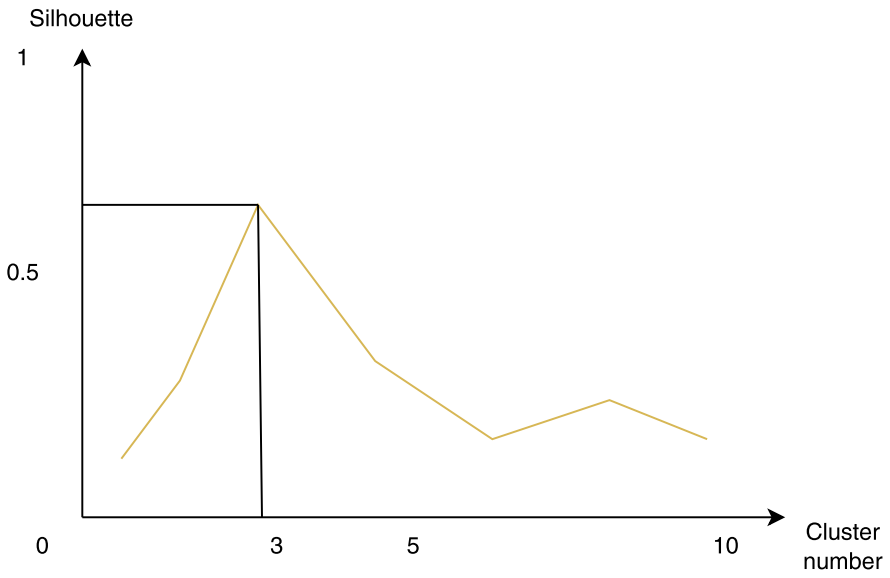


Fig. 2.15 Silhouette score representation

The main limitation of this type of algorithm lies in the assumed shape of the clusters. K-Means performs best when clusters are approximately spherical and of similar size, an assumption that may not hold in many real-world datasets. This limitation can be particularly difficult to identify in high-dimensional spaces. To address this issue, alternative clustering algorithms that do not rely on spherical cluster assumptions have been proposed. One such algorithm is DBSCAN, which is better suited for discovering clusters with arbitrary shapes.

2.7.2 DBSCAN

An example of the application of this type of algorithm is shown in Figure 2.16. As can be observed, the resulting clusters are not spherical, unlike those typically produced by K-Means. In such scenarios, the K-Means algorithm fails due to its intrinsic assumptions about cluster shape, whereas DBSCAN performs effectively thanks to its different clustering strategy. DBSCAN forms clusters by grouping together data points that are within a specified distance of each other. Specifically, a parameter ε is defined by the user to represent the neighborhood radius used to measure distances between instances. If two points lie within a distance smaller than

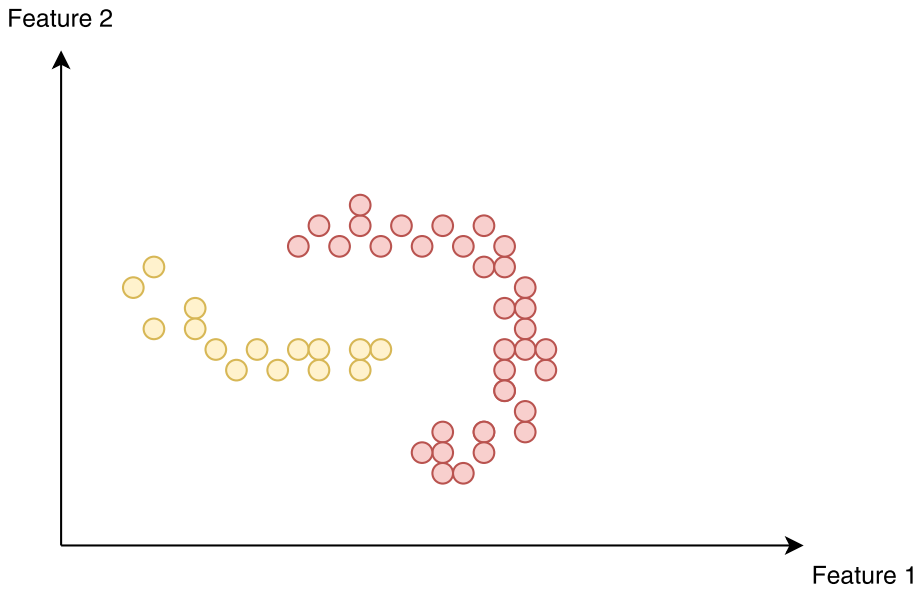


Fig. 2.16 DBSCAN clustering algorithm

ε, they are considered neighbors and may belong to the same cluster; otherwise, they are assigned to different clusters. This density-based approach enables DBSCAN to identify clusters as regions of high data density with arbitrary shapes. Unlike K-Means, DBSCAN does not require the number of clusters to be specified a priori. Data points that do not belong to any dense region and have no sufficient neighborhood are classified as noise or anomalies.

2.7.3 Gaussian Mixtures

This is a particular unsupervised machine learning model that estimates clusters under the assumption that data instances are generated from a mixture of Gaussian distributions. In this framework, each Gaussian distribution corresponds to a cluster, similarly to the clusters produced by K-Means or DBSCAN. Typically, clusters identified by a Gaussian Mixture Model (GMM) exhibit an ellipsoidal shape, with size and orientation determined by the statistical properties of the data belonging to each cluster. Similar to K-Means, the GMM algorithm follows an iterative optimization procedure: the parameters of the Gaussian components are initialized,

often randomly, and then refined through successive iterations until convergence. At each iteration, the model updates the cluster parameters by considering all instances in the dataset and computing the probability that each instance belongs to a given cluster. This probabilistic assignment allows GMMs to model overlapping clusters and capture more complex cluster structures. With respect to evaluation metrics, in this case it is not appropriate to use inertia or Silhouette-based metrics commonly adopted for K-Means. This is because the clusters generated by Gaussian Mixture Models are not necessarily spherical, and therefore cohesion and separation measures based on Euclidean distances may not provide meaningful information. Instead, alternative metrics are typically employed to evaluate the quality of Gaussian Mixture Models, including the following:

$$BIC = log(m)p - 2log(L) \tag{2.30}$$

$$AIC = 2p - 2log(L) \tag{2.31}$$

that are the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC) respectively, where m is the number of instances, p is the number of parameters of the model, and L is the maximized value of the likelihood function of the models itself.

The general objective is to minimize these two metrics in order to obtain a well-performing model. Both criteria penalize overly complex models with a large number of parameters, while rewarding models that fit the data well.

The number of parameters that the model must learn depends on the constraints imposed during training. If the model parameters are left unconstrained, the resulting model is more complex, as it must estimate a larger number of variables. Conversely, if constraints are enforced—such as assuming spherical clusters—the model requires fewer parameters to be learned, leading to lower values of both the metrics.

2.7.4 Other Unsupervised Machine Learning model

In the literature, particularly in the context of anomaly and novelty detection, several additional models have been developed in recent years that are well suited to these tasks. In this section, only the models that are employed in the subsequent parts

of this work are briefly introduced, with the aim of providing an overview of their underlying principles.

Isolation Forest is an ensemble-based method derived from Random Forests and Decision Trees. In this algorithm, each tree is constructed by recursively selecting a random feature and a random split value between the minimum and maximum values of that feature. The key idea is that anomalies, being significantly different from normal instances, tend to be isolated in fewer splits, resulting in shorter path lengths within the trees. These shorter paths are then used as an indicator of anomalous behavior.

Local Outlier Factor (LOF) is a density-based method that measures the local density of a data point relative to the densities of its neighbors. If the local density of an instance is significantly lower than that of its surrounding points, the instance is considered an outlier. Quantitatively, a LOF value close to or less than 1 indicates normal behavior, whereas values significantly greater than 1 indicate anomalous instances.

One-Class Support Vector Machine (One-Class SVM) is an extension of the traditional SVM framework designed for anomaly detection. Instead of separating multiple classes, this model learns a decision boundary that encloses the region containing normal data in a high-dimensional feature space. Instances that fall outside this learned region are then classified as anomalies.

Although many other unsupervised machine learning models and variants exist in the literature, they are not considered relevant to the work presented here.

2.8 Deep Learning

The term Deep Learning has become increasingly common in modern discourse, knowing applications as ChatGPT, Google Gemini and so on. Broadly speaking, deep learning aims to emulate certain aspects of the human brain by performing a large number of simple computations through interconnected structures composed of neurons organized in layers. A deep learning model can be viewed as a composition of multiple layers, each containing a predefined number of artificial neurons. These layers are interconnected in various ways, and given an input to the model, the network produces an output that is defined by the learning objective. As with

traditional machine learning models, the training phase is a fundamental component of deep learning. During training, each neuron learns appropriate weights, which determine how inputs are combined and propagated through the network layers to generate the desired output.

2.8.1 The Perceptron

It is possible to say that the perceptron corresponds to the easiest Deep Learning model. Basically, it is a linear function that follows this rule:

$$Perceptron = \sum_i (x_i w_i) + b > 0 \tag{2.32}$$

$$y = \begin{cases} 1 & if \quad \sum_i (x_i w_i) + b > 0 \\ 0 & otherwise \end{cases} \tag{2.33}$$

where y denotes the output of the perceptron, x_i is the i-th input feature, w_i is the weight associated with the corresponding input, and b is the bias term. During training, the objective is to determine the optimal values of w_i and b in order to produce the desired output y. This formulation can be interpreted as a single artificial neuron, also known as a threshold logic unit. In deep learning, multiple such neurons are combined to form more complex architectures, with the number of neurons depending on the application and the complexity of the task. In principle, a neural network may consist of an arbitrary number of neurons organized into layers. When each neuron in a layer is connected to all neurons in the preceding layer, the layer is referred to as a fully connected (or dense) layer, which is one of the most commonly used layer types in deep learning. In the context of fully connected layers, activation functions play a crucial role, as they introduce nonlinearity by transforming the output of the layer. Consequently, equation 2.34 can be extended as follows:

$$h = \phi(XW + b) \tag{2.34}$$

in vector notation, where ϕ is the activation function.

The training of a perceptron is conceptually simple and is based on the difference between the target output and the predicted output. The update rule follows:

$$\hat{w_{i,j}} = w_{i,j} + \eta(y_j - \hat{y_j})x_i \tag{2.35}$$

where η is the learning rate, i is the input feature, j is the whole instance, $\hat{y}$ is the predicted output while y is the target output wanted.

2.8.2 Multilayer Perceptron

A Multilayer Perceptron (MLP) is composed of multiple single perceptrons organized into layers. A typical architecture consists of an input layer, one or more hidden layers, and an output layer. A crucial aspect of this model is the training process, which is carried out using the backpropagation algorithm, originally introduced by Rumelhart, Hinton, and Williams in 1985 [30]. Backpropagation is based on Gradient Descent optimization and enables the automatic computation of gradients for all network parameters. The training process begins with a forward pass, during which the input data are propagated through the network from the input layer to the output layer, producing a prediction. Subsequently, a backward pass is performed, propagating the error from the output layer back toward the input layer. Through this backward propagation of errors, the algorithm computes the necessary corrections for each weight and bias in the network. These parameters are then updated accordingly, and the process is repeated iteratively until convergence to a (local) minimum of the loss function is achieved.

In practice, the algorithm processes the training data using mini-batches, which are subsets of the full dataset. This procedure continues until the entire dataset has been processed. Repeating this process over the full dataset multiple times results in several training iterations, known as epochs.

For each mini-batch, the data are propagated from the input layer through the successive layers up to the output layer, following the forward pass. At this stage, the algorithm computes the output error by comparing the predicted output with the target values. Subsequently, the error is propagated backward through the network using the chain rule, allowing the algorithm to determine how much each connection contributes to the observed error. This process is referred to as the backward pass.

Based on the computed error gradients, a Gradient Descent step is performed to update the weights and biases of the network. In early neural network models, the

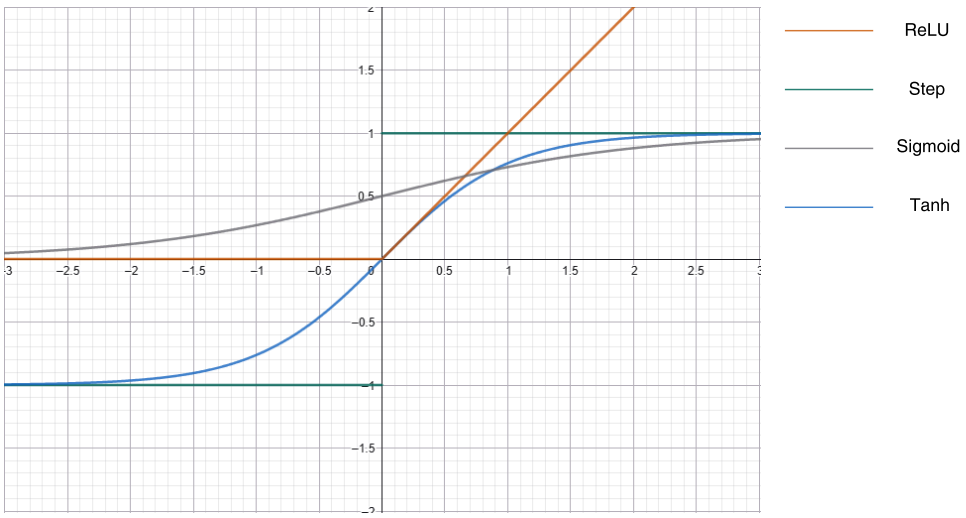


Fig. 2.17 Common activations functions representation

step function was used as the activation function to adjust these parameters. However, due to the discontinuity of the step function, alternative activation functions were introduced to enable effective gradient-based optimization.

One of the first widely adopted alternatives was the sigmoid function, whose smooth and differentiable shape addresses the discontinuity issue. Subsequently, other activation functions were proposed, such as the hyperbolic tangent function and the widely used Rectified Linear Unit (ReLU). Activation functions introduce nonlinearity into the network, enabling it to model complex relationships. In fact, the combination of multiple nonlinear activation functions allows neural networks to approximate any continuous function.

A graphical representation of the most common activation functions discussed above is shown in Figure 2.17. Having analyzed the fundamental principles of how a deep learning algorithm operates, it is now possible to examine the most common types of layers in order to better understand their roles within the deep learning framework.

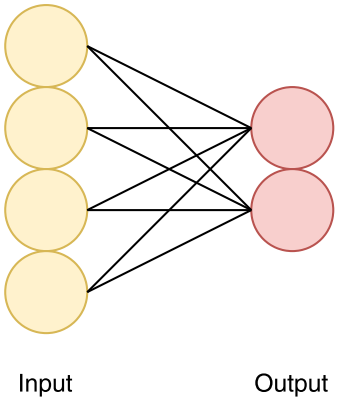


Fig. 2.18 Fully Connected Layer

2.8.3 Fully Connected Layer

Perhaps the most common type of layer in deep learning architectures is the fully connected layer. The defining characteristic of this layer is that every output of the preceding layer is connected to every neuron in the subsequent layer.

This kind of layer sometimes is also called Dense layer. This kind of layer follows the equation we saw previously, thus:

$$y = f(Wx + b) \tag{2.36}$$

where y denotes the output vector, x is the input vector, W represents the weight matrix, and b is the bias vector. As illustrated in Figure 2.18, each input neuron is connected to every output neuron in a fully connected layer.

Simple neural network architectures are often composed exclusively of fully connected layers. Moreover, these kind of layers are frequently used as the output layers of a network to produce outputs suitable for classification or regression tasks. A common design strategy involves starting with a high-dimensional input and progressively reducing the number of neurons across successive layers, until the desired output dimensionality is reached.

Due to their role in decision making, fully connected layers are typically placed at the end of more complex neural network architectures. However, as the number of layers and neurons increases, training such networks becomes increasingly challenging. In particular, excessively complex architectures with a large number of parameters are prone to overfitting during training, owing to their high representational capacity.

2.8.4 Convolutional Layer

This type of layer is commonly used when processing image data. In particular, within the field of Computer Vision, the convolutional layer serves as a fundamental building block for both simple and complex neural network architectures. Convolutional layers are based on the convolution operation, which applies a set of learnable filters to the input data in order to extract local and spatial features. Each convolutional layer operates by sliding a filter (or kernel) over the input array and computing feature maps that capture condensed and informative representations of the input. These layers are typically followed by additional layers that further process and reduce the extracted information. A convolutional layer is defined primarily by its filters, which determine the local substructures of the input that the layer analyzes. Depending on the choice of filter size, stride, and padding, the resulting output may have spatial dimensions that are equal to, smaller than, or larger than those of the input. Each filter produces a distinct feature map, and the total number of filters determines the number of output channels generated by the convolutional layer.

Consider a two-dimensional convolutional layer as an example. As illustrated in Figure 2.19, the kernel is a two-dimensional matrix with predefined coefficients. The convolution operation can be interpreted as a sliding-window process, in which the kernel is moved across the input matrix. At each position, the elements of the kernel are multiplied element-wise with the corresponding elements of the input, and the resulting products are summed to produce a single output value. In the literature, many different types of kernels have been proposed, each designed to extract specific features; the structure of the kernel therefore determines the type of convolution being performed. In the example shown in Figure 2.19, the convolution operation results in a reduction of the original matrix dimensions. This dimensional change depends on the padding strategy adopted. In the illustrated case, no padding is applied: starting from the top-left corner, the kernel slides across the matrix, producing an output

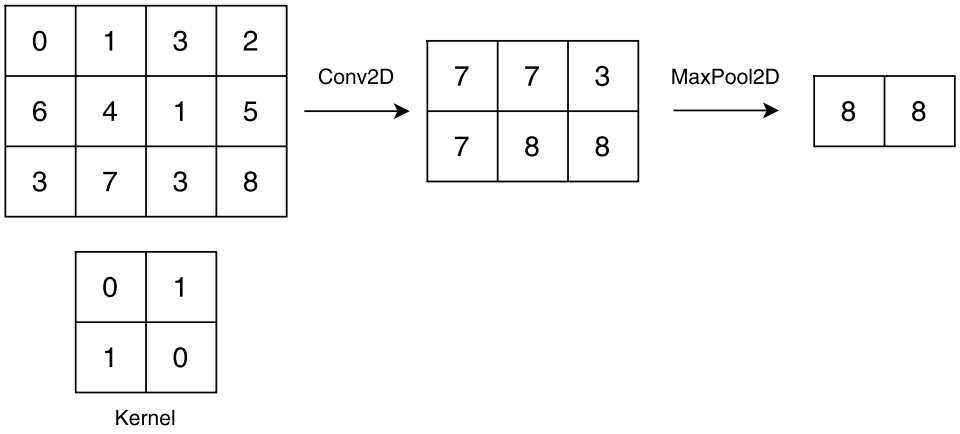


Fig. 2.19 Example of a 2D Convolution with a 2x2 Kernel and a 2D MaxPooling 2x2

matrix smaller than the original one. This approach is commonly referred to as valid padding. Alternatively, same padding preserves the original spatial dimensions of the input, while full padding produces an output matrix with larger dimensions than the input. In most architectures, the convolutional layer is followed by an additional operation aimed at enhancing the extracted features and reducing the spatial dimensionality, namely a pooling layer. Several pooling strategies exist, the most common being max pooling, which selects the maximum value within the pooling window and discards the remaining values. Another widely used approach is average pooling, which computes the mean of the values within the pooling window. In Figure 2.19, a two-dimensional convolution is followed by a two-dimensional max pooling operation, which reduces the original 3×4 input matrix to a 1×2 output array.

Convolutional layers and networks are widely used, particularly in the fields of computer vision and image processing. Thanks to the intrinsic structure of the convolution operation, these layers are highly effective at extracting the most relevant spatial features from images, such as edges, textures, and shapes. However, computer vision is not the only domain in which convolutional layers are applicable. These layers can also be successfully employed with other types of data, especially when meaningful local patterns or feature variations are present. In such cases, convolutional architectures can be highly effective at identifying and modeling such changes within the data.

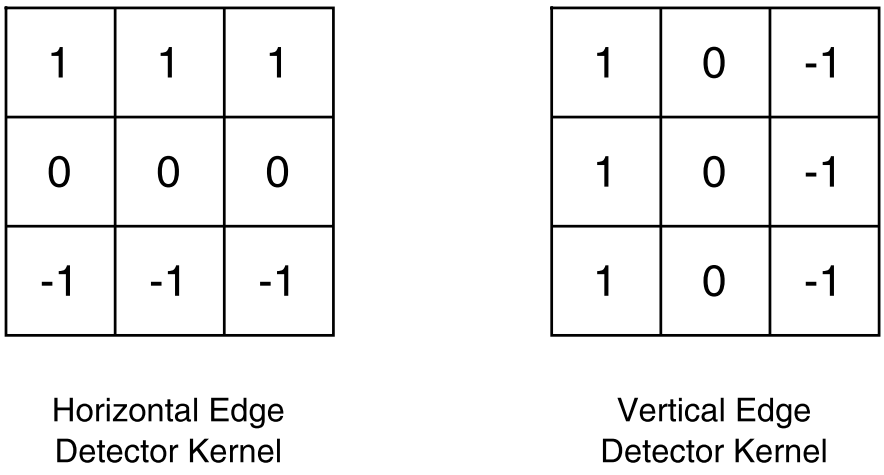


Fig. 2.20 Edge Detection Kernel examples

In the literature, many different convolution kernels have been proposed, each designed to perform specific operations. Examples of kernel shapes are shown in Figure 2.20, where two kernels for vertical and horizontal edge detection are illustrated. In practical applications, kernel values are not limited to ± 1 and may take a wide range of real values. Nevertheless, these simple kernels effectively illustrate the underlying principle of convolutional filtering, as they highlight edges in the input image by emphasizing regions with strong intensity variations.

In deep learning, convolutional layers operate according to the principles described above; however, the kernel parameters are not fixed, but instead learned through the backpropagation algorithm. Specifically, the user defines the kernel size and the number of kernels (filters), while their values are automatically optimized during training in order to minimize the loss function. In a typical convolutional neural network, the overall architecture resembles the one illustrated in Figure 2.21. In this example, each yellow block represents a combination of a convolutional layer followed by a max pooling layer, which progressively reduces the spatial dimensions of the feature maps. The number of channels in each layer is indicated by the last dimension of the layer shape. In this case, the first convolutional layer produces 10 channels, the second produces 20 channels, the third produces 40 channels, and the final convolutional layer produces 80 channels. In most architectures, the convolutional component of the network is followed by a flatten layer, which converts

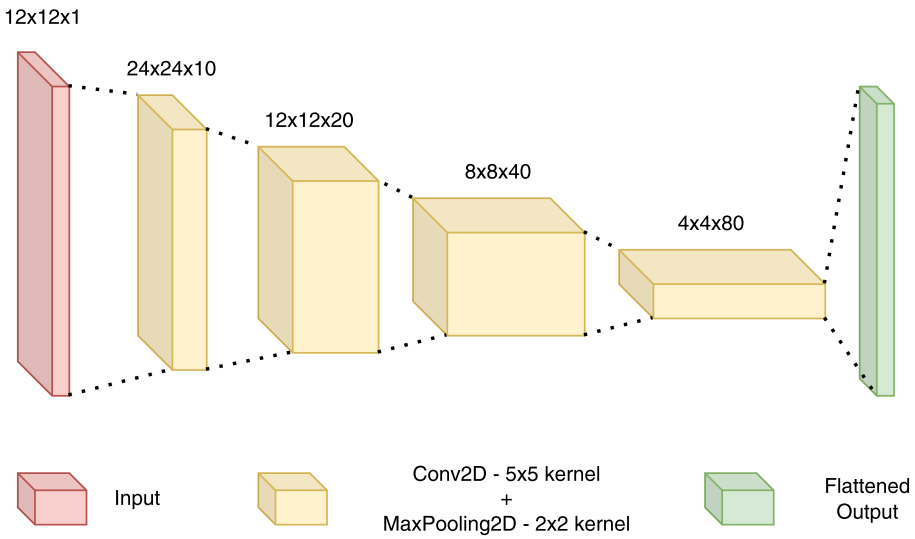


Fig. 2.21 Convolutional Network Example

all feature maps and channels into a single one-dimensional vector representing the extracted features. During the convolution and pooling operations, the spatial dimensions of the input are progressively reduced through the sliding-window mechanism. During training, each convolutional kernel is optimized via backpropagation to learn the values that best extract features relevant to the desired output. Indeed, after the flatten layer, other fully connected layers are inserted in the algorithm in order to obtain a target values that have the dimension as the input target used for training the algorithm.

2.8.5 Recurrent Layer

Last but not least, the final layer analyzed in this thesis is the recurrent layer, which is widely used in the scientific community. This type of layer is particularly well suited for time-series data, where temporal dependencies play a fundamental role. Recurrent layers are especially valuable in deep learning because they are capable of retaining information from previous inputs, effectively providing the model with a form of memory. This property is crucial when modeling sequential data, as the current output may depend not only on the present input but also on past observations. The memory mechanism of recurrent layers is described by the following equation:

$$y_t = \phi(W_x^T x_{(t)} + W_y^T y_{(t-1)} + b) \tag{2.37}$$

where ϕ is the activation function, W_x and W_y are the weight matrices for the actual input x and the previous output y respectively.

2.9 Deep Learning architectures

A neural network architecture refers to the way in which the layers introduced in the previous section are organized and connected to perform a specific task. In the literature, many different architectures have been proposed, each tailored to particular types of problems, such as signal analysis or image processing. In this chapter, only the most common architectures found in the literature are discussed, with a focus on applications involving time-series signals and images.

2.9.1 Autoencoder

One of the most widely used architectures in the novelty detection literature, and the primary architecture employed in the subsequent sections of this work, is the Autoencoder. This architecture is composed of two main components: an encoder, which maps the input data into a latent representation of higher or lower dimensionality, and a decoder, which reconstructs the original input from this latent space. When the latent space has a higher dimensionality than the input space, the model is referred to as an overcomplete autoencoder, whereas when the latent space has a lower dimensionality, it is called an undercomplete autoencoder. Depending on the specific application and task, one formulation may be more suitable than the other. A schematic representation of the autoencoder architecture described above is shown in Figure 2.22. Formally, let $X \in \mathbb{R}^n$ represent the input data, where n denotes the dimensionality of each sample. The autoencoder comprises the encoder, $f_{enc} : \mathbb{R}^n \to \mathbb{R}^p$, and the decoder $f_{dec} : \mathbb{R}^p \to \mathbb{R}^n$. This facilitates the transformation of input data into a latent space $h = f_{enc}(x)$ and a subsequent reconstruction $X_{recon} = f_{dec}(h)$. Overcomplete autoencoder has $p > n$ while undercomplete autoencoder has $p < n$.

The fundamental idea behind an autoencoder is to reproduce the input at the output, thereby forcing the model to automatically learn a latent representation that

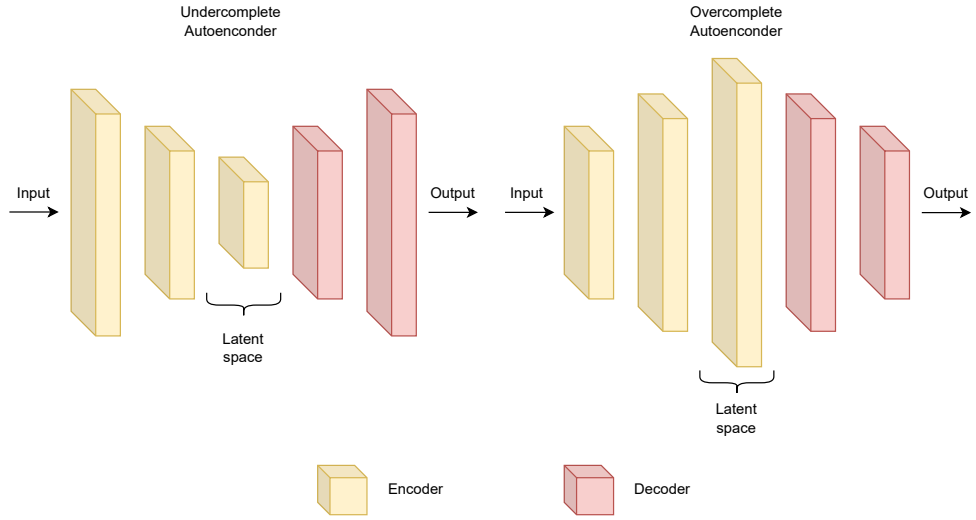


Fig. 2.22 Autoencoder architecture representation. Undercomplete autoencoder on the left, overcomplete autoencoder on the rigth.

captures the most informative features required for accurate reconstruction. As a result, autoencoders can also be employed as feature extractors, in a manner similar to Principal Component Analysis, as discussed previously. A key advantage of autoencoders is their versatility, as they can be adapted to different data types and tasks. One of the most common variants is the fully connected autoencoder, which is composed exclusively of dense layers and is particularly suitable for handling data from multiple domains. When dealing with image data, however, fully connected autoencoders may not be optimal for capturing spatially meaningful features. In such cases, convolutional autoencoders are more effective, as they exploit the local spatial structure of images. Similarly, for time-series, sequential, or signal data, recurrent autoencoders can be employed to better capture temporal dependencies in the latent representation. In addition to the choice of layers that define the architecture, several specialized variants of autoencoders have been proposed for specific applications. These include sparse autoencoders, denoising autoencoders, and variational autoencoders. A sparse autoencoder is typically an overcomplete autoencoder that enforces sparsity constraints on the latent representation, resulting in an output where many activations are close to zero while the relevant information is distributed across a limited number of active features. A denoising autoencoder is designed to learn robust representations by reconstructing clean inputs from corrupted versions during training. Through this process, the model learns to capture

the most salient features of the data while ignoring noise. Consequently, during inference, it is able to distinguish meaningful information from noise, effectively separating signal from unwanted disturbances. This approach can be applied to tasks such as noise reduction in audio signals recorded in crowded environments or image restoration in the presence of blur or other distortions. Finally, the variational autoencoder (VAE) differs from standard autoencoders in that it outputs two vectors: one representing the means and the other representing the standard deviations of a latent probability distribution. Rather than learning a deterministic mapping, the VAE learns a continuous probabilistic representation of the data. As a result, this type of autoencoder can also be used to generate new synthetic samples by sampling from the learned latent distribution.

2.9.2 LeNet

Layer	Type	Kernel/Filter Size	Stride	Output Feature Map Size
Input	Image	N/A	N/A	32×32×1
C1	Convolutional	5×5	1	28×28×6
S2	Average Pooling	2×2	2	14×14×6
C3	Convolutional	5×5	1	10×10×16
S4	Average Pooling	2×2	2	5×5×16
C5	Convolutional	5×5	1	1×1×120
F6	Fully Connected	N/A	N/A	84
Output	Fully Connected	N/A	N/A	10

Table 2.1 LeNet-5 Architecture

The LeNet-5 architecture is illustrated in Table 2.1. Through a sequence of convolutional layers, this architecture is able to recognize handwritten digits from the MNIST dataset, starting from grayscale images of the digits. The final output layer consists of 10 neurons, each corresponding to one of the digit classes, representing the predicted class of the input image.

2.9.3 AlexNet

The AlexNet architecture is another widely used model in the literature and is capable of performing image classification tasks with high accuracy. Unlike the LeNet

Layer	Type	Kernels / Output Size	Kernel Size	Stride / Pool Size	Output Size
Input					227×227×3
1	Conv	96	11×11×3	Stride 4	55×55×96
(1b)	Max-Pool			Pool 3×3, Stride 2	27×27×96
2	Conv	256	5×5×48	Stride 1, Pad 2	27×27×256
(2b)	Max-Pool			Pool 3×3, Stride 2	13×13×256
3	Conv	384	3×3×256	Stride 1, Pad 1	13×13×384
4	Conv	384	3×3×192	Stride 1, Pad 1	13×13×384
5	Conv	256	3×3×192	Stride 1, Pad 1	13×13×256
(5b)	Max-Pool			Pool 3×3, Stride 2	6×6×256
6	Fully Connected (FC)	4096			4096
7	Fully Connected (FC)	4096			4096
8	Fully Connected (FC) / Output	1000			1000

Table 2.2 AlexNet Architecture

architecture, AlexNet requires an input image of size $227 \times 227 \times 3$, corresponding to an RGB image rather than a grayscale one. AlexNet represented a significant advancement in image classification, owing to its deeper architecture and its ability to effectively handle higher-resolution, multi-channel images.

2.9.4 ResNet

The main advantage and innovation of this model lies in the use of skip connections within the network architecture, which distinguish it from the previously discussed architectures. These connections allow information to bypass one or more layers, facilitating the training of very deep networks. ResNet architectures have been successfully applied to a wide range of tasks in the image domain, including image classification, object detection, and semantic segmentation.

Chapter 3

Novelty Detection on Industrial Context

In this section, the works and publications developed during my PhD are presented, with the aim of applying the theoretical concepts discussed in the previous chapter in practice. This part of the thesis is organized into two main macro-sections, consisting of Sections 3 and 4. This first macro-section focuses on an industrial context and investigates the application of novelty detection techniques to industrial mechanical components [2, 31].

3.1 A Real-Time Novelty Recognition Framework Based on Machine Learning for Fault Detection

The first work presented in this thesis concerns the application of novelty detection techniques to an industrial bearing monitoring context. Specifically, this study demonstrates how the novelty detection paradigm can be employed to identify deviations in bearing behavior over time, enabling the detection of early warning conditions that should be considered by maintenance personnel.

3.1.1 Introduction

In this work, the research is focused within the broader context of the Industry 4.0 paradigm, which has driven the rapid integration of advanced technologies into modern company environments. A central pillar of this context is the application of Artificial Intelligence to manufacturing processes, specifically to automate the detection of system anomalies. The cost of unplanned failure occurring in industrial machines is estimated at USD 50 billion annually, as reported in the analysis by [32]. An explanation of how predictive maintenance is able to reduce the overall costs by 5–10% and the standard maintenance costs by 20–50% is reported in [33], highlighting its impact on the company's economic aspect.

In this work's introduction, the critical evolution of maintenance strategies is analyzed by comparing traditional preventive maintenance, which relies on periodic, often inefficient component replacement, with Predictive Maintenance (PdM), the solution that companies are actually trying to implement. The advantage of this paradigm is that, by using real-time monitoring to forecast failures before they occur, it can minimize the costs associated with unnecessary replacements and reduce harmful downtime.

A significant challenge is the limitation of conventional investigation techniques. Many existing industrial solutions rely on monitoring the deviation of statistical features from standard operating conditions. However, the transformation of raw data into extracted statistical features frequently results in a loss of critical information, potentially obscuring subtle signs of degradation. Consequently, a growing need for more sophisticated Machine Learning and Deep Learning methodologies capable of handling complex data behaviors and facilitating the estimation of the Remaining Useful Life of components is created.

To address these challenges, a framework focused on Novelty Detection is introduced. Unlike standard anomaly detection tasks [34–36] that require labeled instances of every possible failure, this approach aims to identify when the behavior of incoming data diverges from a learned "normal" or "nominal" baseline, as said previously. This capability is particularly valuable for detecting changes induced by external events, such as environmental shifts or maintenance operations, and assists companies in gathering preliminary datasets for future predictive modeling. The architecture employed a sensor fusion approach, integrating data from multiple sources

into an optimized software system designed to enhance computation scalability and reduce response latency. Finally, the methodology of this study is detailed, which includes a theoretical analysis of the architecture and an evaluation of the framework through two real-world industrial case studies.

3.1.2 Related Works

In this section, the current landscape of Fault Detection (FD) and Predictive Maintenance (PdM) is reviewed, categorizing the existing methodologies into three primary approaches: traditional statistical techniques, supervised machine learning models, and unsupervised novelty detection strategies. Traditional approach typically rely on monitoring specific statistical features such as the mean, standard deviation, skewness, and kurtosis—extracted from raw sensor data [37]. While these techniques are computationally lightweight and easy to interpret, they suffer from significant limitations. Specifically, the process of aggregating raw signals into simple statistical summaries often results in a loss of information, masking subtle temporal patterns or complex correlations that are indicative of early-stage degradation. Furthermore, these methods usually depend on static, pre-defined thresholds (e.g., warning and alarm limits) that lack the flexibility to adapt to dynamic operational contexts.

Algorithms such as Support Vector Machines (SVM), Random Forests, and Artificial Neural Networks (ANN) have demonstrated high accuracy in classifying known fault conditions [38, 39]. The main problem is that supervised models require massive, balanced datasets containing labeled examples of every possible failure mode. In high-quality manufacturing environments, failures are rare events, making it virtually impossible to gather sufficient "faulty" data to train these models effectively. This dependency renders supervised approaches impractical for new machinery or systems where historical failure data is nonexistent, as mentioned in Chapter 1.

To address these shortcomings, the emergence of Unsupervised Learning [40, 41] and Novelty Detection [42] is analyzed. This last paradigm flips the problem statement: instead of learning what a fault looks like, the model learns to characterize "normal" behavior. Various techniques exist in this domain, including clustering methods and reconstruction-based approaches like Autoencoders. By training exclusively on healthy data, these models can flag any significant deviation as an "anomaly".

A common Novelty Detection algorithm often uses complex algorithms to perform its tasks, but sometimes, highly skilled systems are unavailable (e.g., embedded systems). The work developed in this paper aims to give an alternative use of ML that differs from common utilization, able to perform novelty recognition in real-time, requiring very few computational efforts. Easy mathematical computations allow us to use an embedded device as a framework's container, which is not common in the known approaches. The approach is based on a real-time ML error computation between predicted and real data to understand if the incoming information is similar to the trained ones; in case of deviations, a domain expert understands if the system's status is healthy or faulty. This technique can be used without statistical features but with proper raw data preprocessing, as reported in Section 3.1.3.

A comparison between known algorithms and the one used in this paper can be summarized in the following steps:

- Low computational effort required: thanks to easy mathematical computations and the models selected, this algorithm requires low energy in terms of computational effort;
- Real-time: thanks to the low computational effort, the system can work in real-time. A resume table at the end of this work reports the small time required for each framework's stage;
- Flexibility and scalability: the framework has a general template in which different sensors can collect data to understand if some of them are changing from the trained one. The sensor addition is always possible without changing anything in the framework, but just retraining/updating the model with new data added;
- Data collection: using this framework, healthy and faulty data can be collected in order to build a complete dataset for a future PdM implementation.

3.1.3 Methodology

In this chapter, a detailed explanation of the elements that compose the framework is provided in order to highlight the functions of the elements to better understand the complete pipeline of the novelty detection framework. In Figure 3.1, there is a representation of the whole pipeline.

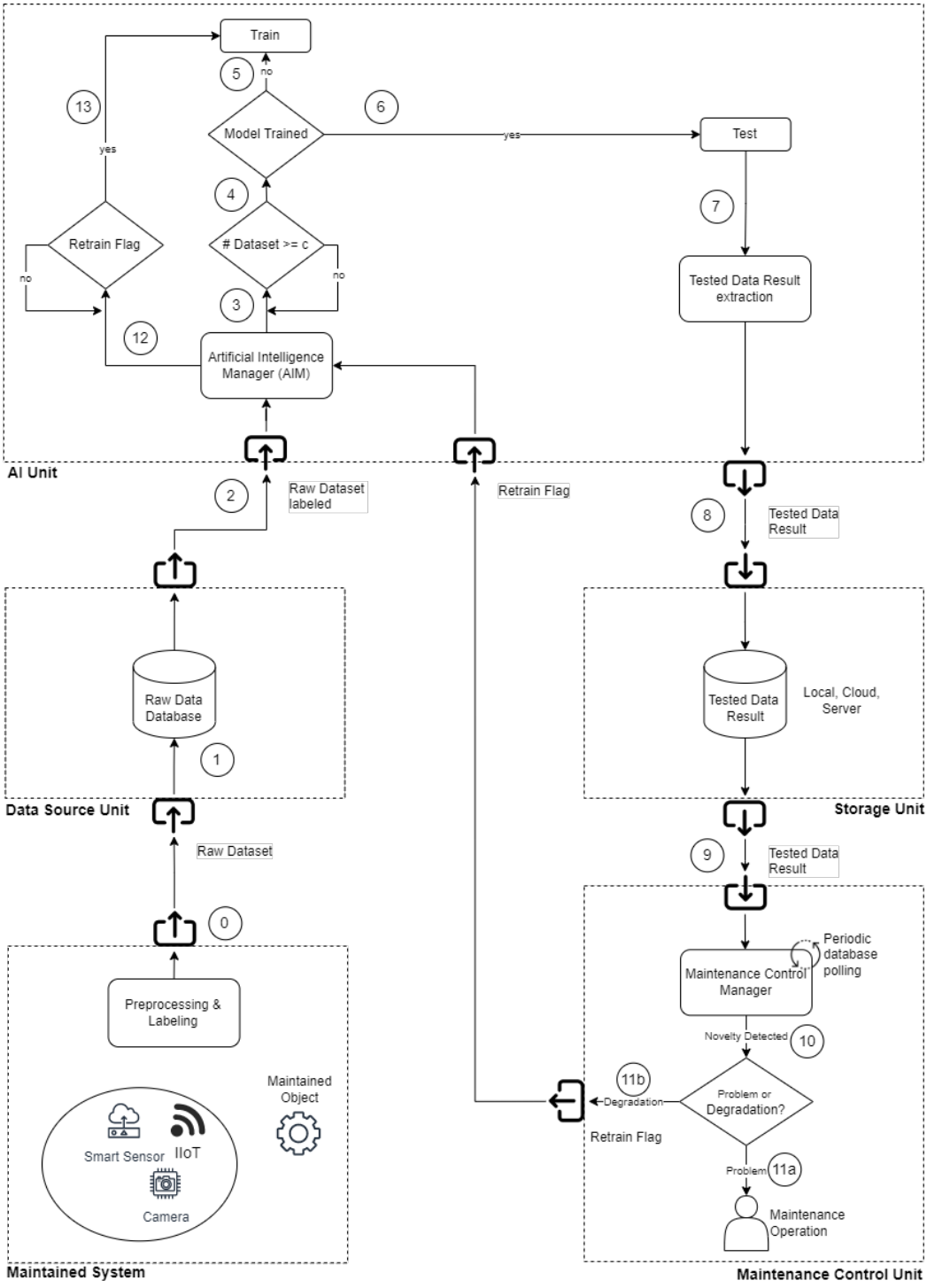


Fig. 3.1 Novelty Detection framework pipeline. Raw data are preprocessed and provided to the AI unit for either training or testing. The Maintenance Control Unit determines whether degradation is present.

Maintained system This part introduces the system integration module, which provides the interface between the hardware components under investigation and the software architecture. Data acquisition is facilitated by various elements, including, but not limited to, sensors, cameras, and machine logs. The foundational premise of this framework is the identification of pattern discrepancies through the comparison of real-time data against a pre-trained reference dataset. System environments are categorized into two types: those maintaining consistent operational parameters (e.g., fixed-speed motors) and those exhibiting highly variable, time-dependent parameters (e.g., vehicle powertrains). For variable systems, establishing a consistent reference necessitates constraining the operating conditions during training (e.g., recording the same motion profile). This unit also executes data normalization, specifically data sizing, to ensure all collected records, subsequently termed a "Dataset", possess an identical number of samples for reliable predictive modeling.

Data Source Unit This component is responsible for storing the processed data in a local or cloud-based database. Data are formatted into a framework-specific table where each row is a record, and each column represents a characteristic feature (like temperature, acceleration, or sound) used for pattern analysis. Feature names label the columns, as shown in Table 3.1.

id	**Timestamp**	**Fs**	**Acc_1**	**Acc_2**	**...**	**Acc_n**	**Sound_1**	**...**	**Sound_n**
...	...	...	...	...	...	...	...	...	...

Table 3.1 Input dataset template example

AI Unit The AI Unit is shown in Figure 3.2, the framework's "brain". This paper proposes an ML application that deviates from the conventional workflow where datasets are rigidly separated into Training, Testing, and Validation sets. Instead, this framework operates in a continuous testing mode, calculating a prediction error for every new dataset received by the database. Monitoring this error is key to identifying data pattern deviations. A low error indicates that the collected data aligns with the original training data. Conversely, an increasing error signals a machine issue or the emergence of a novelty (e.g., environmental changes) due to altered data patterns. This detection method is therefore effective for both fault recognition and identifying significant operational shifts.

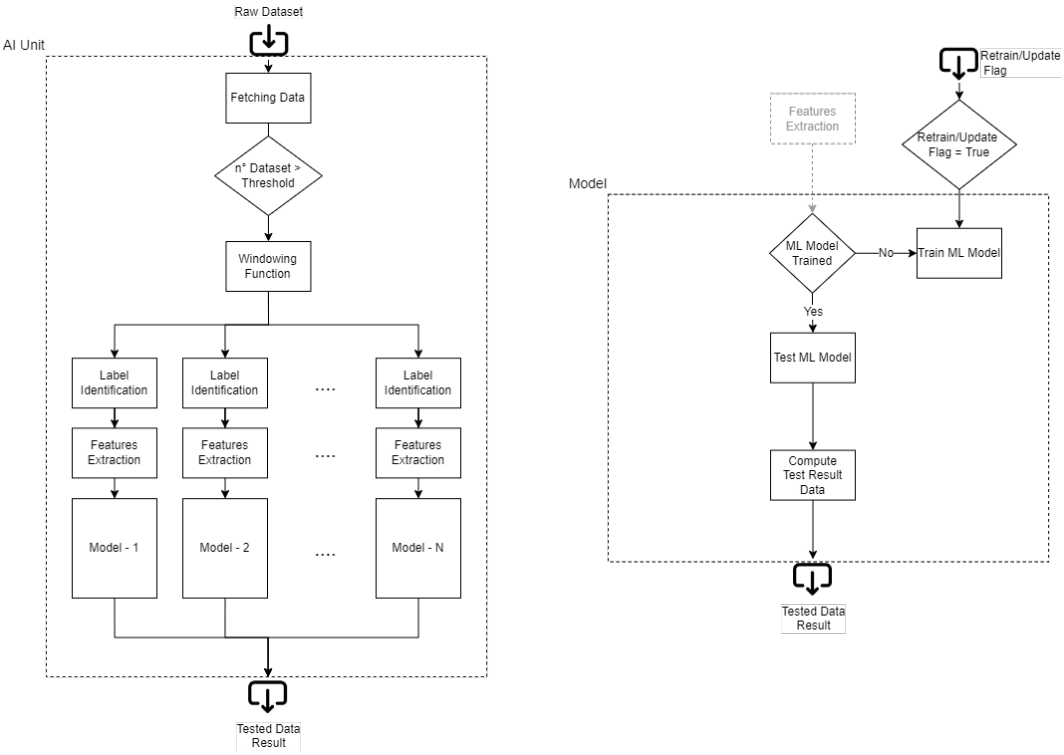


Fig. 3.2 AI unit architecture (left) and ML model (right). The AI unit contains one model for each feature computed using the windowing function. The ML model can operate in either training or testing mode, depending on the user-selected flag.

To ensure effectiveness, the trained dataset must accurately represent the machine's full range of normal, healthy behavior. The choice of training dataset size involves a trade-off:

- Larger Datasets: Increase prediction accuracy and minimize the risk of false positives but necessitate a longer training phase.
- Smaller Datasets: Allow for earlier deployment, but require the maintainer to perform more frequent updates, as the system is more sensitive to operational changes.

A balance must be struck concerning the training duration. The ideal time for training is often during the machine's initial calibration and testing period (which can last from a few days to a month). Collecting data across a full cycle of environmental changes (e.g., day, night, temperature shifts) is beneficial for capturing all component variations. While there is no universal rule, a suggested guideline is to collect one

week's worth of data to ensure consistency. The best training opportunity is at the machine's assembly stage. However, training can also occur later in its life. In this scenario, the framework cannot detect preexisting faults due to the absence of a healthy baseline. Nonetheless, because machine failures are progressive, the system's adaptability allows it to detect the machine's worsening condition, even if trained on a partially faulty system. While the lack of a perfectly healthy dataset may affect accuracy, it does not prevent the framework from operating.

The first stage of the ML process is preprocessing. This involves fetching datasets from the database and identifying the features they contain (e.g., Acceleration, Sound, or Temperature). The AI manager then applies a windowing function to each feature type. This technique segments the large feature set into a parameterized number of smaller windows (e.g., acceleration into k_1 windows, sound into k_2 windows). The number of windows chosen determines the system's recognition resolution: more windows yield higher resolution but increase complexity. Each window is assigned to an Artificial Intelligence Unit Predictor (AIUP) (see Figure 3.2). The i-th window serves as the label, and the remaining windows act as features for that specific AIUP. The windowing approach helps condense the initial data amount for computation (in this paper, using a cumulative sum for each window). The windowing function divides the original data vector $\mathbf{X}$ of length n into smaller segments: Original Data: $\mathbf{X} = [x_1, x_2, \ldots, x_n]$ Window Segmentation:

$$[x_1, x_2, \ldots, x_n] \rightarrow [x_1, x_2, \ldots, x_m] \quad \text{with } m < n \tag{3.1}$$

$$[x_{m+1}, x_{m+2}, \ldots, x_{m+k}] \quad \text{with } m < m+k < n \tag{3.2}$$

$$\ldots$$

$$[x_{m+k+1}, x_{m+k+2}, \ldots, x_n] \tag{3.3}$$

The cumulative absolute sum is then calculated for each resulting window. The initial step in feature extraction involves data condensation via the Cumulative Absolute Sum function, which is applied to the segmented windows created during

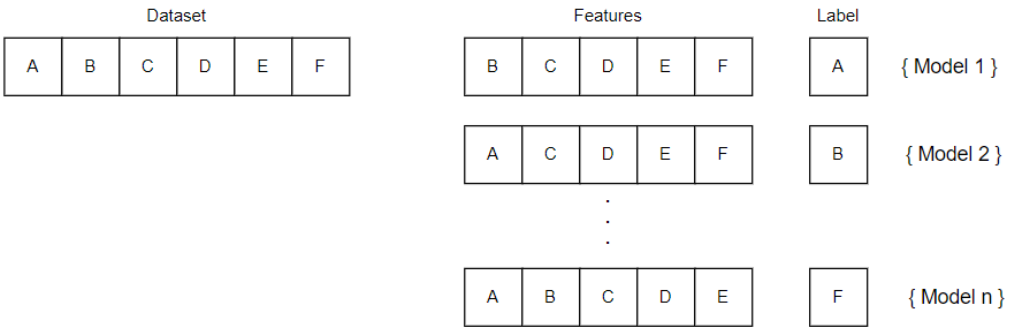


Fig. 3.3 Features selection approach

preprocessing. The Cumulative Absolute Sum function transforms the windowed data segments into single numerical features:

$$\begin{matrix} [x_1, x_2, \ldots, x_m] \\ [x_{m+1}, x_{m+2}, \ldots, x_{m+k}] \\ \vdots \\ [x_{m+k+1}, x_{m+k+2}, \ldots, x_n] \end{matrix} \longrightarrow \begin{matrix} \sum_{i=1}^{m} |x_i| \\ \sum_{i=m+1}^{m+k} |x_i| \\ \vdots \\ \sum_{i=m+k+1}^{n} |x_i| \end{matrix} \tag{3.4}$$

Following this condensation, a multi-threading approach converts one extracted feature from the dataset into the label for a specific ML model, while the remaining features serve as the inputs. This process is repeated for every window/feature, resulting in a number of training sets and corresponding labels equal to the number of windows/predictors.

The AI Unit operation is divided into two macro-stages, the training and the testing process.

- Training Process: The system waits until a predefined amount of data is collected, which is then used to train the ML models.
- Testing Process: Incoming, new data are continuously tested. The same preprocessing and feature extraction methods are applied, and the predicted label for the i-th model is compared with the actual (real) label to compute the prediction error, which is the basis for novelty detection.

For the Training phase, three regression models were evaluated to assess potential differences in prediction behavior: Linear Regressor, Decision Tree Regressor, and

Random Forest Regressor. Generally, the model's prediction error is low when the machine is healthy (at the start of its life) and begins to increase when a failure appears. This expected behavior — low initial error rising as failure progresses. Figure 3.4 concerns model selection based on this behavior, and Figure 3.5 represents the error progression.

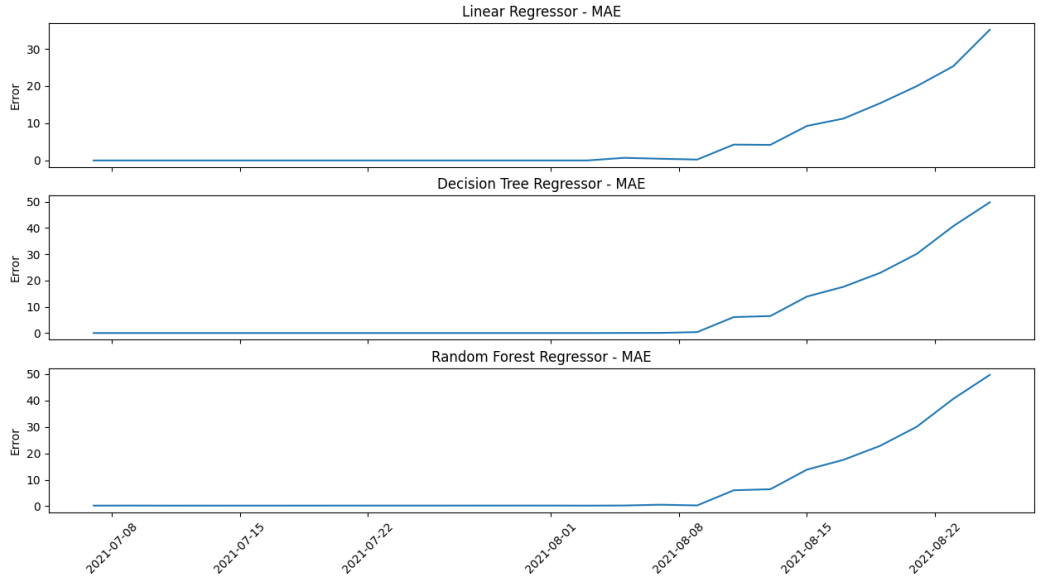


Fig. 3.4 Model Evaluation

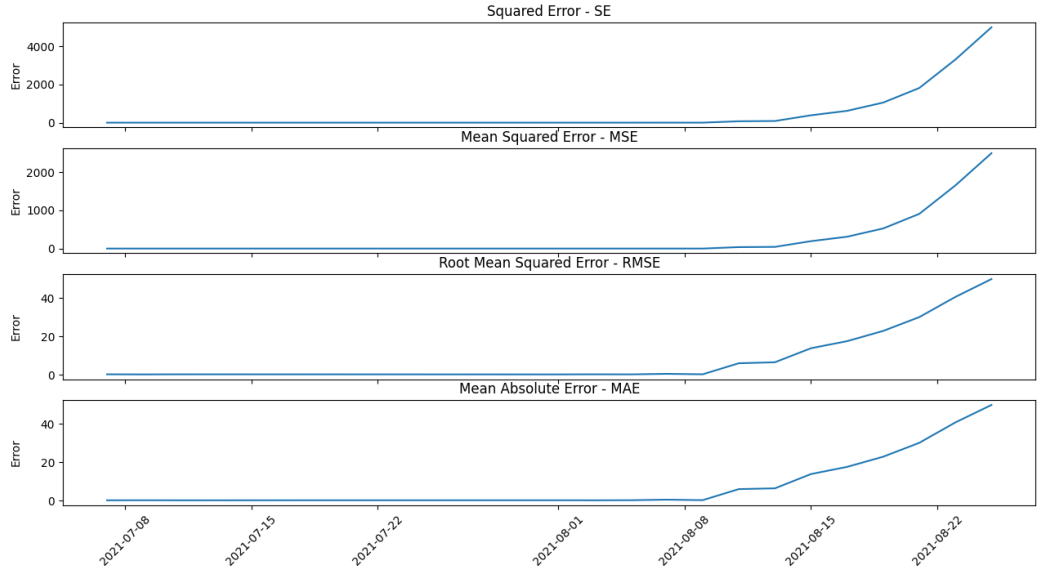


Fig. 3.5 Error evaluation

The three evaluated regression models produced almost identical results. While the Linear Regressor exhibited a slightly lower variance (smaller y-scale range) for the error, the other two models (Decision Tree and Random Forest) showed comparable error ranges.

- Model Performance Trade-offs: Generally, Tree Models offer better accuracy than the simpler Linear Regressor. However, the Linear Regressor is significantly faster during training.
- Decision Tree vs. Random Forest: These two models function similarly, but the Random Forest Regressor (which uses an ensemble of decision trees) offers higher accuracy at the cost of greater computational complexity compared to the single Decision Tree Regressor.

Model selection should be based on the framework's deployment environment:

- For systems with no computational constraints (e.g., a workstation), the Random Forest Regressor is ideal for maximizing prediction accuracy.
- For resource-limited, embedded devices, the Linear Regressor or Decision Tree Regressor should be chosen to reduce the overall computational effort.

Since the framework in this paper is deployed on a workstation, the Random Forest Regressor is chosen as the reference model. During the study, four error metrics were evaluated: Squared Error (SE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

Let y_i be the real value and $\hat{y}_i$ be the predicted value for n samples.

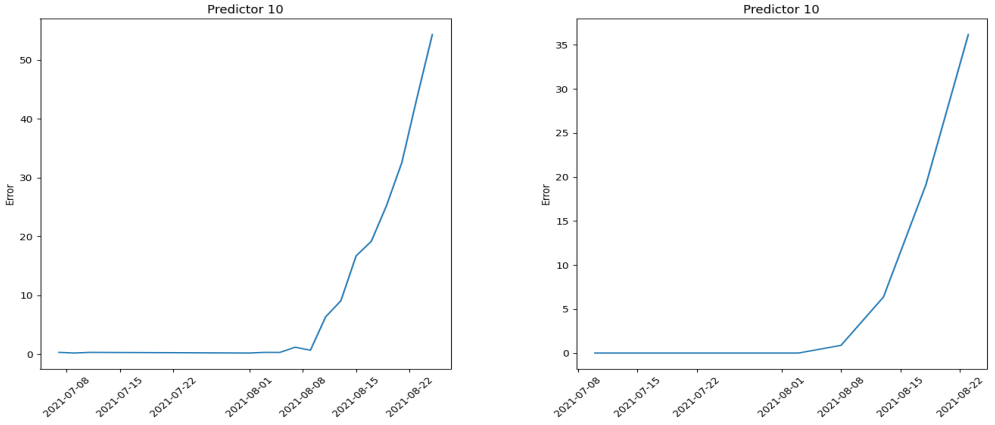
$$\mathrm{SE}(y,\hat{y}) = \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2 \tag{3.5}$$

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2 \tag{3.6}$$

$$\mathrm{RMSE}(y,\hat{y}) = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2} \tag{3.7}$$

$$\mathrm{MAE}(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i| \tag{3.8}$$

Figure 3.5 reveals substantial differences in the error trends. The RMSE and MAE graphs exhibit a faster and more "disrupted" response to prediction behavior than SE and MSE. The RMSE and MAE show very similar graphical behavior due to the similarity in their formulas when the sample size (n) is small. Specifically, the factor $\frac{1}{n}$ in MAE is comparable to the $\sqrt{\frac{1}{n}}$ factor applied to the squared error sum in RMSE when n is small, leading to highly correlated results. The operator has the freedom to choose any combination of these error models for monitoring, though using all four would involve an unnecessary computational cost. For this paper and the subsequent case studies, the Mean Absolute Error (MAE) is chosen for evaluation. Figure 3.6a and 3.6b illustrate how MAE results change based on the sample range n (comparing the last two records versus the last five records).



(a) Error computation on the last two records; (b) Error computation on the last five records;

Fig. 3.6

A two-sample error calculation is utilized in the experimental validation due to its capacity for rapid prediction and prompt anomaly reporting. A core capability of the primary framework is the ability to update or retrain already established models. Since the Novelty Detection mechanism identifies any system change—whether related to a genuine failure or a benign operational variation—the maintainer must assess the situation and implement one of the following adaptive procedures:

- Model Update: If a deviation in the collected data is observed but is not correlated with a physical machine issue, the maintainer can update the framework. This involves incorporating the last parameterized n data points, thereby accepting the incoming data as the new "normal" operating condition. This update procedure substantially reduces the prediction error value. The model update function also serves a beneficial secondary role by continuously collecting consistent historical data, which is essential for future PdM (Predictive Maintenance) implementation.
- Model Retraining: When a component substitution or a definite machine problem occurs, a complete retraining of the models is necessary due to the inevitable shift in the underlying data patterns (e.g., a constant offset or shift error). The maintainer can choose to discard all previous training data and initiate the model training process from the beginning.

The workflow of the AI Unit is cyclical and consists of the following phases:

- Data Fetching Phase: The unit continuously loops, monitoring the local or remote database for new data uploads, which are subsequently fetched for either the Training or Testing phase.
- Training Phase: This phase is triggered when a predefined threshold of datasets has been collected and uploaded to the database. Upon completion, a number of ML models are created, corresponding to the number of features extracted (which is determined by the number of windows defined by the windowing function).
- Testing Phase: Following the initial training, every new dataset is processed by the framework using a number of predictors equal to the number of trained models. Each predictor estimates its corresponding label and computes an error. The onset of failures causes the dataset shape to change, leading to a measurable increase in the prediction error. Employing multiple predictors, corresponding to the number of features, ensures comprehensive coverage of the data distribution for robust error recognition.
- Retraining/Updating Phase: This is the adaptive stage. If model error deviates without a corresponding hardware issue, the maintainer can update the models, normalizing the error value by accepting the new data as normal. The retraining

process is reserved for post-maintenance operations where the past operational history must be neglected to establish a new baseline.

Storage Unit After the elaboration done by the AI Unit, the results obtained are stored again in a database; in this way, it is always possible to see the history of data collected till that moment.

Maintenance Unit The Maintenance Control Unit (MCU) is designed as a software Multi-Agent System (MAS), structured into distinct system and application levels. This section provides a detailed architectural, functional, and procedural analysis of the MCU's implementation.

The Novelty Detection Framework (NDF) continuously measures a system's evolution using various metrics to predict when these measures will indicate significant changes in the system model. The NDF comprises two principal modules: the Artificial Intelligence Unit (AIU) and the MCU. The AIU processes raw data, calculating prediction errors to quantify model accuracy over time. The MCU's central function is to monitor this degradation in model accuracy to detect relevant shifts in system behavior compared to the initial training baseline. Specifically, the MCU performs the following tasks:

- Sensing the errors generated by the AIU Predictors (AIUPs).
- Evaluating current AIUP errors by applying a set of condition-action rules.
- Predicting the time-based degradation of AIUP errors using their most recent results.
- Producing maintenance events to signal necessary changes or updates to the AIUP models.

The AIU consumes raw system data and outputs various error types. The MCU processes these errors to detect changes and calculates the Remaining Time to Retrain (RTR) prediction. The RTR concept is analogous to the RUL (Remaining Useful Life) found in standard PdM environments, but it focuses on a different prediction target:

- RUL predicts the time remaining until a failure occurs.

- RTR predicts the time remaining until the prediction error exceeds a predefined warning threshold, indicating the model requires retraining or updating.

The MCU executes its functions by analyzing each error collected by the AIU separately. Since the results are sourced from distinct AIUPs, the MCU is logically segmented into smaller, independent units (or agents). Each agent is dedicated to computing the necessary result for a fixed AIUP model. This modularity is why the MCU is realized as a software MAS

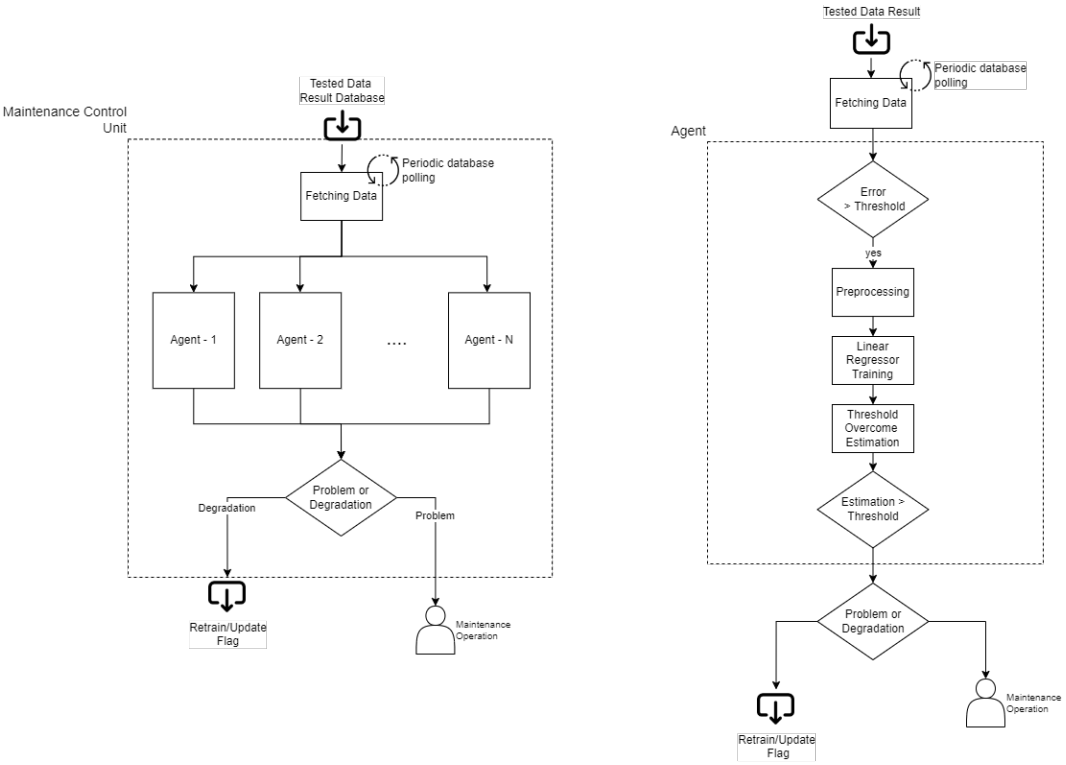


Fig. 3.7 Maintenance Control Unit (left) and agent (right) structures. The Maintenance Control Unit is implemented as a multi-agent system, where each agent is responsible for performing the prediction.

The architecture of the MCU's Multi-Agent System (MAS) is primarily defined by two main operational entities:

- Agent Manager (AM): This entity holds the system control flow and manages the life cycle of individual agents. Its responsibility encompasses maintaining, scheduling, and executing each agent over time.

- Agents: These are the workers that execute their designated job by running their Agent Program (AP) within the system process. Each agent encapsulates its internal intelligence, capabilities, and features necessary to solve its task.

In this architecture, the complexity of the system is largely delegated to the agents, allowing the AM to function as a relatively simpler, control-focused entity. A secondary, overarching component, the Agent System (AS), also appears in the MCU. The AS controls the AM, managing the initialization and configuration of all agents.

The AM is implemented within the MCU system control layer and remains separated from the application process. During the initialization phase, both the AM and all associated agents are loaded. As illustrated in Figure 3.8, the AM main process consists of a stoppable loop composed of three sequential phases:

- Agents Scheduling: The AM identifies and collects all agents that are ready for execution. It then calculates the Wait Time for the Next Scheduling (WTNS).
- Agents Execution: The AM executes the collected ready agents. To enhance the MCU's reliability, the AM enforces a Maximum Wait Time for Execution (MWTE) to prevent deadlock or excessive delays.
- Waiting: The AM then enters in a sleep state for the duration of the calculated WTNS time units before restarting the scheduling phase.

The Model Change Detector Agent (MCDA) is developed as a reactive software agent that operates by perceiving its environment, applying condition-action rules, and executing a predefined action. The MCU application architecture is built from multiple MCDAs, which operate independently and periodically over time.Each MCDA executes its Agent Program (AP) by performing the following sequential operations:

- Sense Environment and Data Acquisition: The agent senses the environment and retrieves the last k prediction errors corresponding to a specific AIU Predictor (AIUP).
- Initial Error Evaluation: A condition-action rule is applied to evaluate the most recent error value. The agent works within an operative range defined by a

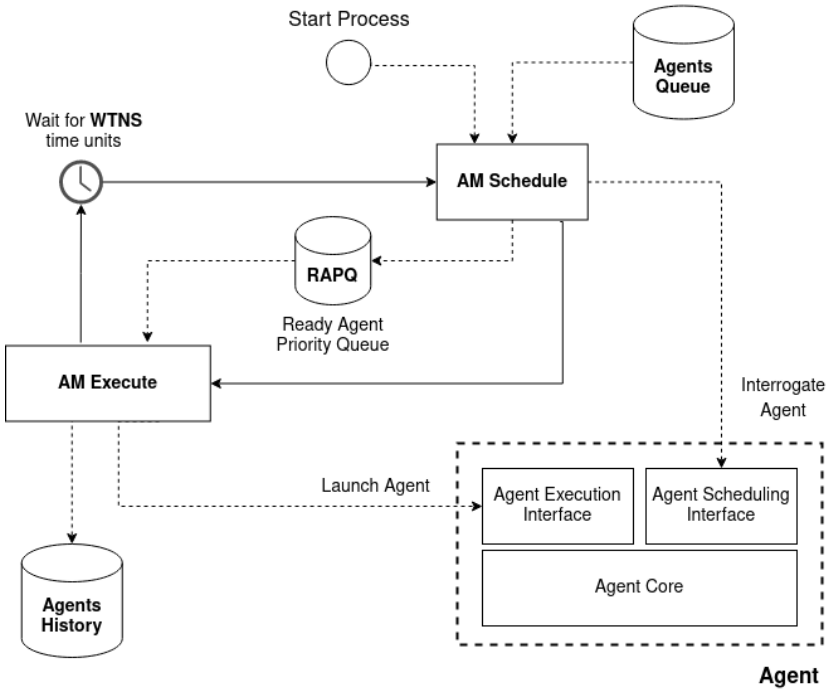


Fig. 3.8 The Agent Manager main process that schedules and executes all ready agents.

lowest bound (which triggers the prediction phase) and a highest bound (the maximum error allowed, used as the target for RTR prediction).

- Data Segmentation and Averaging: Once the last error surpasses the lowest bound, the agent is triggered. It takes the last k errors and divides the set into an "oldest" part and a "newest" part. The agent then computes the average error and average time for both segments. This process yields two distinct data points crucial for the subsequent trend identification phase.

- Trend Degradation Analysis: A Linear Regressor (LR) model is trained using the two computed points. This model is then used to predict the time required to reach the highest error bound.

- Notification Condition Check: A final condition-action rule is applied to the predicted time to degradation. A notification is issued only if the predicted time is within a parameterized notification prevention tolerance time. This prevents notifying model changes whose error threshold will likely not be reached for a long duration, thus reducing unnecessary alerts. If the condition is not met, the agent program ends.

- Notification: If the prediction meets the time condition, a model change notification is executed via an interface actuation function. The notification message specifies the identified predictor that requires attention and the predicted time remaining before the error threshold is exceeded.

In summary, the MCU is implemented as a leader-follower, time-triggered agent system. Both the system and application architectures are entirely developed using multi-threading and Object-Oriented Programming (OOP) principles. Techniques such as inheritance and polymorphism are leveraged to structure the agent hierarchy. This MAS approach ensures high code reusability and extendability, facilitating the easy development of new applications or agents.

3.1.4 Real Case Study

Validation of the framework was performed using a real bearing dataset obtained from the NSF I/UCR Center for Intelligent Maintenance Systems (IMS) [43]. This collection is advantageous because it encompasses the complete component life—from normal behavior to failure—which facilitates the simulation of a genuine operating environment necessary for rigorous framework testing.

Setup and Dataset Description The IMS Bearing setup is configured with four Rexnord ZA-2115 double-row bearings on a rotating shaft, as shown in Figure 3.9. An AC motor is coupled with the shaft by rub belts in the proximity of bearing number 1 to keep the shaft moving at the required speed. Datasets are withdrawn at a constant rotational speed for the AC motor, corresponding to 2000 RPM. A radial load of 27,000 N is applied through a spring mechanism onto bearings 2 and 3. The four bearings are force lubricated.

Bearings The dataset is collected using High Sensitivity Quartz ICP accelerometers sampling data at 20 kHz. The dataset is composed of files in which every file corresponds to 20,480 measures sampled every 10 min (except for he first 43 files taken every 5 min). After 100 million revolutions, an inner race defect occurred in bearing 3.

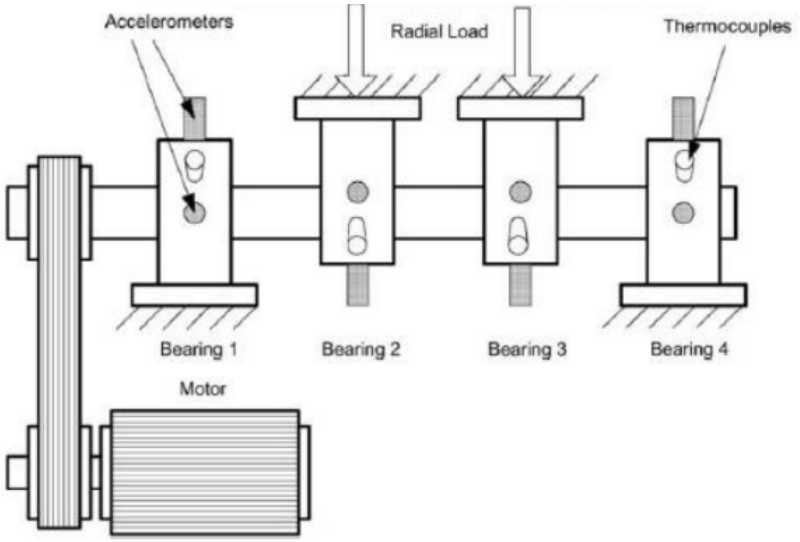


Fig. 3.9 IMS Setup

Framework Application Testing on the IMS dataset focused on validating the framework's ability to recognize the inner race failure of bearing 3. The data infrastructure relies on MongoDB, a non-relational database, for both the Data Source and Storage Units. The specific parameters defining the AIU configuration used for this evaluation are presented in Table 3.2.

Description	**Q.ty**
Train Elements	200
Test Elements	2
Windows	20

Table 3.2 Parameters of the AIU

The chosen parameters were selected to establish an effective trade-off between system responsiveness and prediction accuracy. With this configuration, the AIU models were fully trained and ready for monitoring after 30 hours. Subsequently, the framework initiated the continuous error evaluation process for each model to monitor the status of the bearings. The specific settings employed for the Maintenance Control Unit (MCU) configuration during this evaluation are summarized in Table 3.3.

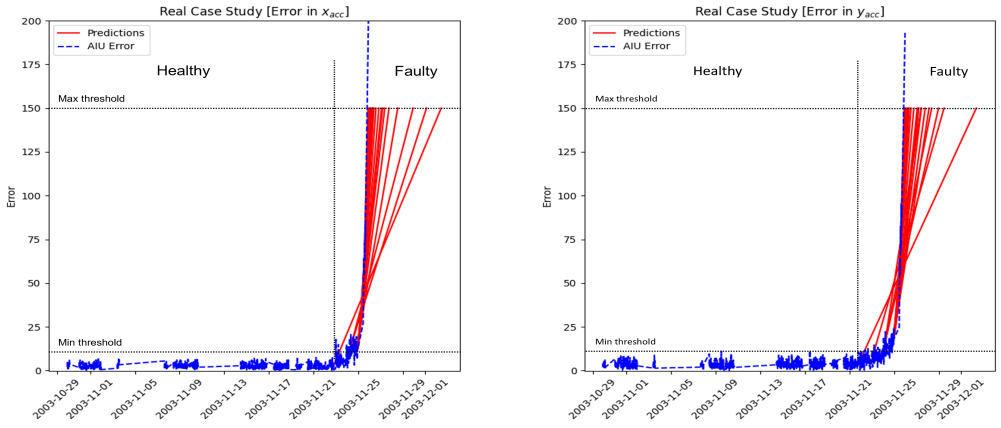
The results obtained from applying the framework to the bearing dataset are presented in Figures 3.10a and 3.10b. In these figures, the blue dashed line illustrates the degradation of the x and y prediction errors computed from the acceleration data

Description	**Q.ty**
Average	16
Lowest Bound	10
Highest Bound	150

Table 3.3 MCU Configuration

of bearing 3, while the red lines represent the predictions generated by the MCU. As observed in the graphs, the initial prediction error is low and increases accordingly as the bearing failure develops. The system successfully recognized the relevant change approximately half a day to one full day before the final failure. While the detection trigger could be lowered to alert the framework earlier, doing so would proportionally increase the number of false positive alerts. The predictions, represented by the red lines, show an improving fit over time, with their monotonicity increasing as the system approaches complete breakdown. The architectural advantage of having multiple predictors (equal to the number of features) allows for compensation against potential false positives. The framework can be configured to issue a warning only if more than k predictors simultaneously detect a change. The long straight lines visible in the graphs indicate periods of missing data. Using a line plot instead of a scatter plot is beneficial in this context as it helps identify potential shifts in the data pattern. Such shifts may be correlated with physical events, such as a maintenance operation or component substitution performed on the machine.

Model Update Functionality The behavior of the framework's update function is demonstrated in Figure 3.11. The effect of this function is clearly noted by comparing Figure 3.11 with Figure 3.10b around November 21st. The training set was updated with 200 additional datasets, increasing the total size of the collected dataset to 400. Following this model update procedure, the error computed by the AIU successfully decreased, reaching a value close to zero. The maintainer can deploy this procedure when the Novelty Detection identifies a change that is not related to a physical problem on the machine or component, effectively resetting the baseline for normal operation.



(a) Predictions with BPFI fault [x_{acc}] (b) Predictions with BPFI fault [y_{acc}]

Fig. 3.10

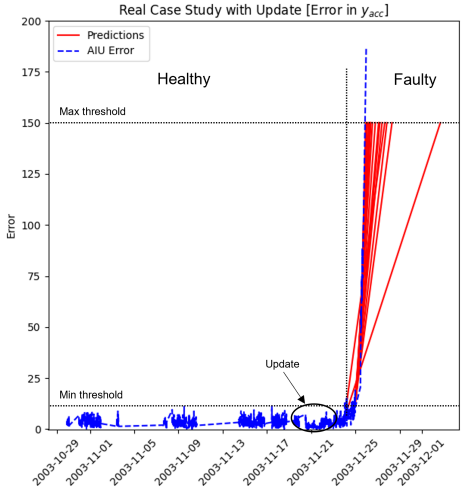


Fig. 3.11 Predictions with BPFI fault [y_{acc}] and update.

3.1.5 Discussion

In order to have an idea about the framework's performances and better see the scalability property, two tables are compiled with Train, Test, and Preprocess time with different configurations. This analysis is performed for both the AIU and the MCU. Tables 3.4 and 3.5 report the results of this analysis.

The results presented in the preceding tables confirm the system's high efficiency. The training stage requires a minimal amount of time, a benefit derived from the use of computationally lightweight ML models. This rapid training capability is vital for

Feat.	***W***	s_{train}	s_{test}	t_{train}	$t_{preprocess}$		t_{test}	
				[s]	**min [s]**	**max [s]**	**min [s]**	**max [s]**
120	20	5	2	0.9880	0.9296	1.4058	0.4927	0.6483
120	20	5	5	0.9857	1.8990	2.2966	0.7211	0.5164
120	20	5	10	1.0237	0.3692	0.6698	0.4664	0.7705
120	20	10	2	1.04	1.8848	2.2882	0.5146	0.5387
120	20	10	10	1.0764	1.8849	2.2882	0.4696	0.4882
120	20	20	2	1.2819	0.3692	0.4345	0.4689	0.4967
120	20	50	2	1.3426	0.3681	0.4476	0.4680	0.5377
120	20	70	2	1.3275	0.9265	1.0170	0.4974	0.5057
120	20	70	5	2.6040	0.9265	1.0170	1.3399	1.3878
300	50	70	5	4.4466	0.9281	1.0314	1.3418	1.4066
600	100	70	2	12.6561	0.3772	0.4523	2.5674	2.7112

Table 3.4 AIU time performance indicator W, s_{train}, s_{test} and [min, max] time statistics for *Preprocess*, *Train*, *Test*.

$t_{execution}$		t_{train}		$t_{predict}$	
μ **[ms]**	σ **[ms]**	μ **[ms]**	σ **[ms]**	μ **[ms]**	σ **[ms]**
16.6287	5.0196	0.0106	0.1027	0.0067	0.0813

Table 3.5 MCU time performance indicator. It shows the statistics μ and σ calculated over all MCDAs *Execution*, *Train*, *Predict* operations.

real-time maintainability, allowing the framework to be quickly updated or retrained when new sensors are incorporated into the machine. The time required for core computations consistently remains in the order of seconds. The Artificial Intelligence Unit (AIU) demands more time than the Maintenance Control Unit (MCU) due to its higher computational effort, with the initial training process representing the hardest computational task. All performance metrics are critically dependent on the number of windows (w) elaborated; a higher w directly correlates with increased time requirements. The MCU's performance, detailed in Table 3.5, refers to the mean and variance of the agent's execution time on a single window. Since each agent requires only a small amount of time compared to the AIU, and the MCU processes agents sequentially via a time-scheduling approach, the number of windows significantly

influences the total MCU time. The total MCU time (T_m) is calculated by multiplying the execution time per agent (T_e) by the total number of windows (w):

$$T_m = T_e \cdot w \tag{3.9}$$

The framework's flexibility and adaptability are proven across diverse domain examples presented in this paper.

- Bearing Degradation: the framework accurately identified when the system deviated excessively, successfully alerting the maintainer prior to complete breakdown.
- Fault Detection Flexibility.
- Domain Adaptability.

The framework offers the following significant advantages:

- No Prior Experience or Data History Required: This framework can detect novelties (errors, faults, problems, and environmental changes) by comparing current observations against the trained baseline. This capability is also instrumental in collecting historical datasets and generating maintenance warnings, which are essential for future definitive Predictive Maintenance (PdM) applications.
- Flexibility and Scalability: The framework is highly flexible and scalable. A maintainer can integrate new sensors at any time, updating or retraining the framework to establish a new operational baseline while still recording all past data for future PdM use.
- Low Computational Effort Required: By utilizing straightforward ML models to estimate the divergence between trained and current data, the framework maintains a low computational effort. This feature significantly facilitates future implementation on embedded devices.

3.1.6 Future research

A primary future enhancement involves optimizing computational load by neglecting useless predictors. By identifying and removing predictors that do not contribute significantly to the overall prediction accuracy, the following benefits can be achieved:

- Lightening the MCU Payload: Reducing the data and model size managed by the Maintenance Control Unit.
- Decreasing Prediction Time: Directly reducing the time required for continuous monitoring.
- Improving Real-Time Efficiency: Increasing the responsiveness of the entire framework.

Future work will concentrate on broadening the framework's applicability in complex and dynamic environments:

- Multi-Domain Dataset Support: Adapting the framework to handle datasets where features are sampled at different frequencies (e.g., high-frequency vibration data from an accelerometer and low-frequency temperature data from a thermometer).
- General Motion Adaptation: Enhancing the framework to monitor systems undergoing random or general motion, rather than being restricted solely to previously trained, identical motion profiles.

A significant future step involves moving the framework from a workstation environment to real embedded devices. This requires:

- Framework Optimization: Tailoring the existing code and architecture for resource-constrained hardware.
- Tiny ML Evaluation: Assessing the performance and viability of using Tiny ML models within the framework to maintain low computational requirements.
- Intelligent Edge Device Creation: Ultimately aiming to create an intelligent edge device that can be mounted directly onto the analyzed component, enabling localized, real-time monitoring.

Furthermore, a comparative analysis will be conducted to more clearly assess its differences from baseline algorithms in terms of speed and accuracy.

3.2 Unsupervised Novelty Detection Methods Benchmarking with Wavelet Decomposition

3.2.1 Introduction

In today's data-centric landscape, the ability to detect unusual occurrences is increasingly important. Novelty detection—namely, the task of identifying previously unseen or unfamiliar data that significantly diverges from expected behaviour—is fundamental across many sectors because it helps uncover irregular events that may signal critical situations such as equipment malfunctions, fraudulent activity, or emerging phenomena.

For instance, in manufacturing settings, novelty detection can reveal defects arising in a production line [2], while in cybersecurity it can highlight atypical behavioural patterns that may point to an ongoing attack [44]. In the financial domain, anomaly detection supports fraud prevention by enabling institutions to flag suspicious transactions before they lead to major losses. Overall, the ability to recognize novel conditions strengthens decision-making, enhances safety, and mitigates risks in a wide range of applications.

Classical novelty-detection techniques often draw on statistical strategies [45, 46]. These approaches typically rely on predefined boundaries or explicit models of normal behaviour, making it difficult for them to adapt to shifting patterns or previously unseen anomalies.

Advances in Artificial Intelligence have greatly transformed how novelty detection is approached. Machine Learning methods can learn directly from data, uncovering relationships that would be hard to encode by hand. Within AI-based strategies, supervised learning models require labeled datasets to train predictors capable of identifying anomalies; prominent examples include neural networks and support vector machines trained on historical samples [47, 48]. Semi-supervised approaches, which combine labeled with unlabeled data, offer a compromise by

using limited annotated data to guide the learning process while exploiting unlabeled samples to enhance robustness and accuracy [49, 50].

Unsupervised learning approaches, which operate without labeled data, are particularly useful when annotation is costly or when anomalies are too rare to label reliably [49]. These techniques include clustering methods such as K-means, autoencoders, and density-based algorithms [51–56]. Such models can reveal the intrinsic structure of the data and pinpoint deviations from it, making unsupervised approaches highly adaptable and effective in dynamic environments.

To the best of our knowledge, existing research does not provide a thorough comparison of Unsupervised Machine Learning (UML) techniques capable of producing a continuous degradation metric, particularly when paired with different preprocessing pipelines. This benchmark would be highly beneficial to the novelty-detection community, offering a valuable reference for both researchers and practitioners.

In this work, several unsupervised AI algorithms are evaluated for novelty detection using a dataset collected in a controlled laboratory environment. Vibrations generated by a shaker at specific frequencies were recorded while novel conditions were artificially introduced by varying the input waveform. Our goal is to estimate a continuous metric rather than a binary classification outcome, as this enables a more nuanced assessment of how well each method can capture emerging patterns in complex data. A set of statistical descriptors and wavelet-based coefficients is computed from the raw signal to serve as inputs for the UML models. Additionally, a comprehensive suite of performance metrics to compare the various framework configurations is established. Finally, computational efficiency is assessed by measuring each configuration's inference time. The subsequent sections describe our methodology, experiments, and findings in detail.

3.2.2 Related Works

In this section, a summary with the key prior research on novelty detection, with particular emphasis on unsupervised methodologies.

In [54, 55], the K-means algorithm is employed to categorize bearing degradation conditions. In [54], the IMS bearing dataset [57] is clustered using both Traditional Statistical Features (TSF) and Mel-frequency Cepstral Coefficients. The resulting labeled sequences are then used to train a Convolutional Neural Network (CNN)

directly on the raw vibration signals, eliminating the need for additional feature extraction. Similarly, [55] uses TSF and Shannon entropy to cluster data from the IMS and Case Western Reserve University (CWRU) [58] datasets. The authors also transform the time series into two-dimensional image representations and adopt a CNN (AlexNet [59]) to classify levels of degradation.

The work in [52] introduces a procedure for assessing bearing health and tests it on the IMS dataset. Time-domain features are first extracted from the vibration signals, and a cross-correlation filter is applied to eliminate redundant information. K-means is then utilized to select the most informative features through an optimization scheme aimed at producing compact and well-separated clusters. A Self-Organizing Map is subsequently used to compute a health indicator for the bearing.

An enhanced initialization strategy for K-means is proposed in [60]. The method integrates Ant Colony Optimization with K-means and is evaluated using a three-level Wavelet Packet Decomposition (WPD) applied to an expanded CWRU dataset, created via a sliding-window technique to demonstrate scalability to large data collections.

In [53], the authors present a subset-based deep autoencoder designed to automatically learn discriminative feature representations. The model is validated on vibration data from CWRU, IMS, and Self-Priming Centrifugal Pump (SPCP) [61] bearings.

A detailed investigation based on the IMS dataset is given in [56]. The authors exploit prior knowledge of bearing fault frequencies to assign labels to the measurements. Initially, TSF and Redundant Second-Generation Wavelet Packet Transform coefficients are extracted, after which Principal Component Analysis (PCA) is used to reduce dimensionality. Subsequently, K-means, Support Vector Machines (SVM), and agglomerative clustering techniques are applied to detect anomalies and identify failure modes.

Another noteworthy contribution is the data-stream classification technique introduced in [51]. Unlike one-class strategies, this approach supports multiple “normal” classes and handles the emergence of new classes over time by integrating coherent clusters of previously unseen samples into the training set.

[2] describes a machine learning framework designed for real-time anomaly detection in sensor data, providing support for companies developing predictive

maintenance datasets. The study outlines an optimized software architecture and demonstrates its effectiveness using both a digital twin and industrial case studies.

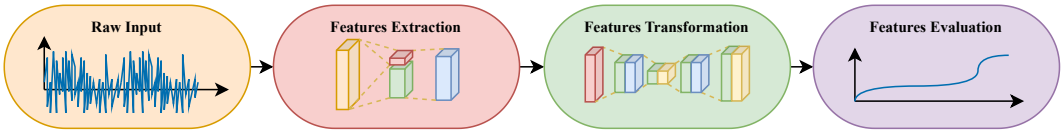


Fig. 3.12 The architecture of the proposed framework is organized into three core components. First, the feature extraction module applies Wavelet Packet Decomposition (WPD) to obtain wavelet coefficients, which are then combined with statistical descriptors to form its output. Next, the feature transformation module uses either an Autoencoder or Principal Component Analysis (PCA) to map these features into a new representation, allowing the system to better capture variations in behaviour when computing the novelty metric. The final component, the feature evaluation module, employs six different unsupervised machine learning algorithms to generate the novelty metric from the transformed feature set.

[62] introduces a computer vision–based approach for detecting anomalies in mechanical equipment. Compared with the previously mentioned techniques, this method offers the benefit of monitoring vibrations at multiple locations without requiring direct contact with the components under inspection.

Another widely used model in novelty detection is the Local Outlier Factor (LOF) [63], which produces a continuous anomaly score determined by the location of a point relative to the density of its neighbours. This idea is further developed in [64], where the clustered nature of the dataset is incorporated into the formulation, resulting in the Cluster-Based LOF (CBLOF) metric.

A related approach for quantifying normality using a distance-based measure is described in [65]. The authors define a novelty score as the distance of a sample to its nearest cluster, scaled by the standard deviation of distances among the known samples in that cluster. This method has been evaluated on vibration data obtained from a jet engine.

A further distance-driven technique is presented in [66], where each data instance is categorized as normal, an extension, or unknown according to the radius of the nearest cluster and the point's relative position. When a sample lies just beyond the decision boundary but within an acceptable margin, it is labeled as an extension and leads to an update of the learned model. If the point falls beyond this margin, it is deemed unknown; such samples are stored in a buffer and later used to revise the model when new classes appear in the data.

3.2.3 Methodology

The proposed novelty detection framework operates in a fully unsupervised manner. The training data represent the system's nominal behaviour and contain no labels. In contrast to classification-based approaches, the framework has no prior knowledge of the system's operating modes. After training, the model evaluates incoming data to identify deviations from expected behaviour and thus detect anomalies. An overview of the architecture is presented in Fig. 3.12.

The inputs to the system are raw time series recorded by an accelerometer. These signals are processed to extract a set of informative features, which are subsequently mapped into a latent representation that may be of lower or higher dimensionality. The transformed features are then analysed by multiple unsupervised machine learning models, each producing a novelty metric (NM), which constitutes the framework's output. The main advantage of this approach is its ability to quantify the degree of deviation rather than simply assigning a binary label (0/1) to distinguish normal from abnormal conditions—a limitation common in traditional anomaly detection methods. Each component of the framework is described in detail in the next section.

The features extracted from the time series are a combination of statistical and WPD coefficients. The statistical features extracted for each signal are: mean, root-mean-square (RMS), peak-to-peak (P2P), standard deviation (STD), skewness and Kurtosis.

Wavelet Packet Decomposition

The WPD is a tree-based decomposition method able to extract the frequency content of a signal. In this work, a Daubechies wavelet is used. The decomposition method divides the input data $x \in \mathbb{R}^N$ into two subsets: a set with high-frequency and the other with low-frequency content, each containing $N/2$ elements. This process can be applied an arbitrary number of times (up to $\log_2(N)$) for each subset created. The process is repeated recursively until the desired number of levels L is reached. Hence, the final output is a signal decomposed into 2^L vectors in which each vector has $N/2^L$ elements. Finally, each vector generated by the decomposition is transformed into a single value using the l_2 norm. This value assumes the role of a feature. The

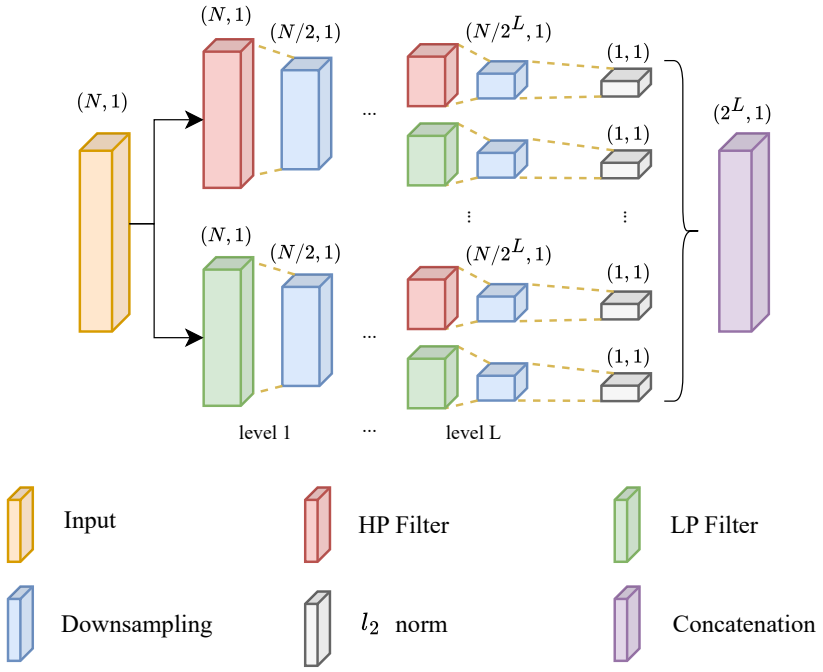


Fig. 3.13 Wavelet Packet Decomposition features extraction architecture. At each level of decomposition, the input signal is split into two sub-band signals, each with a dimension of $N/2$. This process is repeated through L levels, resulting in 2^L sub-band signals. For each sub-band signal, the l_2 norm is computed, condensing each array into a single value. These values are then aggregated into a final array of dimension 2^L, forming the complete WPD feature array that encapsulates the characteristics of the original time series data.

2^L computed norms form the WPD feature array. The structure of the decomposition is shown in Fig.3.13.

In the end, the WPD feature vector is concatenated with the six statistical features to form the complete features array with dimension $(2^L + 6, 1)$ representing the final output of the features extraction block.

After the feature extraction process, the data are normalized, along the whole training dataset, removing the mean and scaling to unit variance in order to use them for the following steps.

At this stage, the extracted features can be directly used to train the models. However, additional processing can be implemented to analyze the different behaviours of the unsupervised model. Specifically, two different Autoencoders and PCA are tested to observe the change in the novelty metric, as explained in the next section.

In this section, the feature transformation module of the framework is described, which incorporates three alternative algorithms: an undercomplete autoencoder, an overcomplete autoencoder, and PCA.

Autoencoder

The first transformation strategy is based on an autoencoder (AE), a generative neural network composed of two main components: an encoder that compresses or expands the dimensionality of the input, and a decoder that reconstructs the original representation. In this architecture, the input and output layers must have matching dimensions, while the intermediate layers can vary in size and depth.

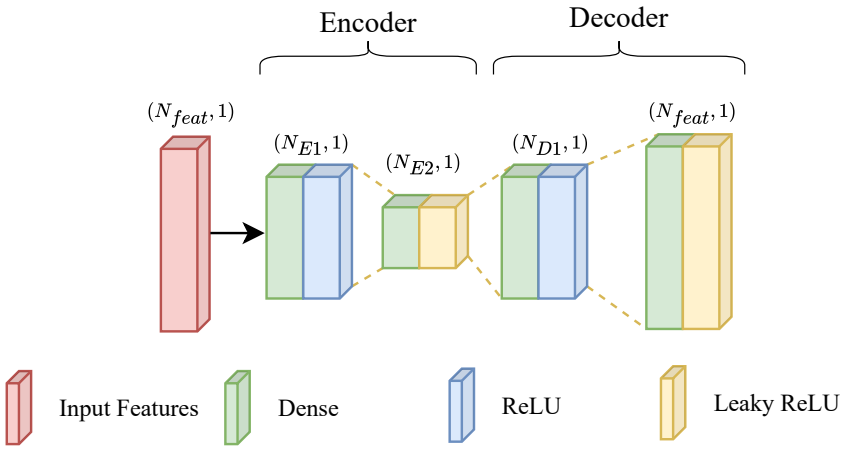


Fig. 3.14 Architecture of the proposed undercomplete autoencoder for feature transformation. The model uses fully connected layers with dimensions $N_{E2} < N_{E1} < N_{feat}$ and $N_{E2} < N_{D1} < N_{feat}$. The input consists of the preprocessed statistical and wavelet-based features.

Undercomplete Autoencoder The structure of the undercomplete autoencoder is provided in Figure 3.14. The input dimension is $n = N_{feat} = 2^L + 6$, as described in Section 3.2.3. The encoder includes layers of size N_{E1} and N_{E2}, where the latent dimension is $N_{E2} = p$ and satisfies $N_{E2} < N_{E1} < N_{feat}$. The decoder mirrors this structure, expanding the latent vector back to N_{feat} using layers N_{D1} and N_{feat}, with $N_{E2} < N_{D1} < N_{feat}$. The first layer of both encoder and decoder uses a ReLU activation, while the second layer uses LeakyReLU to improve stability and avoid excessive zero activations. The training objective is the Mean Squared Error (MSE).

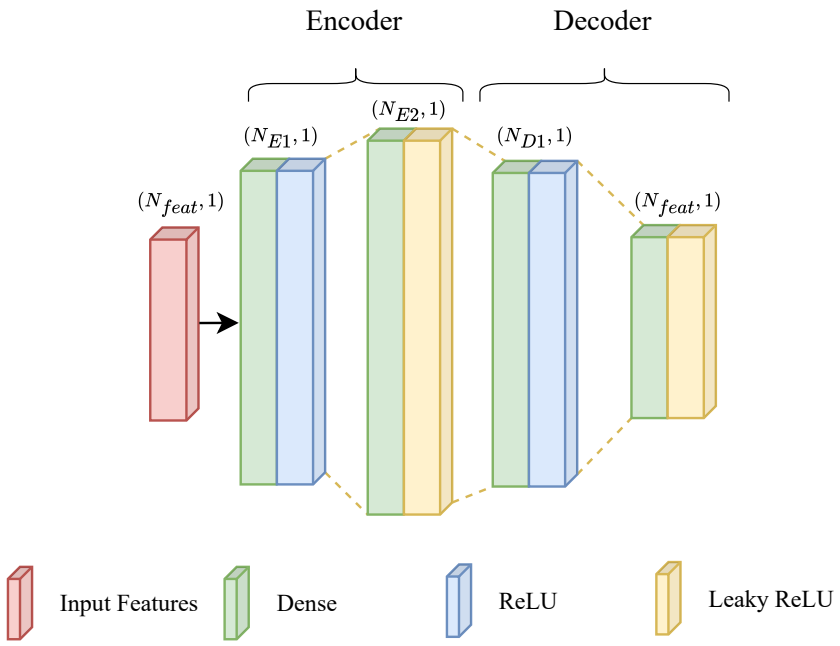


Fig. 3.15 Architecture of the proposed overcomplete autoencoder. The model consists of fully connected layers satisfying $N_{feat} < N_{E1} < N_{E2}$ and $N_{feat} < N_{D1} < N_{E2}$.

Overcomplete Autoencoder Figure 3.15 illustrates the configuration of the overcomplete autoencoder. Unlike the undercomplete variant, this model increases the dimensionality of the data within the encoder: $N_{feat} < N_{E1} < N_{E2}$. The decoder then projects from N_{E2} back to the original size using layers that satisfy $N_{feat} < N_{D1} < N_{E2}$. As with the undercomplete architecture, ReLU and LeakyReLU activations are applied in the first and second layers respectively, and MSE is used as the reconstruction loss.

Principal Component Analysis

Principal Component Analysis (PCA) is employed to project the extracted features into a reduced-dimensional subspace while retaining as much of the original data variance as possible. The dimensionality reduction is controlled through the proportion of variance that must be preserved.

Let $X \in \mathbb{R}^n$ denote the input vector, where $n = N_{feat}$. PCA computes a latent representation $h = PCA(X)$, where $h \in \mathbb{R}^{N_{PCA}}$ and $N_{PCA} < N_{feat}$. Formally, the transformation is expressed as $PCA : \mathbb{R}^{N_{feat}} \rightarrow \mathbb{R}^{N_{PCA}}$.

Each of the three transformation strategies produces a latent feature set of different dimensionality, which is subsequently fed to the unsupervised learning models that compute the novelty metric described in the following section.

The six unsupervised learning models that can be combined with any of the feature transformation techniques are already described in the Section 2.7 of the theoretical analysis. Each model receives as input the transformed features—either the latent representations produced by the autoencoders (specifically, the output of layer E_2) or the features obtained after the PCA projection. The output of this module is the novelty metric (NM), which quantifies how anomalous a sample is.

3.2.4 Experiments

A pivotal point of our research is the realization of a dataset to test the Novelty Detection algorithms in a real environment. For this purpose, a shaker (mod. n° K2007E01) and an accelerometer (mod. n° STM32L4R9, an evaluation board of STMicroelectronics equipped with an accelerometer sensor) were used to collect a vibration dataset. The evaluation board is mounted on the shaker to gather the dataset with reduced noise and uncertainties.

The shaker is controlled by a PC, reproducing an audio file. Two audio files were generated: the first reproduces a signal $v_1(t)$ and the second one the signal $v_2(t)$, defined as:

$$v_1(t) = \sum_{i=1}^{9} A_i \cdot \sin(2\pi \cdot \alpha_i \cdot t)) \tag{3.10}$$

$$A = \{0.5, 0.5, 0.5, 1, 1, 1, 0.5, 0.5, 0.5\}$$
$$\alpha = \{50, 100, 150, 230, 300, 440, 460, 530, 600\}$$

$$v_2(t) = \sum_{i=1}^{9} B_i \cdot \sin(2\pi \cdot \beta_i \cdot t)) \tag{3.11}$$

$$B = [0.5, 0.5, 0.5, 1, 0.2, 1, 0.5, 0.5, 2]$$
$$\beta = [50, 100, 150, 230, 300, 440, 460, 530, 600]$$

Set name	Signal Type	P2P [V]
Set 1	$v_1(t)$	0.25
Set 2	$v_1(t)$	0.50
Set 3	$v_1(t)$	0.75
Set 4	$v_1(t)$	1.00
Set 5	$v_1(t)$	1.25
Set 6	$v_2(t)$	0.50
Set 7	$v_2(t)$	0.75
Set 8	$v_2(t)$	1.00

Table 3.6 Experimental setup for data collection.

where A, B are the amplitudes and α, β are the frequencies of the harmonics. The volume setting determines the $P2P$ amplitude of the physical signal generated by the PC.

Firstly, the signal $v_1(t)$ was fed to the shaker with five different volume settings, producing Sets 1–5. After each volume change, the new $P2P$ value was measured with an oscilloscope. The same procedure was then repeated using $v_2(t)$, yielding Sets 6–8. Each of the eight datasets consists of a 206-second vibration recording, later segmented into 1-second windows to generate 206 samples per set. Table 3.6 summarizes the experimental setup.

Since the highest harmonic frequency is 600 Hz, the sampling frequency of the evaluation board was set to $f_s = 1666$ Hz to satisfy the Nyquist–Shannon sampling theorem.

The procedure described in Section 3.2.3 is applied to the experimental data collected. The depth level L of the WPD decomposition tree is set to 6, resulting in a WPD feature array composed of 64 coefficients. Additionally, the 6 statistical features are computed, leading to a final feature vector of 70 elements.

The performances of each architecture are evaluated to establish a comprehensive benchmark comparison. Four different performance metrics are computed for each framework.

Variance

The variance of the novelty metric is evaluated using the portion of nominal data not employed for training. A robust model should produce a stable novelty metric when the input data represent the same nominal operating conditions. High variance indicates that the model is either overly sensitive to noise or unable to generalize normal behaviour effectively. Formally, the variance is defined as:

$$Variance = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{NM}(i) - \overline{\mathrm{NM}}\right)^2 \tag{3.12}$$

where $\mathrm{NM}(i)$ is the novelty metric of the i-th nominal sample, NM is the mean novelty metric computed over all nominal samples, and n is the number of nominal samples.

Reactivity

This metric measures how effectively a model distinguishes nominal from anomalous samples under identical conditions. The reactivity of the model is defined as the difference between the mean nominal novelty metric and the mean novelty metric of the most anomalous samples. A higher reactivity indicates a stronger capability to separate nominal from novel behaviour. Formally:

$$\mathrm{Reactivity} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{NM}(i) - \frac{1}{m}\sum_{j=1}^{m}\mathrm{NM}(j) \tag{3.13}$$

where $\mathrm{NM}(i)$ is the novelty metric of the i-th nominal sample, and $\mathrm{NM}(j)$ is the novelty metric of the j-th novel sample.

Inference Time

The inference time quantifies the average time required by the UML model to evaluate a single sample and compute the corresponding novelty metric. This value is obtained by averaging the evaluation time over multiple samples. All inference-time measurements are performed using an Intel CPU i9-12900K.

Feature Percentage

The Feature Percentage (FP) indicates the proportion of the original feature dimensionality preserved after feature transformation. A FP of 100% means that no dimensionality reduction is applied. Lower FP values are desirable, as they reduce computational cost and mitigate issues associated with high-dimensional feature spaces (i.e., the curse of dimensionality).

The algorithms are also optimized to determine the best-performing configuration for each UML model. For the autoencoder-based methods, the optimization process varies the number of neurons in the first encoder layer N_{E1} (which is mirrored in the decoder, i.e., $N_{E1} = N_{D1}$), the number of neurons in the latent space N_{E2}, the learning rate, and the batch size. For PCA, the only hyperparameter considered is the number of latent features n_f.

The variance of the novelty metric is used as the objective function because, during optimization, it is the only metric that can be computed using solely nominal data—without requiring knowledge of future system behaviour (e.g., reactivity). Minimizing this variance is essential, as it quantifies how consistently the algorithm models the nominal operating condition of the system. A lower variance indicates reduced sensitivity to noise and environmental perturbations and therefore better nominal-behaviour modelling.

The optimization procedure is based on a Gaussian process Bayesian algorithm. At each iteration, it proposes new hyperparameter values in order to minimize the objective function and progressively improve the model's robustness and detection capability.

The first 100 samples from Set 1 in Table 3.6 are used for training, while the remaining 106 samples are used to compute the variance metric. Formally:

$$J = min(\sigma_{100}NM)$$

$$\sigma_{100}(NM) \rightarrow N_{E1,D1}, N_{E2}, lr, bs, n_f$$

where J is the objective function to be minimized, lr is the learning rate, bs is the batch size, and n_f is the number of PCA latent features. The optimization algorithm requires predefined search ranges for each hyperparameter. These ranges are reported in Table 3.7.

Transf. Method	N_{E1}, N_{D1}	N_{E2}	lr	bs	n_f
AEA	$50-65$	$10-45$	$0.01-0.1$	$32, 64$	-
AER	$75-80$	$85-100$	$0.01-0.1$	$32, 64$	-
PCA	-	-	-	-	$2-70$

Table 3.7 Hyperparameters ranges chosen for the models' optimization algorithm

Model	N_{E1}, N_{D1}		N_{E2}		lr		bs		n_f
	AEA	AER	AEA	AER	AEA	AER	AEA	AER	PCA
KMeans	80	61	85	32	0.08	0.03	64	64	3
DBSCAN	75	56	85	10	0.20	0.08	32	64	2
GMM	79	61	85	10	0.08	0.01	32	64	2
nuSVM	80	65	93	10	0.09	0.10	32	64	2
IForest	79	50	85	45	0.02	0.03	64	64	70
LOF	79	50	88	10	0.03	0.01	32	32	3

Table 3.8 Optimized hyperparameters for each UML model. AEA = overcomplete autoencoder, AER = undercomplete autoencoder.

The optimal hyperparameters found through the Bayesian optimization process are shown in Table 3.8. AEA denotes the overcomplete autoencoder, while AER denotes the undercomplete autoencoder.

Performance metrics are computed for all combinations of UML models and feature transformation methods.

The first 100 samples of Set 1 are used to train the UML models. For each UML model, four scenarios are benchmarked: (i) the baseline using the features extracted directly from the feature extraction block (OF, "Original Features"), and (ii–iv) the three transformation methods—undercomplete AE, overcomplete AE, and PCA.

The remaining 106 samples of Set 1 are used to compute the variance and the mean of the nominal signal (the "mean–nominal signal"). Set 5 is used as the reference novelty condition to compute the "mean–novelty signal" and thus obtain the reactivity. Set 5 is chosen because it corresponds to the most dissimilar vibration among all available sets.

For each model and transformation method, the novelty metric (NM) is computed and normalized to a common range such that $\min(\text{NM}) = 0$ and $\max(\text{NM}) = 1$, ensuring fair comparison across methods.

UML model	Transformation	n_f	FP [%] ⇓	Variance [10^{-3}] ⇓	Mean		Reactivity ⇑	Inference Time [μs] ⇓
					Nominal signal ⇓	*Novelty Signal* ⇑		
KMeans	OF	70	100.00	0.2680	0.0063	**0.9686**	**0.9623**	241.4
	AER	32	45.71	0.1098	0.0072	0.9352	0.9280	210.8
	AEA	85	121.43	0.1615	**0.0013**	0.9612	0.9599	212.5
	PCA	**3**	**4.29**	**0.0780**	0.0190	0.8860	0.8669	**193.1**
DBSCAN	OF	70	100.00	0.2994	0.0119	**0.9835**	**0.9716**	11.3
	AER	10	14.29	0.0957	**0.0035**	0.9243	0.9208	7.2
	AEA	85	121.43	0.0776	0.0095	0.9234	0.9139	8.0
	PCA	**2***	**2.86***	**0.0111**	0.0043	0.9093	0.9050	**6.5**
GMM	OF	70	100.00	**0.0000** *	**0.0000** *	**0.9479**	**0.9479**	22.8
	AER	10	14.29	0.0009	0.0003	0.8850	0.8847	8.1
	AEA	85	121.43	0.0122	0.0005	0.9287	0.9282	30.9
	PCA	**2***	**2.86***	0.0002	0.0006	0.8321	0.8316	**5.7**
nuSVM	OF	70	100.00	23.1632	0.3165	**1.0000** *	0.6835	3.0
	AER	10	14.29	7.9910	0.0683	**1.0000** *	0.9317	**0.9***
	AEA	93	132.86	**2.1082**	**0.0580**	**1.0000** *	**0.9420**	1.3
	PCA	**2***	**2.86***	2.3468	0.0630	**1.0000** *	0.9370	1.0
IForest	OF	70	100.00	**4.5572**	**0.1176**	0.8893	**0.7717**	6.1
	AER	**45**	**64.29**	14.9819	0.2134	0.9192	0.7058	**5.3**
	AEA	85	121.43	16.2062	0.1836	0.9462	0.7625	**5.3**
	PCA	70	100.00	12.2452	0.2568	**0.9823**	0.7255	5.9
LOF	OF	70	100.00	0.2762	0.0028	**0.9804**	**0.9777** *	95.9
	AER	10	14.29	0.0816	0.0029	0.9157	0.9128	**2.7**
	AEA	88	125.71	0.0522	**0.0015**	0.9448	0.9432	4.5
	PCA	**3**	**4.29**	**0.0064**	0.0032	0.8838	0.8805	4.2

Table 3.9 Performance metrics for all the preprocessing and model combinations. OF means original features, AER means autoencoder reduced, AEA means autoencoder augmented. n_f is the number of features transformed by the autoencoders or PCA. FP is the features percentage with respect to the original dimension. Variance defined in the Equation 3.12, and the reactivity defined in Equation 3.13. Train signal is set 1 of Table 3.6 and test signal is set 5. The symbols ⇓ and ⇑ indicate if the value should ideally be minimized or maximized, respectively. The best results among all the UML models and features transformation combinations are highlighted with *, for each metric.

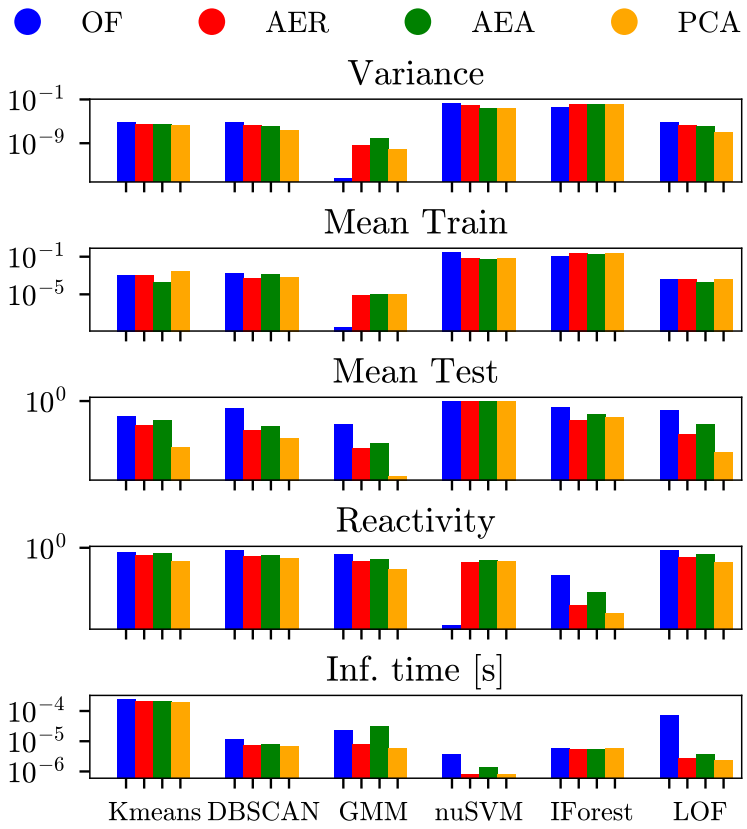


Fig. 3.16 Graphical representation of the performance metrics reported in Table 3.9. All y-axes are shown in logarithmic scale. AEA corresponds to the overcomplete autoencoder, AER to the undercomplete autoencoder, and OF to the original features.

The resulting metrics are listed in Table 3.9. For each UML model, the best transformation method for each metric is highlighted in bold. The best overall results across all models and transformations are additionally marked with an asterisk. A visual summary of these results is provided in Fig. 3.16.

Several observations can be made from Table 3.9. First, in terms of dimensionality reduction, PCA consistently produces the lowest number of output features, except when paired with the IForest model, for which the AER transformation achieves an even smaller dimensionality. Regarding the variance of the NM under nominal conditions, the GMM achieves the lowest variance, indicating an almost constant novelty metric for known data. Among the remaining UML models, KMeans, DBSCAN, and LOF achieve their lowest variance when used together with PCA, whereas IForest performs best when applied directly to the original features. Ideally, the NM evaluated on nominal data should be as low as possible. The lowest nom-

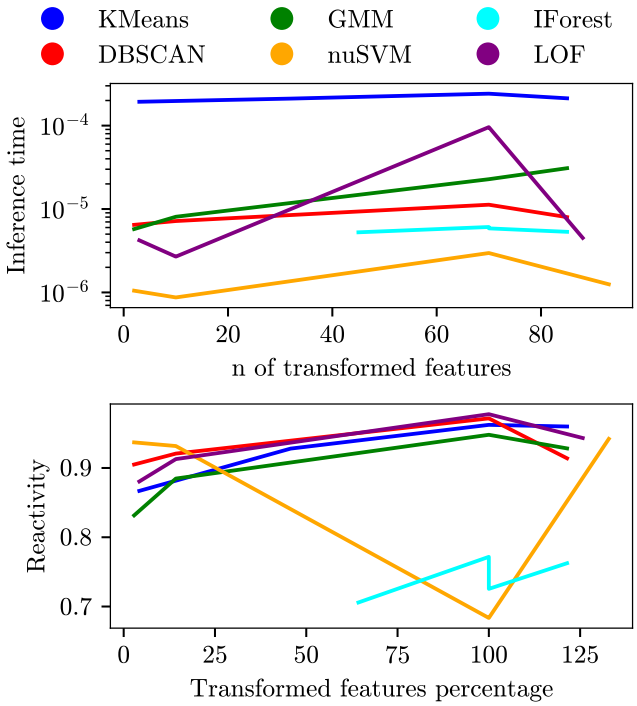


Fig. 3.17 Top: correlation between inference time and number of features. Bottom: correlation between the feature dimensionality reduction FP and the reactivity.

inal NM is obtained by GMM without feature transformation. Across other UML models, there is no clear pattern indicating that a specific transformation consistently minimizes this metric.

Conversely, the NM should be high when evaluating novel samples. The nuSVM exhibits the highest novelty values regardless of the transformation method. However, this behaviour comes at the cost of reduced continuity in the NM: the values tend to saturate, even when the deviation from the nominal signal is slight, limiting its usefulness for gradual anomaly severity estimation.

Reactivity tends to be highest when the UML models operate on the original features, with the exception of nuSVM, which displays its highest reactivity when used with the AEA. The LOF combined with the original features shows the overall highest reactivity, followed by the DBSCAN model, which exhibits a comparably strong response.

The nuSVM also provides the lowest inference time due to its simple decision function, which requires only a distance computation. For the other UML models,

evaluation time generally decreases as the dimensionality of the feature space is reduced.

To examine relationships between performance metrics, all metrics were plotted pairwise to highlight potential correlations. The most informative of these plots are shown in Fig. 3.17. The first plot confirms that inference time tends to increase with feature dimensionality. The second plot suggests that reducing the feature dimensionality (i.e., using FP$<$ 100) generally reduces the reactivity of the UML models, except for nuSVM, which displays the opposite behaviour.

For completeness, the novelty metric was computed for all remaining vibration sets (Sets 1–8), excluding the training portion of Set 1. The evolution of the normalized NM across all sets and all model/transformation combinations is shown in Fig. 3.18. KMeans, DBSCAN, GMM, and LOF show a clear progression of the NM that correlates with the increasing dissimilarity of the vibration signals (particularly amplitude changes). On the contrary, nuSVM and IForest saturate the NM to nearly constant values for novel conditions, effectively producing a binary output. This behaviour contradicts the main objective of this work, which is to derive a continuous NM reflecting anomaly severity.

Another observation from Fig. 3.18 is that GMM is significantly influenced by the choice of feature transformation. When fed with original features, GMM produces systematically lower NM values compared to the transformed feature cases.

3.2.5 Conclusion

In this work, several unsupervised machine learning models combined with different feature transformation strategies are benchmarked. A real vibration dataset is also gathered to evaluate all frameworks under realistic, noise-affected conditions. The results show that a continuous Novelty Metric is most effectively produced by KMeans, DBSCAN, GMM, and LOF, which are able to provide a meaningful indication of anomaly severity. Conversely, models such as nuSVM and IF tend to behave like binary detectors, limiting their usefulness for continuous degradation monitoring. The benchmark further demonstrates the critical role of both feature extraction and feature transformation in shaping the behaviour and performance of unsupervised models. Moreover, inference times are measured to quantify the computational cost associated with each configuration.

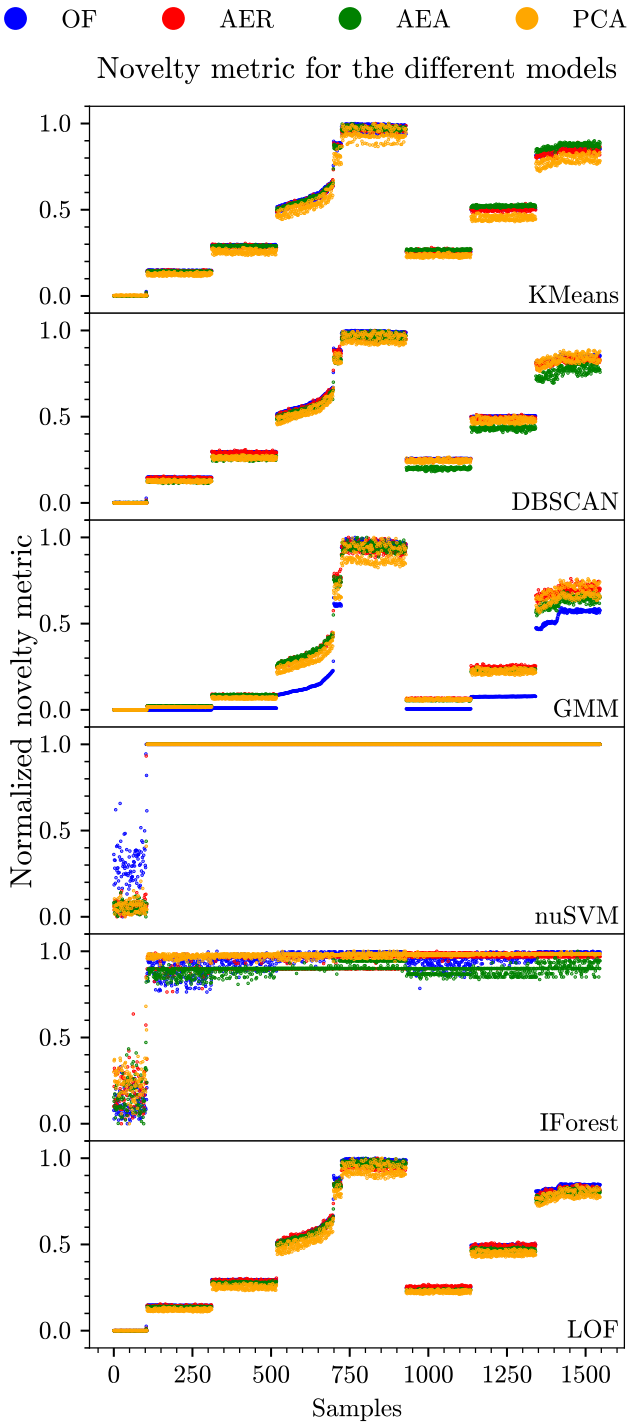


Fig. 3.18 Normalized novelty metric for all combinations of UML models and feature transformation methods. Vertical steps correspond to transitions between data sets with different characteristics. KMeans, DBSCAN, and LOF are less affected by the choice of feature transformation. The nuSVM exhibits strong saturation for novel data and struggles to maintain a stable NM for nominal samples. The IForest model shows similar saturation behaviour and a strong dependence on the transformation method.

Future work will focus on deploying and evaluating these frameworks on embedded devices to establish an EDGE-level benchmark. It is planned to explore additional feature extraction techniques, including generative models such as RealNVP, to assess whether they lead to improved performance. Finally, future works will extend the benchmarking campaign to other industrial datasets to further validate and generalize the findings presented in this study.

Chapter 4

Novelty Detection in Robotics

The rapid emergence of service mobile robots marks a significant step forward in automation, influencing a wide range of daily-life applications, including domestic and healthcare assistance [67, 68], precision agriculture [69, 70], and facilities inspection [71]. Localization technologies play a central role in ensuring the future development and reliability of autonomous mobile robots [72]. Although wheel odometry and visual odometry have been extensively studied in the past decade, both remain sensitive to long-distance drift and environments lacking repetitive visual features [73–75].

Among the localization methods explored in recent years, Ultra-Wideband (UWB) has emerged as a promising and cost-effective solution for mobile robot positioning, particularly in GPS-denied environments [76]. UWB operates by transmitting extremely short-duration pulses, enabling highly accurate time-of-arrival estimation and, consequently, precise distance measurements. Despite these strengths, UWB performance is strongly affected by Non-Line-of-Sight (NLoS) conditions and multipath reflections, which can significantly degrade range accuracy [77]. Such effects occur when direct signal propagation is blocked, or when reflections distort the received waveform, producing biased distance estimates that compromise autonomous navigation (Fig. 4.1).

Most classical UWB-based localization frameworks rely on Kalman filtering [78, 79]. In parallel, recent advances in machine learning have led to data-driven methods that model UWB reflectance patterns and NLoS-induced disturbances [80–82]. Although neural networks can capture complex nonlinearities, these techniques

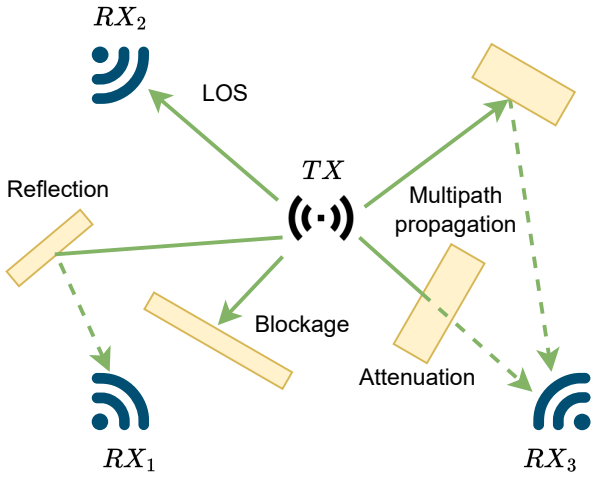


Fig. 4.1 UWB signal attenuation types depending on the obstacles within and around the environment.

typically require large datasets and must be retrained whenever environmental conditions change. Furthermore, the dynamic nature of real-world service environments complicates the identification of locations where UWB reliability may decrease.

Motivated by these challenges, these works aim to develop a robust approach to learn a precise, environment-dependent characterization of UWB range reliability and exploit it to adapt the localization pipeline across space and time.

The key contributions of these works are summarized as follows:

- A novelty-detection autoencoder for learning the environmental characterization of UWB range signals.
- A localization pipeline that dynamically adapts EKF covariance and bias terms using the range-specific novelty score, linked to the UWB reliability through simple, general mapping functions.
- An extensive experimental evaluation featuring increasing NLoS severity and dynamic environmental changes, a level of systematic analysis that, to the best of our knowledge, has not been previously reported.

Overall, these works aim to demonstrate the flexibility and adaptability of the novelty detection paradigm across a variety of application domains [50].

4.1 Semi-Supervised Novelty Detection for Precise Ultra-Wideband Error Signal Prediction

4.1.1 Related Works

Several studies have sought to mitigate these challenges. Machine learning and deep learning approaches, for example, have been applied to detect and compensate for NLoS conditions by formulating the problem as either a classification or regression task [81, 82]. Other works have integrated complementary information—such as inertial sensor data [83] or prior map knowledge [84]—to enhance UWB-based localization. In some cases, the receiver's position has been estimated directly from raw UWB data without explicitly identifying NLoS conditions in advance. These strategies have shown strong performance even in complex environments where maintaining a clear Line of Sight (LoS) is not always feasible [85]. Nevertheless, a comprehensive solution capable of robustly handling the diversity and dynamic characteristics of real-world environments is still missing.

In this work, it is emphasized the influence of the surrounding environment on UWB performance, acknowledging that UWB error is inherently dynamic and closely linked to the introduction of anomalies within the map. To address this, novelty detection is leveraged—a family of machine learning techniques designed to identify previously unseen or anomalous patterns relative to a learned nominal distribution [86]. In the context of UWB signals, novelty detection can be used to recognize NLoS and multipath effects as anomalous conditions, as well as detect meaningful environmental changes, thereby providing position-dependent uncertainty estimates essential for reliable localization.

Autoencoders are widely used in novelty detection tasks due to their capacity to learn compact representations of data [40, 87]. Within the domain of Ultra-Wideband (UWB) technology, autoencoders have been employed for various purposes, including assessing and correcting signal reliability. Variational autoencoders, for instance, have proven effective at assigning anomaly scores to the channels of Time-of-Flight (ToF) systems, demonstrating strong generalization capabilities and enabling their integration into Extended Kalman Filter–based tracking frameworks [88]. Additionally, autoencoders have been utilized to refine ranging measurements, improving localization accuracy [89].

4.1.2 Methodology

For this study, the environment considered consists in fixed UWB anchors and a mobile tag. A dataset is collected to build a reference database in which UWB-specific signal features are systematically recorded at predefined locations—this constitutes the offline phase. The localization problem is then formulated as a mathematical optimization task, where the goal is to identify the reference point that minimizes the localization error based on the signal characteristics observed during the online phase. This approach, widely used in wireless systems, is known as fingerprinting. Although effective in static environments, fingerprinting is highly sensitive to environmental changes, which can significantly alter signal properties.

The method proposed in this work addresses this limitation by detecting when the signal becomes unreliable through deep learning–based novelty detection techniques. Then, it is introduced an approach termed UWB Error Prediction with Semi-Supervised Novelty Detection (EPSNoDe), designed to identify anomalous signal conditions that could compromise localization accuracy.

Novelty Identification

In our work, novelty detection plays a key role in identifying warning zones, i.e., areas where localization performance deteriorates due to environmental factors. The novelty detection framework is trained exclusively on nominal data. Because the model is expected to recognize deviations during inference, such altered conditions must not be included in the training set. This design intentionally induces a certain level of overfitting, making the model highly specialized to the nominal training distribution while limiting its generalization capabilities.

Given the adoption of a fingerprinting approach, the novelty detection models are trained on feature sets that remain consistent for each position of the grid map. The core objective of this study is to detect environmental changes reflected in UWB signals. The model learns the characteristic signal pattern associated with each grid location and identifies when a previously unseen pattern emerges. When an obstacle is introduced into the environment, UWB reflections and NLoS conditions modify the signal characteristics, producing distortions relative to the original pattern. As a result, the model detects these deviations and flags them as novelties at the corresponding positions.

Novelty identification is based on reconstruction error, which quantifies the discrepancy between the observed signal features and those reconstructed by the model. Two error metrics are employed. The first measures the reconstruction error for a single UWB anchor by computing the absolute difference between the real and reconstructed features:

$$e = |\hat{y} - y| \tag{4.1}$$

where e represents the error, $\hat{y}$ represents the predicted sample, and y represents the real one. The second error is computed by using the errors of i_{th} anchors and summing them as a multidimensional distance, as reported in the following equation:

$$e_{total} = \sqrt{e_1^2 + e_2^2 + \cdots + e_n^2} \tag{4.2}$$

where e_{total} is the total error computed for a single prediction, e_i is the error of the i_{th} anchor prediction computed by using Equation 4.1, and n is the total number of anchors used for the robot's localization.

Neural Network Architecture

The architecture adopted in this work is an autoencoder. The architectures of the autoencoder can assume several shape following the same rule described in Section 3.2.3.

In this work, the proposed architecture is an overcomplete autoencoder, as illustrated in Fig. 4.2. The encoder consists of three dense layers with dimensions N, N_{E1}, and N_{E2}, where $N = n$, $N_{E2} = p$, and $N < N_{E1} < N_{E2}$. All encoder layers use *ReLU* activation functions. The decoder is composed of two layers of dimensions N_{D1} and N, with $N_{D1} > N$ to restore the original dimensionality. The first decoder layer uses a ReLU activation, whereas the final layer employs a *Leaky ReLU* activation to mitigate the risk of dead neurons and enhance overall robustness. The network is trained using a Mean Squared Error (MSE) loss function.

Different UWB data-selection and pre-processing strategies are explored to extract informative features from the raw sensor signals. Three configurations of the proposed method are considered:

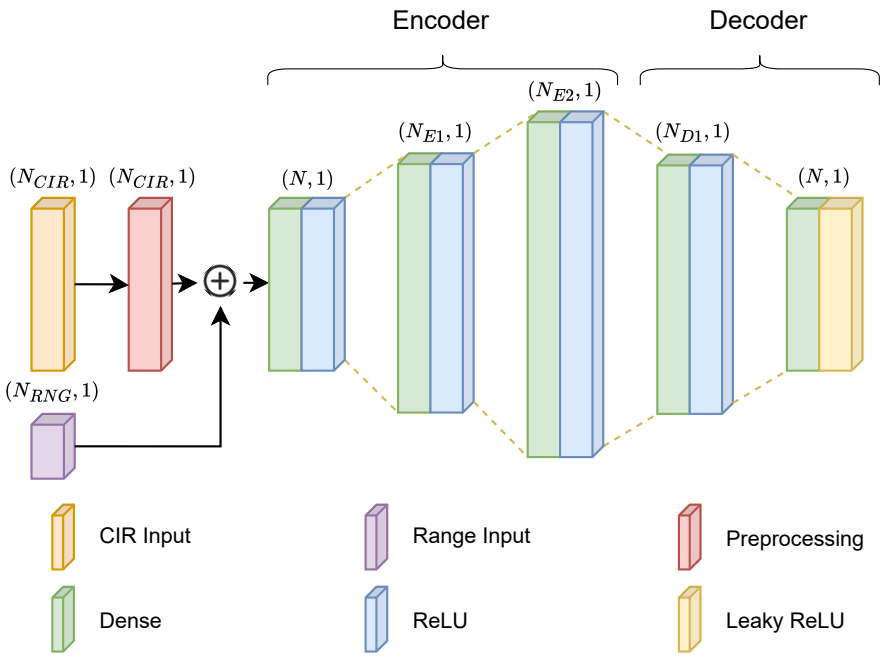


Fig. 4.2 Architecture of the proposed UWB EPSNoDe model. It consists of an overcomplete autoencoder ($N < N_{E1} < N_{E2}$ and $N_{D1} > N$). The legend indicates all layer types. The input dimension varies depending on the selected input modality (i.e., CIRs and/or ranges, according to the adopted model configuration).

- **EPSNoDe$_{\mathbf{RNG}}$**: uses only the anchors' range measurements as inputs;
- **EPSNoDe$_{\mathbf{MA}}$**: combines anchors' ranges with the first six peaks of the Moving Average applied to the Channel Impulse Response (CIR);
- **EPSNoDe$_{\mathbf{PCA}}$**: combines anchors' ranges with a PCA-reduced CIR representation obtained by applying Principal Component Analysis directly to the CIR signal.

Position-specific Error Prediction

The final stage of the framework focuses on identifying unreliable zones within the grid map for UWB localization, using the output data predicted by the *EPSNoDe* model. The computed reconstruction errors serve as key indicators to determine whether a specific grid point is anomalous with respect to the nominal environment.

Detecting an anomaly in a particular grid cell suggests that an environmental change may have occurred in the vicinity of that location.

A possible strategy for leveraging these errors is to define a threshold, which would trigger a warning whenever the reconstruction error exceeds it. At such positions, the autonomous robot becomes aware that the localization algorithm may be compromised, enabling the implementation of additional strategies to improve UWB-based localization accuracy. These strategies may include error mitigation or compensation techniques. However, the design of such strategies and the selection of an appropriate error threshold fall outside the scope of this paper.

Several experiments are conducted to evaluate the performance and functionality of the proposed framework.

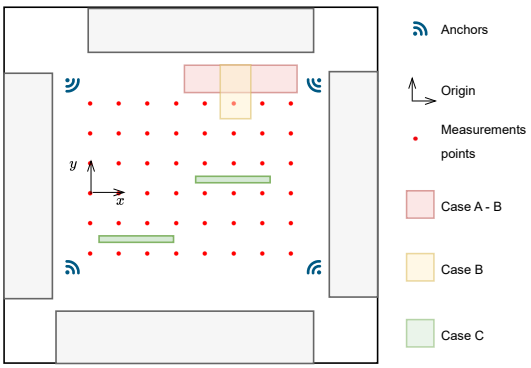


Fig. 4.3 Sketch of the environment used to test the algorithm. Three novelty scenarios are identified: A, B, and C.

4.1.3 Tests and Results

Experimental Setup

The environment used for evaluating the framework is an office room at the PIC4SeR center. The room contains various desks, chairs, and wardrobes. A sub-area is defined within a free rectangular region, allowing the robot to move unobstructed. The robot follows a predefined rectangular grid path with a mesh size of 50 cm along both the x and y directions.

Three experimental scenarios are considered, as illustrated in Fig. 4.3.

- **Case A**: a metal plate is placed near the top-right corner of the grid map.
- **Case B**: a wooden bridge and a metal plate are positioned in the same top-right area, creating NLoS conditions at several grid points.
- **Case C**: two obstacles are placed directly within the grid map.

In all experiments, the remainder of the environment is kept unchanged. Four datasets—covering nominal and perturbed scenarios—are collected during the experimental campaign. The added obstacles are taller than the robot, thereby introducing NLoS conditions by obstructing visibility to the anchors in certain regions of the grid during the measurement phase.

Experiments and Results

An extensive experimental session is carried out to demonstrate the effectiveness of the method in characterizing the quality of the UWB signal in the environment. Moreover, the role of UWB data selection is analyzed in the learning process. More precisely, three different pre-processing operations are defined for the proposed autoencoder architecture described in 4.1.2. Each network configuration is tested on scenarios A, B, and C, and the results obtained are shown in Fig. 4.4. Each quadrant shows the total error computed for the related position, where a bright quadrant indicates a high error.

The first model analyzed is the $EPSNoDe_{RNG}$, which specifically detects novelty based only on ranging distances. In this scenario, the model employs the minimum number of features compared to other models. Using four anchors, the features are limited to four, corresponding to the ranging values of each anchor. As can be noticed, the model detects the presence of obstacles in both Case B and Case C, where obstacles are positioned within the map. The obstacle in Case A is less visible using only ranging distance, possibly due to persistent LoS conditions throughout the measurement phase. By exclusively considering ranging distance, potential reflections of the CIR are neglected in the analysis.

On the opposite side, the $EPSNoDe_{PCA}$ model exhibits a contrasting performance: in this instance, a more extensive set of features is considered. The original CIR

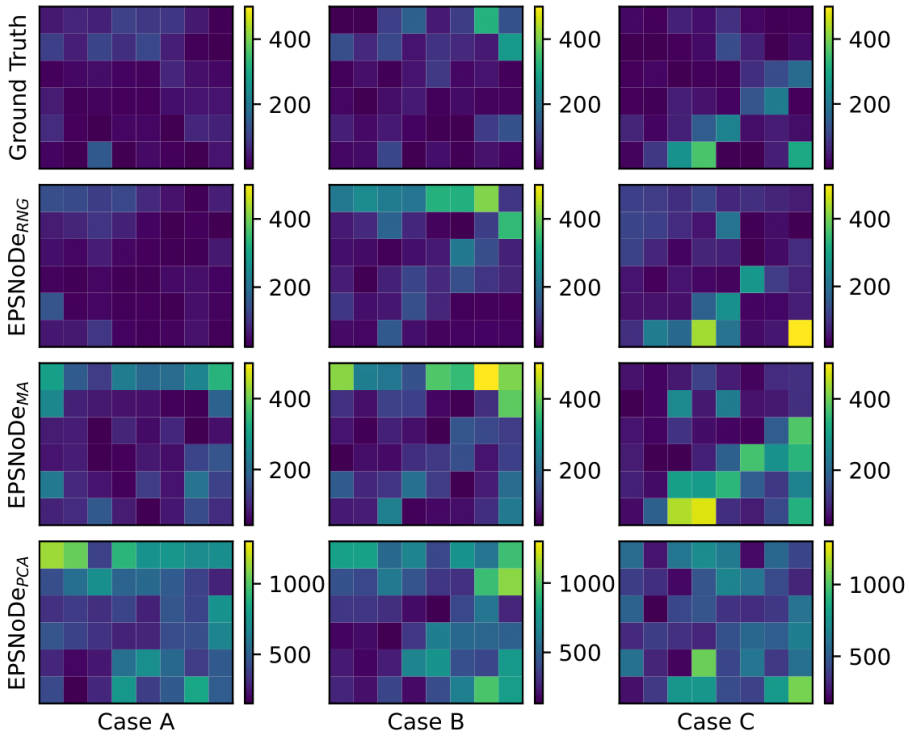


Fig. 4.4 The total error is computed for each test conducted. Each row corresponds to the architecture type applied to the dataset, and each column corresponds to the dataset type employed in the test. *EPSNoDe*$_{RNG}$ uses only ranging distances, *EPSNoDe*$_{MA}$ involves the Moving Average of the CIR along with the ranging distances, and *EPSNoDe*$_{PCA}$ involves the application of Principal Component Analysis to the CIR along with the ranging distances.

comprises 152 samples multiplied by the number of anchors, resulting in 608 features for our case with four anchors. The PCA model reduces the feature dimension while retaining 90% of the variance of the initial data, reducing the features to 68 elements. Adding the four UWB ranges results in 72 input features for the framework. Due to this elevated number of features, this architecture encounters difficulties in learning the patterns for each point of the grid map. The results exhibit a considerable level of uncertainty in the overall map.

The most promising outcomes in this study are associated with the *EPSNoDe*$_{RNG}$ and *EPSNoDe*$_{MA}$, where a two-period moving average is applied to each anchor's CIR to filter the raw signal and its reflections. In the *EPSNoDe*$_{MA}$ approach, the

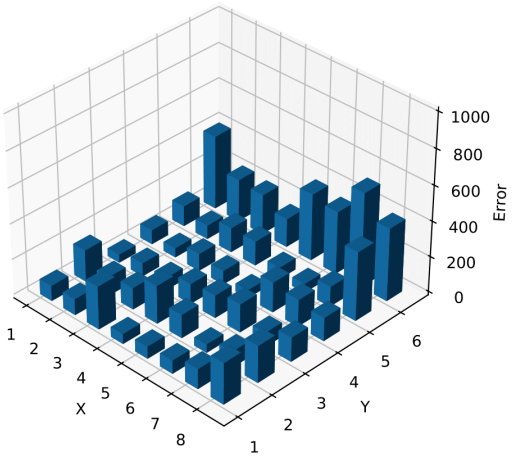


Fig. 4.5 3D plot obtained using the $EPSNoDe_{MA}$ architecture in Case B with the obstacle placed within the grid map in the top-right corner. The graph shows the total error computed by applying Eq. 4.2 for each map grid point. The highest errors are detected in the top-right corner of the map.

framework considers the first six peaks of the CIR along with the ranging distances as inputs.

The obstacles in Cases B and C are clearly visible in the map. Case A, involving an obstacle beside the map, is harder to recognize. Only the $EPSNoDe_{MA}$ detects some novelties in the top row of the map. These distinctive elements are recognized due to UWB signal reflections interacting with the obstacle outside the map. By incorporating the first six peaks of the CIR into the range distances, significant signal reflections are integrated and detected as novelties. A detailed visualization is carried out for the $EPSNoDe_{MA}$ applied to Case B to better understand the warning-zone identification. Fig. 4.5 shows the total error resulting from applying Eq. 4.2.

The plot shows the framework's difficulty in reconstructing the nominal input values when close to the perturbation due to untrained data (e.g., high error). In Case B, the novelty is located in the top-right corner of the map. This obstacle introduces alternation between LoS and NLoS conditions during UWB measurements. The discrepancy appears near the perturbation and along the top line of the grid map. This behavior is tied to reflections induced by metal plates introduced in the environment, altering the ToF of the UWB signal. When considering the first six

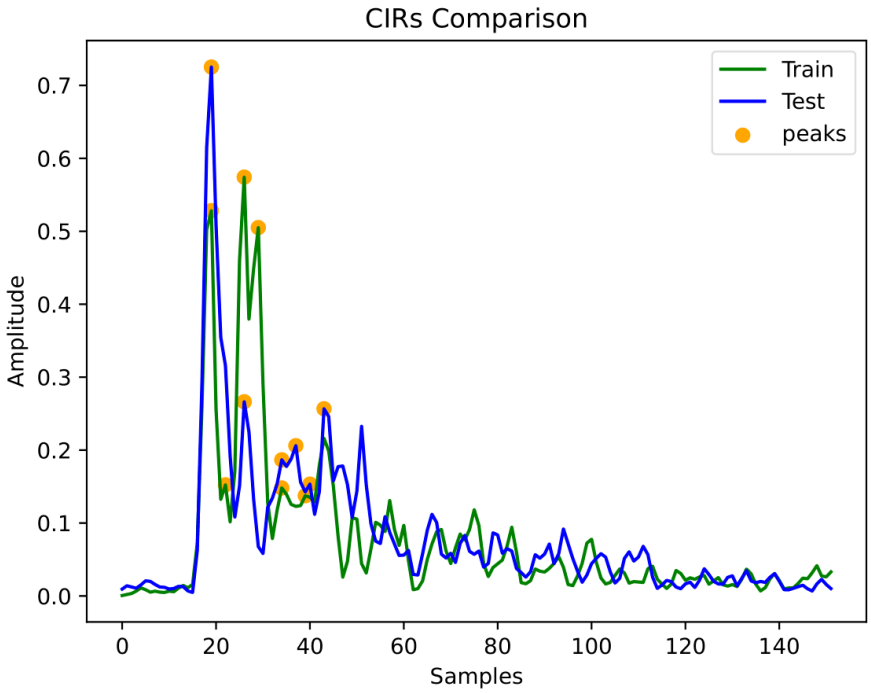


Fig. 4.6 Comparison between the normal CIR and the anomalous one, both obtained at the same grid point. The discrepancies arise from reflections caused by a new object near that grid point.

peaks of the CIR ($EPSNoDe_{MA}$), the reflections become more prominent, facilitating novelty detection.

A clearer explanation of this point is presented in Fig. 4.6, which displays both the normal and anomalous CIR. The two signals differ substantially due to reflections caused by the new obstacle. Trained exclusively with nominal CIR signals, the model fails to reconstruct the anomalous one, leading to a predicted range that diverges from the real one and contributes to the global error shown in Fig. 4.5.

Another intriguing aspect of this generative framework is the ability to identify the anchor that mainly contributes to the total error. The representation of the error for each anchor is shown in Fig. 4.7. In this instance, the error for each anchor is computed using Eq. 4.1, which is the relative difference between the real and predicted range. Notably, the top-right and bottom-left anchors significantly contribute to the total error, as shown in the heat map, suggesting the presence of an

obstacle between them. From the anomaly grid zone, it is reasonable to infer that the obstacle lies in the top-right corner.

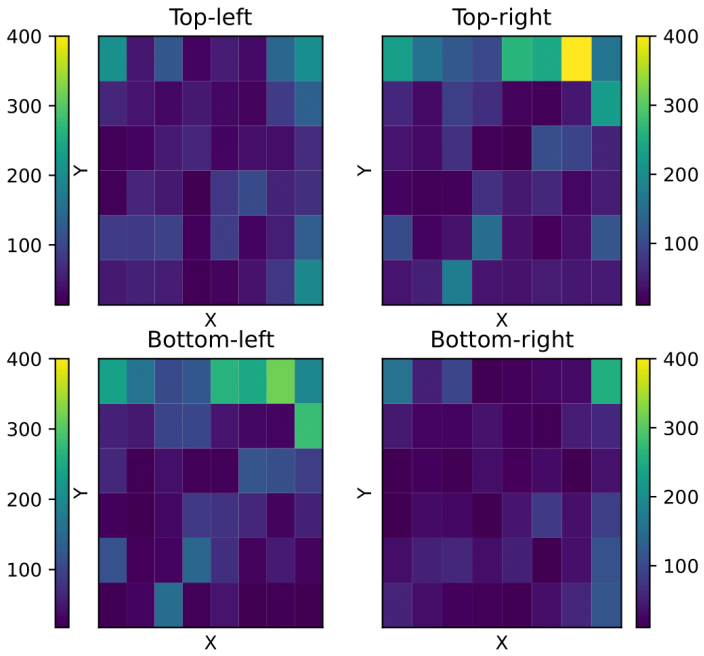


Fig. 4.7 Heat maps obtained by applying Eq. 4.1 for each anchor. The novelty is placed in the top-right corner of the map. The highest errors are detected from the top-right and bottom-left anchors.

Finally, the results shown in Fig. 4.4 are obtained by testing several hyperparameter combinations to find the best optimization for each network. This optimization involves varying the number of neurons, batch size, and learning rate to achieve optimal performance. The combinations considered are summarized in Table 4.1.

The conducted tests also varied the number of training samples to explore potential limitations from insufficient data. Cases A and B utilize five datasets for training, whereas Case C uses only two. As shown in Fig. 4.4, the framework detects novelties even with only two datasets for training. However, increasing the number of datasets enhances the reliability of novelty detection due to a better overfit of the nominal environment. The best hyperparameter combinations for each model are shown in Table 4.2.

Architecture	E1,D1 Neuron	E2 Neuron	Batch Size	Learning Rate
$\text{EPSNoDe}_{\text{RNG}}$	5,15, 20	20,30, 40	16,32,64	0.001,0.01
$\text{EPSNoDe}_{\text{MA}}$	50,55,60 65,70	70,75,80 85,90	16,32,64	0.001,0.01
$\text{EPSNoDe}_{\text{PCA}}$	120,125,130 135,140	145,150,155 160,165	16,32,64	0.001,0.01

Table 4.1 Hyperparameters combinations examined for model optimization.

Architecture	E1,D1 Neuron	E2 Neuron	Batch Size	Learning Rate
$\text{EPSNoDe}_{\text{RNG}}$	15	30	32	0.001
$\text{EPSNoDe}_{\text{MA}}$	70	90	64	0.001
$\text{EPSNoDe}_{\text{PCA}}$	120	165	32	0.001

Table 4.2 Best hyperparameter chosen after the optimization

Error Density Quantitative Evaluation

In the last step, a similarity metric is employed to enhance the visualization of the fit between each model and the ground truth, providing a single quantitative value to measure this agreement. To this end, the heat maps are transformed into Probability Density Functions (PDFs) using the Kernel Density Estimation (KDE) method. Subsequently, the Kullback–Leibler (KL) divergence is used to quantify the similarity between the predicted PDF and the ground truth. The KL divergence is defined as

$$D_{KL}(P||Q) = \sum_{x \in X} p(x) \cdot log(\frac{p(x)}{q(x)}) \tag{4.3}$$

where p and q are the two PDFs being compared. The KL divergence is always non-negative, i.e., $D_{\text{KL}}(P\|Q) \geq 0$. The results obtained by applying the KL divergence metric are reported in Table 4.3.

The model that best fits the ground truth is the $EPSNoDe_{RNG}$, which relies solely on ranging distances to detect novelties. Conversely, the $EPSNoDe_{MA}$ shows

Architecture	Case A	Case B	Case C
$\text{EPSNoDe}_{\text{RNG}}$	0.15	0.19	0.14
$\text{EPSNoDe}_{\text{MA}}$	0.50	0.46	0.31
$\text{EPSNoDe}_{\text{PCA}}$	0.83	1.36	2.15

Table 4.3 KL divergences computed for each model, comparing the predicted heat map's PDF with the ground truth's PDF.

compelling results in both the metric table and the heat maps: although it does not provide the closest match to the ground truth, it effectively highlights where novelties occur in the environment, offering meaningful spatial insight into changes. Thus, $EPSNoDe_{RNG}$ can be effectively used to mitigate localization errors, while $EPSNoDe_{MA}$ is particularly suited for detecting subtle environmental variations.

4.1.4 Conclusion

The objective of this study is to advance the understanding of how UWB signals can be employed in autonomous robot localization. Rather than correcting localization errors through mitigation or compensation techniques, this paper focuses on mapping a specific environment to identify changes in UWB signal reliability within it.

Future work will involve integrating the proposed method into robot localization and navigation tasks. The UWB error information produced by the presented framework can be incorporated as prior knowledge within sensor fusion strategies or included as a cost term in navigation pipelines.

Additionally, further research will explore alternative approaches for novelty detection. A promising direction involves the use of Normalizing Flow models, which differ from autoencoders by directly estimating the density distribution of the input data. This property enables more principled computation of reconstruction-based anomaly measures and may provide an enhanced understanding of UWB signal deviations.

4.2 Adaptive Robot Localization with Ultra-wideband Novelty Detection

4.2.1 Related Works

UWB technology has emerged as a leading solution for precise localization, particularly in complex indoor environments where traditional systems are ineffective. Its broad frequency spectrum enables high temporal resolution, allowing accurate distance measurements and positioning [90]. This positioning capability has become increasingly popular due to its centimeter-level accuracy, enabled by the high temporal resolution of UWB pulses [77, 91]. Moreover, UWB signals are inherently resilient to interference from other radio technologies due to their distinct signal characteristics and frequency usage, and they can penetrate various materials, although they remain susceptible to significant signal distortions.

Most UWB-based localization systems follow a two-step process. First, parameters related to the target node are estimated using fixed anchor nodes deployed at known positions. Among the most common parameters are Time of Arrival (ToA), Angle of Arrival (AoA), and Received Signal Strength (RSS), all of which play a major role in range computation [92]. In the second phase, these estimated parameters are used by localization algorithms to determine the position of the target node.

The integration of UWB technology with advanced filtering techniques, such as Kalman filters, has become a focal point of research aimed at improving accuracy and reliability in indoor localization. Numerous studies show that combining UWB with other sensing modalities significantly enhances position estimation in complex environments. Kalman filtering, in particular, enables sensor fusion, combining UWB data with additional information to obtain improved localization accuracy. Several examples of sensor fusion can be found in the literature, where UWB is integrated with inertial sensors [78, 93], visual–inertial systems [94], and GPS–inertial systems [95].

In recent years, machine learning (ML) techniques have gained significant traction in enhancing UWB-based localization, as extensively discussed in the literature. ML models have been employed either to correct individual anchor-to-tag range measurements or to refine the final position estimate. For instance, deep autoencoders have been proposed for direct 2D position estimation based on time-

difference-of-arrival measurements [96]. More sophisticated neural networks, including recurrent neural networks—and especially Long Short-Term Memory (LSTM) architectures—have been utilized to estimate tag positions by learning temporal dependencies and extracting high-level representations from UWB features [97–99]. Convolutional neural networks have also been explored: for example, [100] proposes converting received signals into RGB-image-like representations to serve as neural network inputs.

Several studies instead focus on mitigating errors in individual range measurements. Deep learning and machine learning models have been proposed to detect and compensate for NLoS conditions, addressing the problem through both classification and regression formulations [80–82, 101]. Additional approaches incorporate complementary data sources, such as inertial measurements [83] or map information [84], to enhance localization robustness. In some cases, receiver positions have been estimated directly from raw UWB data, avoiding explicit NLoS identification; these approaches have demonstrated excellent performance even in challenging environments where maintaining Line-of-Sight (LoS) is difficult [85]. Nevertheless, a comprehensive solution capable of robustly addressing the diverse and dynamic nature of real-world environments is still lacking.

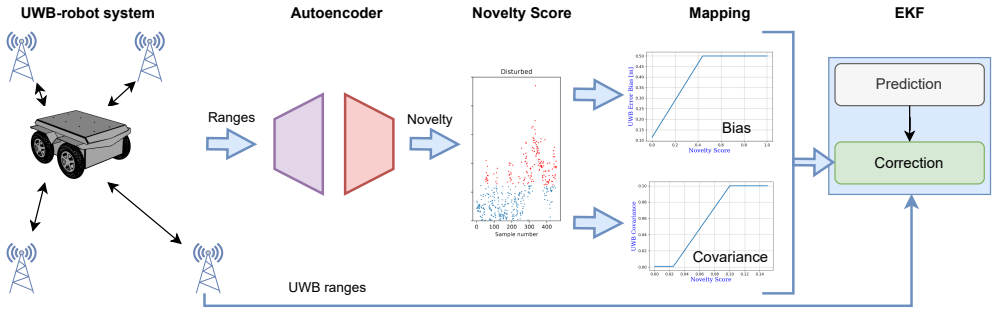


Fig. 4.8 The complete pipeline of the proposed adaptive UWB positioning solution. UWB ranges are used to feed a novelty autoencoder, the novelty score is then dynamically mapped into a correction bias and a covariance value for the correction step of the EKF.

Our method integrates an Extended Kalman Filter (EKF) with the benefits of unsupervised neural-network-based novelty detection. Specifically, an autoencoder is trained to detect subtle deviations in UWB ranges, estimating a novelty score that reflects local signal uncertainty. As illustrated in Figure 4.8, this novelty score is then mapped to position-dependent covariance and bias terms for the EKF correction step.

Unlike fingerprinting-based approaches [50], our method requires only a limited number of nominal robot trajectories for training, with no need for dense sampling or labeling.

In this study, it is emphasized the central role of the environment in determining UWB performance, given that UWB measurement errors are highly dynamic and strongly influenced by environmental anomalies. Novelty detection provides a powerful set of machine learning techniques for identifying new or previously unseen patterns with respect to nominal data [86]. In the context of UWB ranging, novelty detection can identify NLoS and multipath conditions as novel events, as well as detect significant environment-induced changes, thereby providing position-specific uncertainty information essential for reliable localization.

As it is demonstrated in this work, dynamically adapting the measurement covariance of the Extended Kalman Filter (EKF) can significantly improve localization performance by embedding environmental information directly into the filtering process. For example, [79] employs a neural network to model the relationship between UWB positional errors and signal-quality metrics such as precision estimates and RSS. Their results show that neural networks can effectively learn sensor covariance and improve EKF robustness by adapting to varying measurement reliability. However, because their neural network is trained on the difference between UWB-based and ground-truth positions, the method requires large quantities of labeled data and ground-truth measurements for every new environment—an often impractical requirement.

A different approach is introduced in [102], which proposes an empirical variance model derived from time-of-flight (ToF) estimates and enhanced using first path power (FPP) measurements. Building on this model, the authors design a robust M-estimation Kalman filter (M-RKF) capable of handling multipath-induced outliers that frequently compromise measurement reliability.

Autoencoders have become widely used in novelty detection due to their strong ability to learn compact latent representations of nominal data [40, 87]. In UWB applications, they have been employed for both assessing and correcting signal reliability. Variational autoencoders, for example, have been used to assign anomaly scores to channels in ToF-based systems, demonstrating strong generalization capabilities. These anomaly scores can be incorporated into a Kalman filter for improved tracking, as shown in [88]. However, this method relies on Channel Impulse Re-

sponse (CIR) data, which is highly sensitive to noise and may impose significant computational burdens on embedded hardware. Autoencoders have also been utilized to enhance UWB range measurement precision in localization applications [89].

4.2.2 Methodology

UWB localization relies on Time-of-Arrival (ToA) range measurements between a tag and multiple anchors. However, these measurements are often affected by environmental factors such as metallic obstacles and electronic interference, which introduce significant errors in ToA estimation. Consequently, corrupted ranges propagate through the system and degrade the overall positioning accuracy.

This paper addresses the limitations of UWB-based localization by proposing an effective and general solution, particularly suited and validated for small, cluttered, and noisy indoor environments where ToA inaccuracies severely hinder precise localization. The proposed approach leverages the capability of Deep Neural Networks to detect novelty patterns in range measurements, combined with the robustness of classical estimation and filtering techniques. The resulting localization framework is composed of two key modules: a Novelty Detection Neural Network (NDNN) and an Extended Kalman Filter (EKF).

The full pipeline of the proposed system is depicted in Figure 4.8. The complete implementation of the proposed system is publicly available in the project repository[1]. As a preliminary step, a UWB dataset is collected to characterize the nominal behavior of range measurements within the environment. This dataset is used to train the NDNN, enabling it to learn the feature patterns representative of nominal range values. During operation, the NDNN output is compared with the current input data to compute a novelty score, defined as the reconstruction error between predicted and observed ranges. This novelty score is subsequently mapped to a covariance value and a bias term for the EKF correction step through simple activation functions. By integrating the adaptability of the NDNN with the robustness of the EKF, the proposed system provides an environment-aware localization framework capable of adapting to a wide variety of conditions.

The following sections describe the working principles and architectural details of each component of the framework.

[1] https://github.com/PIC4SeR/UWB-Nov-localization

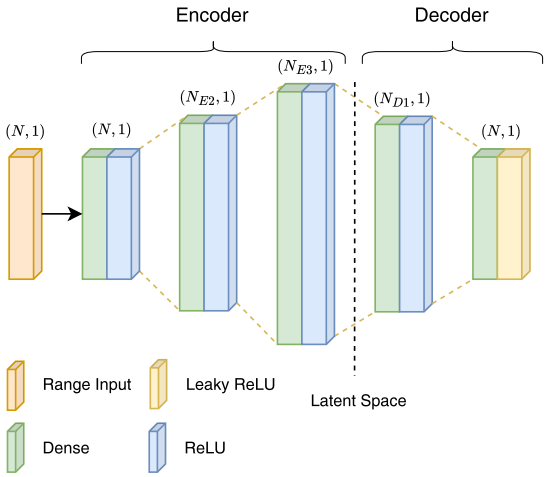


Fig. 4.9 The autoencoder architecture. N is the number of neurons of the first encoder layer and the last decoder layer. N_{E2}, N_{E3} are the neurons of the second and the third encoder layers. N_{D1} is the number of neuron of the first decoder layer.

Neural Network Architecture

The neural network architecture adopted is a semi-supervised autoencoder. In this work, an overcomplete autoencoder is employed. This choice is motivated by the relatively low dimensionality of the UWB range inputs, which limits the representational capacity of undercomplete models and hinders their ability to learn meaningful feature mappings. The adopted architecture is illustrated in Fig. 4.9. The network input consists of normalized UWB ranges between the tag and the anchors. The encoder progressively expands the dimensionality of the input until reaching the latent space of size k, whereas the decoder mirrors this process by compressing the latent representation back to an output of dimension n.

Formally, the encoder layer sizes satisfy $N < N_{E2} < N_{E3}$, and the decoder layer sizes satisfy $N_{E3} > N_{D1} > N$, where $N = n$ denotes both the dimensionality of the input and the output layers, and $N_{E3} = k$ denotes the dimensionality of the latent space. Layers N_{E2} and N_{D1} correspond to the intermediate encoder and decoder layers, respectively. All layers use ReLU activation functions except for the final decoder layer, which employs a Leaky ReLU to mitigate zero-saturation effects and improve reconstruction robustness.

Data collection and novelty learning

In the majority of previous works, UWB range errors are learned through extensive dataset collection and manual annotation [79, 97, 100]. In contrast, the proposed novelty-based method adopts a semi-supervised learning strategy, avoiding the costly effort of manually labeling data. Furthermore, it is demonstrated with this work that it is possible to learn a complete characterization of UWB range signals in a given environment under nominal conditions without relying on fine-grained fingerprinting, as required in recent studies such as [50]. Instead, the novelty autoencoder is trained using UWB range data acquired from only a small number of robot trajectories. This represents a key advantage of the proposed approach, opening the door to future extensions involving online learning and continuous adaptation of the neural network during robot operation.

Novelty score computation

As described in the previous paragraph, once the model is trained, its output should ideally match the input under nominal conditions. To quantify deviations from this behavior, a novelty score is computed to measure the discrepancy between each input range and its reconstructed counterpart. The novelty score for the i-th anchor is defined as:

$$e_i = |X_{\text{recon},i} - X_i| \tag{4.4}$$

where $X_{recon,i}$ is the reconstructed range of the i^{th} anchor, while X_i is the input range of the same i^{th} anchor. Thus, the number of novelty scores equals the number of anchors in the system. The underlying idea is that when nominal environmental conditions change, the autoencoder can no longer accurately reconstruct the input, resulting in a higher discrepancy between the input and output. An increase in the novelty score therefore indicates the presence of an anomaly at that location. Such discrepancies typically arise from environmental changes—such as obstacles—that alter the propagation of the UWB signal.

Adaptive robot localization with EKF

The overall system combines the novelty-detection output with a standard Extended Kalman Filter (EKF) for position tracking of a mobile robot. This research focuses on improving the adaptivity of UWB-based localization to disturbances arising from both static environmental structure and temporal changes in the scene. For static filter-based approaches, evolving environments and varying signal-disturbance conditions are difficult to handle, particularly when attempting to define a single set of covariance values that remains valid across mixed conditions. Conversely, learning-based methods would typically require re-training on datasets collected under the new environmental conditions.

The proposed method aims to dynamically achieve sufficient flexibility and adaptation to previously unseen UWB range conditions by integrating the novelty information into the EKF correction step. The novelty scores computed for each UWB anchor by the autoencoder are exploited to estimate both a correction bias for the range measurements and the associated covariance values regulating the expected measurement uncertainty.

The state used in the EKF is defined using a constant-velocity motion model in a fixed global reference frame:

$$\begin{aligned} x_{t+1} &= x_t + v_x \cdot dt + \delta_x \\ y_{t+1} &= x_t + v_x \cdot dt + \delta_y \\ v_{x,t+1} &= v_{x,t} + \delta_{vx} \\ v_{y,t+1} &= v_{y,t} + \delta_{vy} \end{aligned} \tag{4.5}$$

where x, y and v_x, v_y represent the robot position and velocity along the X and Y axes, and δ denotes white noise modeling uncertainties arising from unmodeled dynamics and actuation errors.

The UWB measurement model is defined as the Euclidean distance between the estimated robot pose and each UWB anchor:

$$\begin{aligned} h_{rng}(\mu) = \|(\mu - \gamma)\|_2 = \|\varepsilon\|_2 = \sum_{k=1}^{N} |\varepsilon_k|^{1/2} = \\ = \sqrt{(x_r - x_{uwb})^2 + (y_r - y_{uwb})^2 + (z_r - z_{uwb})^2} \end{aligned} \tag{4.6}$$

where μ is the robot pose and γ is the pose of a given UWB anchor. For wheeled robots, the distance along the z-axis is constant, since the robot moves in the X–Y plane and the tag is mounted at a fixed height relative to the anchors.

In standard EKF formulations, the measurement covariance matrix is constant and defined using prior knowledge of sensor reliability. The diagonal matrix $\mathbf{R}_t$ contains the variances $\sigma^2_{i,t}$ of the i-th UWB range measurement at time t, assuming no cross-correlations:

$$\mathbf{R}_t = \begin{bmatrix} \sigma^2_{1,t} & & \\ & \ddots & \\ & & \sigma^2_{a,t} \end{bmatrix} \tag{4.7}$$

However, UWB ranges exhibit variable quality depending on the geometry and material properties of the environment. It is therefore difficult to define global variance values that accurately represent confidence across all locations. The core idea of this work is to dynamically set the covariance of each UWB measurement by mapping the novelty score to a corresponding covariance value. Figure 4.10 (left) shows the mapping function used to translate the predicted novelty score into a covariance value for the EKF correction step. Covariance values range from a minimum of 0.001 for highly confident ranges to a maximum of 0.1 for ranges deviating substantially from nominal conditions.

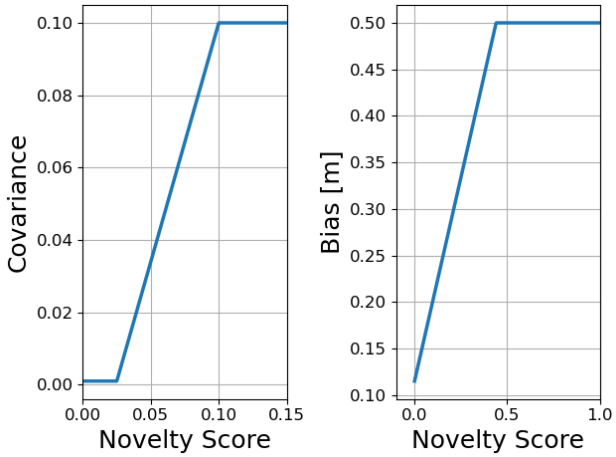


Fig. 4.10 Mapping functions used to transform the estimated novelty score into a bias (right) and a covariance (left) term for the UWB correction step of the Extended Kalman Filter. These functions are kept equal for all the experimental scenarios. The bias term is estimated from novelty scores using a mapping function able to refine the range value passed to the EKF.

Although the covariance informs the filter about the confidence in each UWB measurement, the correction step is still carried out using all ranges, including potentially inaccurate ones affected by multipath reflections. To address this, a bias term is also estimated from the novelty scores via a separate mapping function to refine the range values passed to the EKF. The linear transformation is shown in the right panel of Figure 4.10. Its shape mirrors that of the covariance mapping but with different slope and saturation parameters, chosen empirically based on the average discrepancy observed in the training dataset.

More specifically, the bias function is derived by fitting a first-degree polynomial $y = mx + q$ to training data, modeling the relationship between novelty score and range error. The fitting uses reconstructed ranges obtained via Vicon-based inverse trilateration and the raw UWB ranges. The coefficients are computed per anchor and then averaged. The resulting values are $m = 0.885$ and $q = 0.115$ m, with a saturation threshold fixed at 0.5 m for outlier rejection. A non-zero bias is assigned even for zero novelty scores, which is reasonable since typical indoor environments already introduce baseline multipath effects not encoded in the novelty score. The saturation prevents excessively large corrections for extreme novelty values.

These lightweight yet general mapping functions represent an improvement over previous works that estimate biases and covariances from data using an external ground-truth system [81]. The experimental results in Section 4.2.3 highlight the benefits of integrating an error-bias estimate derived from novelty detection.

The EKF correction step is performed using:

$$\mathbf{K}_t = \Sigma_t \mathbf{H}^T (\mathbf{H}\Sigma_t \mathbf{H}^T + \hat{\mathbf{R}}_t)^{-1} \tag{4.8}$$

$$\mu_{t+1} = \mu_t + \mathbf{K}_t (z_t - \hat{\zeta}_t - \mathbf{H}\mu_t) \tag{4.9}$$

$$\Sigma_{t+1} = (\mathbf{I} - \mathbf{K}_t \mathbf{H})\Sigma_t \tag{4.10}$$

where $\mathbf{H}$ is the Jacobian of the measurement model, $\hat{\mathbf{R}}_t$ is the dynamically estimated covariance derived from novelty scores, and $\hat{\zeta}_t$ is the bias correction applied to the raw UWB ranges z_t. The innovation term incorporates both the learned bias and the adaptive covariance, enabling robust filtering even in the presence of environmental disturbances.

4.2.3 Experiments and Results

This section provides a detailed description of the experiments conducted, along with the laboratory setup and the results obtained. The presentation begins by evaluating the NDNN applied to UWB signals, followed by the robot localization performance across several controlled scenarios. Finally, an ablation study is presented to assess the impact of key factors on localization accuracy, including the number of UWB anchors and the individual contributions of dynamic covariance and bias correction.

Experimental Setup

Fig. 4.11 The laboratory equipped with a Vicon tracking system and the Jackal rover used for the experiments. Obstacles are placed in different positions around the area to create the testing scenarios.

The experiments were carried out in a laboratory room at the PIC4SeR center. The robot operated within a rectangular area of approximately 6 m $\times$ 3 m, surrounded by UWB anchors. The platform used was a Clearpath Jackal mobile robot, with a maximum velocity of 1 m/s. Ground truth data were collected using a Vicon Tracker System, providing full-trajectory measurements. The laboratory environment is shown in Figure 4.11.

Indoor spaces inherently degrade UWB signals due to multipath reflections and distortions caused by surrounding walls and furniture such as chairs, tables, and metal structures. Metallic elements, in particular, introduce significant interference, disrupting communication between the UWB tag and anchors. For the main experiments six UWB anchors were deployed in a hexagonal configuration. In the ablation study, the number of anchors was varied from six (hexagonal) to four (square configuration).

A total of nine scenarios were designed, as shown in Figure 4.12. These scenarios were constructed to capture different forms of UWB signal corruption and evaluate the effectiveness of the proposed adaptive novelty-based filtering strategy. The first three scenarios represent nominal environmental conditions, differing from the NDNN training only in the executed trajectories.

Scenario 4 introduces a random trajectory and a fixed obstacle placed along one side of the rectangular area, causing position-dependent NLoS conditions. Scenarios 5 and 7, and scenarios 6 and 8, repeat the same trajectories but with forced NLoS created by occluding different anchors using a metal plate. Scenario 9 represents the most challenging configuration, involving simultaneously a fixed obstacle in front of one anchor and a dynamic obstacle periodically occluding another during execution.

Thus, scenarios 5–9 include time-varying NLoS conditions, a realistic challenge seldom explored in previous UWB localization literature. The intentional occlusions simulate real-world environmental changes that would require traditional filtering methods to be manually re-tuned and learning-based methods to be re-trained. In contrast, scenario 4 uses a large metal cabinet as a static obstacle whose occlusion effect depends on robot position rather than deliberate manipulation.

Each trajectory lasts approximately 50 s on average, except for scenario 4, which lasts 80 s. Overall, the scenario design aims to demonstrate that learning an environmental characterization provides a substantial advantage in UWB-based localization and that adaptive filtering can outperform static methods under previously unad-

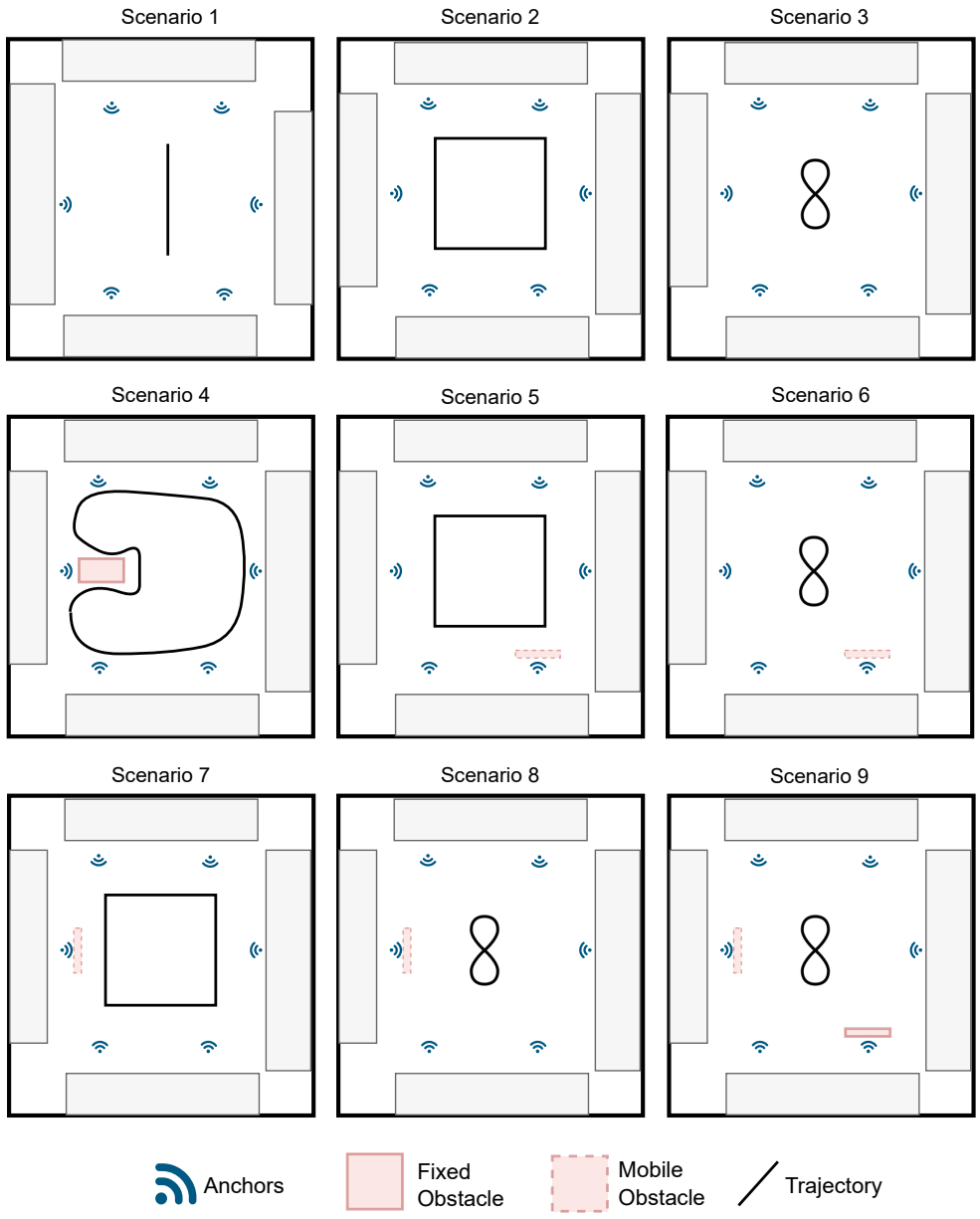


Fig. 4.12 Testing scenarios designed for this paper. The first three scenarios represent distinct trajectories conducted under LoS conditions.

dressed realistic testing conditions. Table 4.4 summarizes the characteristics of each scenario, also reporting the average novelty score associated with LoS and NLoS conditions.

Scenario	Trajectory	NLOS Condition	Average ND Scores
1	Line	LOW	0.018
2	Rectangle	LOW	0.017
3	Infinite	LOW	0.019
4	Random	HIGH	0.032
5	Rectangle	MEDIUM	0.025
6	Infinite	MEDIUM	0.028
7	Rectangle	MEDIUM	0.026
8	Infinite	MEDIUM	0.023
9	Infinite	HIGH	0.042

Table 4.4 Scenarios description with the related NLOS conditions

Novelty Detection on UWB signal

In this section, a graphical analysis of the results obtained on UWB signals using the NDNN is presented. The hyperparameters of the Neural Network were selected through a grid search over the following ranges: $E2, D1$ neurons $\in [15, 20]$, $E3$ neurons $\in [20, 30, 40]$, batch size $\in [16, 32, 64]$, and learning rate $\in [0.001, 0.01]$. The best-performing overcomplete configuration consists of $E2, D1 = 15$ neurons, $E3 = 30$ neurons, a batch size of 32, and a learning rate of 0.001. The input and output layers contain as many neurons as the number of anchors. Training is performed for only 10 epochs. The average inference time over 10 000 predictions is 29.059 ± 5.902 ms on an Intel Core i7-7500U CPU. Once converted to TFLite, the inference time drops to 0.079 ± 0.012 ms, making the model suitable for real-time deployment on standard robot CPUs.

The NDNN is trained using data collected while the robot traverses random trajectories within the rectangular region defined by the anchors. Training with such randomly generated paths offers a significant advantage over the fingerprinting approach in [50], as it greatly accelerates dataset acquisition. Instead of collecting multiple samples at each discrete point, a single measurement is obtained across

many different positions. The dataset becomes representative once the robot has sufficiently explored the entire area.

Figure 4.13 shows an example of the novelty score produced from a single anchor in scenarios 3 (nominal) and 9 (disturbed). In scenario 3, the novelty remains consistently low, whereas it increases noticeably in scenario 9 once disturbances are introduced. Some isolated outliers appear due to sporadic antenna errors that produce incorrect range measurements. Scenario 9 further highlights the effectiveness of the NDNN: an initial period of NLoS conditions (up to the 300^{th} sample) caused by an occluded anchor is followed by a larger deviation once a metal plate is positioned in front of another anchor. This second obstruction significantly degrades the signal, increasing the discrepancy between the network input and its reconstruction.

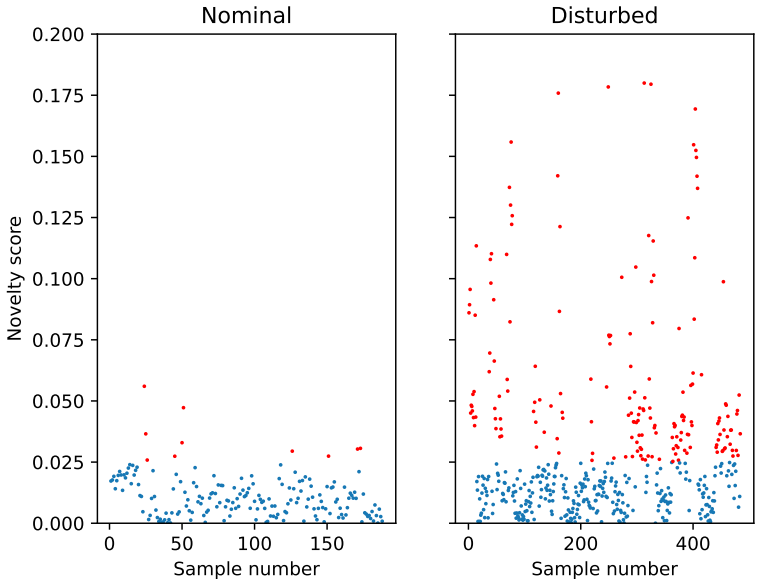


Fig. 4.13 The outputs of the novelty network. The novelty score represents the error between the network's predictions and the raw ranges fed into the network. In nominal scenario 3 (left graph), the errors are lower in all the sample points than in the disturbed scenario 9 (right graph).

The data in Fig. 4.13 also confirm the suitability of the values selected for the EKF covariance adaptation and the bias activation map shown in Fig. 4.10. Novelty scores below 0.025 (blue dots) reliably correspond to near-nominal conditions. Under these settings, scenario 3 is predominantly classified as LoS, consistent with the experimental setup, while scenario 9 correctly shows a high proportion of NLoS points due to the double occlusion. A maximum threshold of 0.1 is applied to saturate

the covariance, reflecting the increased uncertainty caused by UWB disturbances. This correction is applied independently per anchor. In addition, the learned bias activation map converts the novelty score directly into a range correction: under nominal LoS (novelty ≈ 0), an 11 cm correction is applied, while strong NLoS conditions trigger a 50 cm correction. This compensates for distortions typical of cluttered indoor spaces, where even nominal measurements can be biased due to walls or furniture.

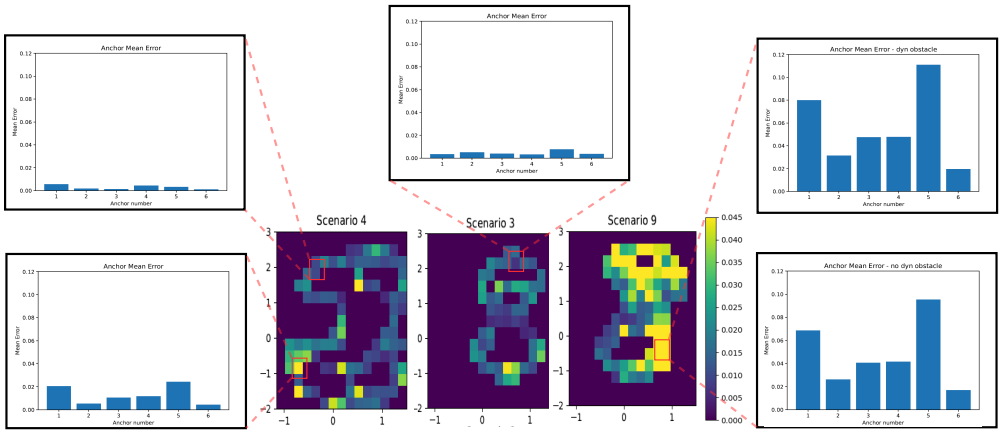


Fig. 4.14 Novelty network's scores heatmaps for scenarios 3, 4, and 9. The more yellow zones show big differences between measured ranges and nominal ones used for training.

Figure 4.14 illustrates three representative scenarios—one nominal (scenario 3) and two disturbed (scenarios 4 and 9)—to provide a clearer picture of how the proposed novelty detection framework operates. The grid map is divided into 16 cells, each covering 0.18 m in X and 0.33 m in Y. For each cell, the corresponding range measurements are averaged to compute an aggregated novelty score. In scenario 3, the heatmap is dominated by low (blue) values, reflecting consistent LoS conditions, with occasional yellow points indicating natural outliers. These outliers highlight the framework's ability to handle sporadic anomalies caused by communication drops or multipath reflections. The associated histograms provide anchor-specific insights into the average errors at selected map points.

Scenario 4 demonstrates the spatial sensitivity of the approach: certain regions of the map exhibit low novelty scores due to clear LoS conditions, while others—particularly the upper and lower left areas—show higher values caused by a metal cabinet that introduces UWB noise and NLoS conditions. This scenario confirms the central idea of the proposed method: accurate UWB localization requires

learning the environmental characteristics, and the novelty score effectively captures deviations tied to specific positions.

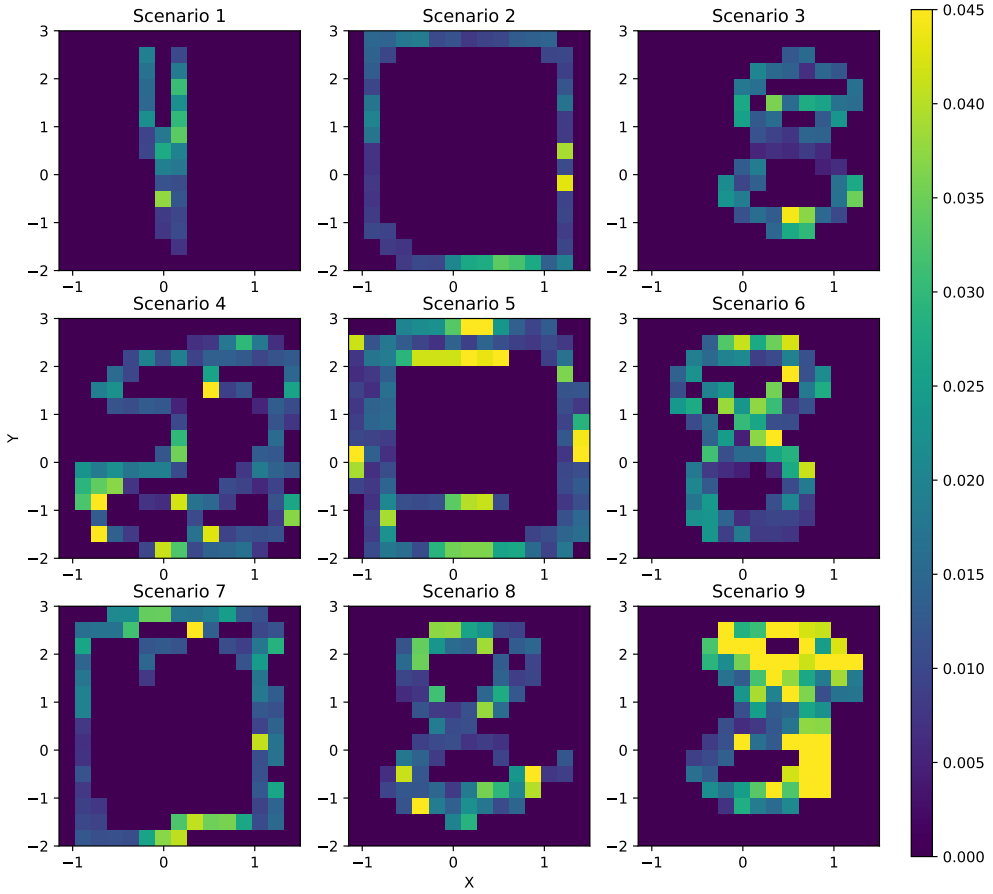


Fig. 4.15 Heatmaps for each scenario related to the novelty score

Scenario 9 is the most challenging case, as two anchors are occluded during significant portions of the trajectory, resulting in pervasive NLoS conditions. The anchor-specific histograms show how the novelty score increases over time, especially once a metal plate is placed in front of an anchor. In such severe conditions, the NDNN cannot fully reconstruct each anchor's nominal range, leading to generally higher errors across all anchors. This behavior is expected: the objective of the novelty framework is not to reconstruct precise ranges under strong NLoS, but to

reliably flag the presence and severity of NLoS so that the filter can correctly adjust the measurement uncertainty.

A summary of all test scenarios is provided in Fig. 4.15. Scenarios 1–3 remain predominantly in LoS, showing low novelty scores. Scenarios 5 and 7 exhibit similar patterns, each involving the occlusion of one anchor. Scenarios 6 and 8 also behave consistently, though with different robot trajectories. Comparing scenarios with single and double occlusions clearly highlights the increased difficulty of scenario 9, where most map locations present high novelty scores due to the two simultaneous obstructions.

Robot Localization

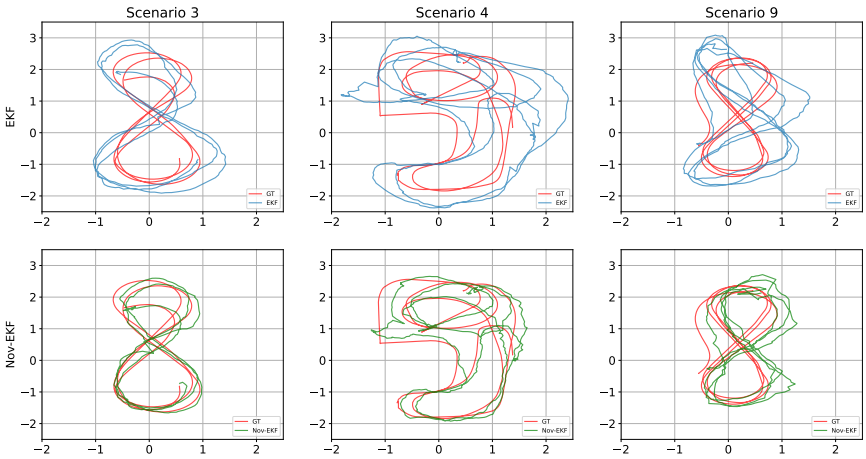


Fig. 4.16 Trajectories reconstruction of scenarios 3, 4 and 9. Standard EKF trajectory in the top row, Nov-EKF trajectories in the bottom row.

The overall localization framework was evaluated across nine experimental scenarios. In addition, the ablation study conducted at the end of this section combines these nine scenarios with three different anchor configurations, resulting in a total of 27 testing conditions. This extensive evaluation highlights the benefits of the proposed approach under a wide variety of operational settings. The nominal experimental environment corresponds to the office room shown in Fig. 4.11.

To assess the effectiveness of the Nov-EKF, a first comparison is performed against a standard EKF that employs static covariance values for the UWB measurement update. The baseline EKF uses a fixed UWB covariance of 0.01, which was

Scenario	Algorithm	RMSE_x [cm]	RMSE_y [cm]	RMSE_tot [cm]	MAE_x [cm]	MAE_y [cm]	MAE_tot [cm]
Training	EKF	33,7	23,9	41,3	29,0	18,7	38,1
	Nov-EKF	**10,4**	**8,8**	**13,6**	**9,0**	**7,0**	**12,5**
1	EKF	7,8	26,6	27,7	6,4	22,8	24,2
	Nov-EKF	**4,8**	**13,2**	**14,1**	**4,0**	**10,8**	**12,0**
2	EKF	29,0	21,8	36,3	26,7	19,3	34,6
	Nov-EKF	**6,4**	**8,2**	**10,4**	**5,4**	**6,8**	**9,4**
3	EKF	54,0	37,6	65,8	45,5	32,6	61,0
	Nov-EKF	**29,5**	**30,0**	**42,1**	**24,6**	**26,5**	**39,6**
4	EKF	53,6	31,9	62,3	45,8	26,5	57,3
	Nov-EKF	**35,3**	**25,1**	**43,3**	**27,7**	**21,4**	**38,7**
5	EKF	32,6	19,7	38,1	28,2	16,3	34,5
	Nov-EKF	**8,7**	**8,2**	**11,9**	**6,6**	**6,4**	**10,2**
6	EKF	38,9	24,0	45,7	31,6	18,9	40,6
	Nov-EKF	**12,5**	**11,1**	**16,7**	**10,4**	**8,4**	**15,2**
7	EKF	31,9	22,4	39,0	27,3	19,4	35,2
	Nov-EKF	**14,9**	**9,5**	**17,7**	**10,5**	**7,7**	**14,3**
8	EKF	39,3	23,9	46,0	32,9	18,1	40,8
	Nov-EKF	**17,1**	**10,4**	**20,0**	**13,8**	**7,6**	**17,2**
9	EKF	51,2	48,7	70,7	42,8	42,5	65,7
	Nov-EKF	**29,1**	**35,7**	**46,0**	**23,9**	**31,3**	**43,4**
Avg Test	EKF	37,2	28,0	47,3	31,6	23,5	43,2
	Nov-EKF	**16,9**	**16,0**	**23,6**	**13,6**	**13,4**	**21,2**

Table 4.5 RMSE and MAE values obtained for all the scenarios with both EKF and the proposed Nov-EKF.

found to provide the best trade-off across all scenarios. All EKF predictions run at 40 Hz, while UWB-based correction updates occur at the arrival of each range measurement packet, averaging approximately 15 Hz.

A quantitative summary of the localization performance is provided in Table 4.5. For each scenario, the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are computed by comparing the filter output against the Vicon ground-truth trajectory. As shown, the Nov-EKF consistently and significantly outperforms the static EKF across all scenarios, including both mild (soft) and severe (hard) NLoS conditions. For the majority of individual scenarios, as well as for the aggregated results reported in the bottom row of the table, the RMSE and MAE values obtained

with Nov-EKF are often reduced by more than half. These improvements demonstrate the substantial increase in accuracy and robustness achieved by incorporating novelty-driven covariance and bias adaptation into the filtering process.

To better illustrate these findings, Fig. 4.16 presents the localization trajectories obtained with the Nov-EKF and the static EKF for three representative scenarios—3, 4, and 9—previously analyzed from the signal-quality perspective. The Nov-EKF trajectories are consistently closer to the Vicon ground truth, whereas the static EKF exhibits larger deviations, particularly under NLoS conditions. The performance degradation with increasing NLoS severity is evident when comparing scenarios 3 and 9, which follow the same trajectory but differ significantly in signal quality due to the double-anchor occlusion in scenario 9. In scenario 4, the largest localization errors appear in the left region of the map, where the static obstacle introduces persistent NLoS conditions. These observations align with the novelty-score maps in Fig. 4.15 and confirm the ability of the proposed framework to adaptively account for environmental disturbances that affect UWB measurements.

State of the art comparison

A comparison between the proposed Nov-EKF method and state-of-the-art UWB positioning algorithms is presented in this section. As shown in previous work [50], Nov-EKF relies exclusively on UWB range measurements, whereas most existing solutions exploit the Channel Impulse Response (CIR) to improve localization accuracy. For this reason, the comparison on Scenarios 3, 4, and 9—those is performed in which both range and CIR data are available—allowing a fair evaluation of established baselines.

The first selected baseline is an LOS/NLOS classification model based on a CNN–LSTM network [101], one of the most widely adopted strategies for UWB error detection. It is used the same architecture proposed in the original paper, modifying only the LSTM input size to match the 157-sample CIR vector produced by our Decawave DWM1001C anchors (the original work uses 1016-sample CIRs from a Pozyx device). The classifier outputs 0 for LOS conditions and 1 for NLOS. This classifier is integrated within the same EKF framework used by Nov-EKF, assigning a low range covariance (0.001) to LOS predictions and a higher covariance (0.1) to NLOS predictions.

As a second baseline, REMnet [82] is included, a regression-based model that predicts UWB range bias. REMnet is directly applied to the test trajectories, and its estimated bias is used to correct the range measurements in the EKF. A fixed covariance of 0.01 is used during the correction step.

Scenario	Algorithm	RMSE_x [cm]	RMSE_y [cm]	RMSE_tot [cm]	MAE_x [cm]	MAE_y [cm]	MAE_tot [cm]
3	EKF	54	37,6	65,8	45,5	32,6	61
	CNNLSTM-EKF	40,5	32,1	51,6	31,8	27,4	47,0
	REMnet-EKF	42,0	34,8	54,5	35,8	30,4	51,2
	Nov-EKF (Ours)	**29,5**	**30**	**42,1**	**24,6**	**26,5**	**39,6**
4	EKF	53,6	31,9	62,3	45,8	26,5	57,3
	CNNLSTM-EKF	48,9	27,9	56,3	39,6	22,9	50,8
	REMnet-EKF	41,2	28,4	50,1	35,0	23,6	46,1
	Nov-EKF (Ours)	**35,3**	**25,1**	**43,3**	**27,7**	**21,4**	**38,7**
9	EKF	51,2	48,7	70,7	42,8	42,5	65,7
	CNNLSTM-EKF	40,0	37,4	54,8	31,1	33,6	51,0
	REMnet-EKF	40,2	45,7	60,9	33,7	39,8	56,9
	Nov-EKF (Ours)	**29,1**	**35,7**	**46**	**23,9**	**31,3**	**43,4**
Avg Test	EKF	52,9	39,4	66,3	44,7	33,9	61,3
	CNNLSTM-EKF	43,1	32,5	54,2	34,2	28,0	49,6
	REMnet-EKF	41,1	36,3	55,2	34,8	31,3	51,4
	Nov-EKF (Ours)	**31,3**	**30,3**	**43,8**	**25,4**	**26,4**	**40,6**

Table 4.6 Results of the comparison with state of the art methods: CNNLSTM-EKF based on LOS/NLOS Classification [101, 91], REMnet-EKF based on range error regression [82].

In both cases, the models are trained using the datasets provided in the original publications; our dataset is used exclusively for testing the resulting localization pipelines. A summary of the comparative results is reported in Table 4.6. The CNNLSTM-EKF classifier outperforms REMnet-EKF in LOS conditions (Scenario 3) and in severe NLOS conditions (Scenario 9), whereas the regression model obtains better performance in moderate NLOS (Scenario 4). Overall, our Nov-EKF consistently surpasses both baselines, reducing the absolute positioning error by approximately 10 cm in all considered scenarios. These results highlight the benefit of our adaptive novelty-based filtering, which provides richer information about the reliability of each UWB range measurement and improves both covariance and bias estimation. Furthermore, the method requires no labeled data, unlike both classification and regression approaches, thus eliminating the burden of costly annotation.

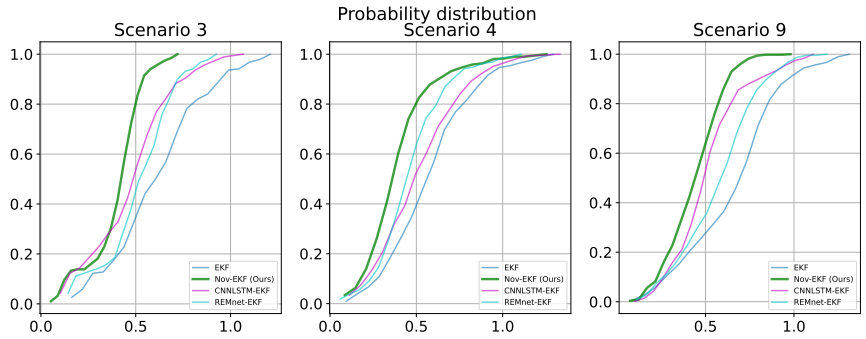


Fig. 4.17 Cumulative Density Function of the position error of the filtered signal with respect to the ground truth. Upper curves represent positioning results with lower error.

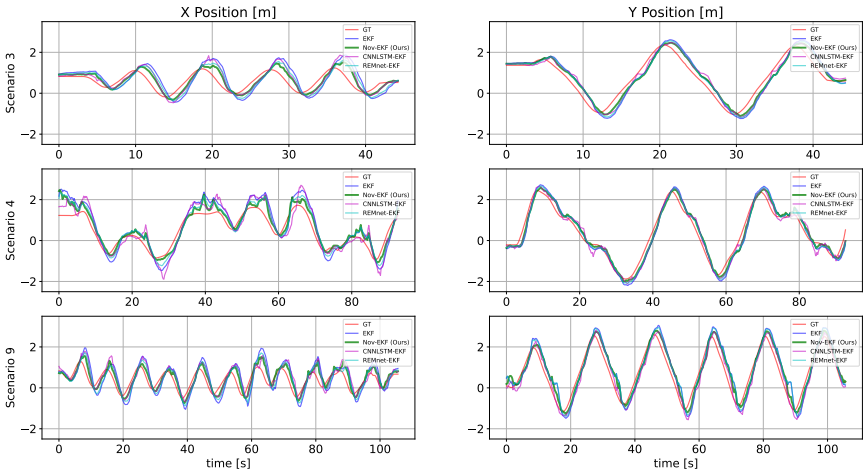


Fig. 4.18 X and Y positions comparison between ground truth, static EKF, CNNLSTM-EKF [101], REMnet-EKF [82] and the proposed Nov-EKF framework.

As an additional visual comparison, Figure 4.17 shows the Cumulative Density Functions (CDFs) of the positioning errors for the standard EKF, CNNLSTM-EKF, REMnet-EKF, and our Nov-EKF. The CDFs are computed from the errors between the corrected estimates and the ground truth. Higher CDF values correspond to lower errors, and the curves obtained with Nov-EKF dominate all others across scenarios, confirming the superior accuracy of our approach.

Finally, Figure 4.18 presents detailed plots of the X and Y coordinate estimates and their corresponding errors for the representative scenarios. The green line denotes the Nov-EKF prediction, the red line the ground truth, and the remaining curves the baseline methods. The plots clearly show the improved tracking accuracy of Nov-EKF, which closely matches the ground truth and significantly reduces

overshoot during turning maneuvers. This improvement is primarily due to the adaptive covariance and bias adjustments applied to UWB range measurements in the correction step, which represent the most influential source of information within the EKF.

Ablation study

In this section, an ablation study is presented to provide an in-depth assessment of the proposed Nov-EKF system. The objective is to quantify the influence of key parameters and design choices on the overall performance of the framework. To this end, 27 experiments are conducted considering configurations with 4, 5, and 6 anchors across all nine scenarios. The robot trajectories and obstacle layouts remain unchanged for the reduced-anchor configurations: the side anchors shown in Figure 4.12 are removed, converting the original hexagonal arrangement into a four-anchor square layout.

This ablation study focuses on isolating and evaluating the contribution of two main components:

- **Number of anchors**: the system's behavior is analyzed as the number of deployed anchors decreases, simulating increasingly challenging conditions. With only four anchors, frequent NLoS conditions may occur, degrading trilateration accuracy and leading to localization failures.
- **Bias and covariance estimation**: as previously described, the adaptive Nov-EKF estimates both a bias term and a covariance for each UWB range measurement in the EKF correction step. In this ablation, three variants of the pipeline are compared: (i) a standard EKF with static covariance, (ii) the adaptive Nov-EKF without bias estimation, and (iii) the full Nov-EKF with both covariance and bias estimation. The bias activation mapping is inherently more challenging to design from the novelty score alone—often requiring ground-truth information to tune the polynomial coefficients—thus motivating a dedicated analysis of its contribution.

Figure 4.19 summarizes the ablation results for all nine scenarios. Each subplot reports the average RMSE with respect to ground truth for the three compared methods: the static EKF (green), the adaptive Nov-EKF without bias estimation (orange),

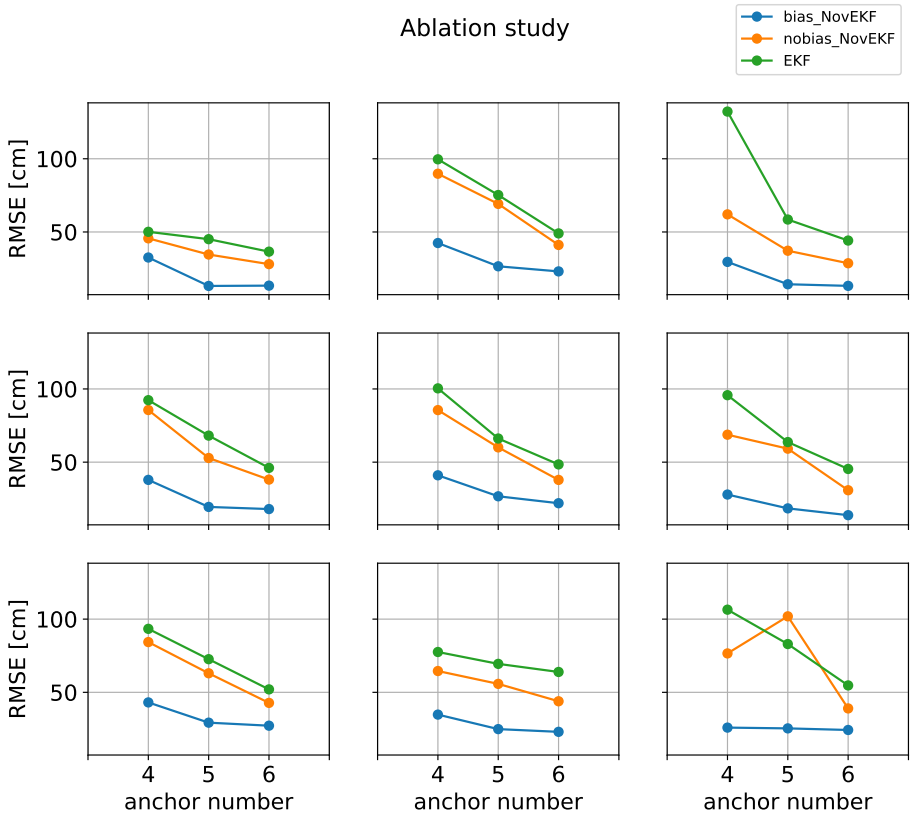


Fig. 4.19 RMSE metrics of the ablation study, comparing bias and nobias correction and analyzing the dependency with the number of anchors.

and the full Nov-EKF (blue). Overall, the adaptive Nov-EKF consistently improves accuracy even when only the covariance is estimated. However, incorporating the bias term provides a substantial additional performance boost across all scenarios, particularly when the system operates with fewer anchors. This improvement is expected since the novelty score effectively represents a rescaled estimate of the range error; even in low-novelty conditions, cluttered indoor environments introduce reflections and disturbances that lead to non-negligible baseline bias. Therefore, applying an adaptive range correction term remains beneficial for the filter.

Reducing the number of anchors from 6 to 5 or 4 has a considerable impact on the baseline methods. Both the static EKF and the Nov-EKF without bias suffer from reduced redundancy in trilateration, leading to degraded prediction accuracy. In contrast, the full Nov-EKF demonstrates substantially more stable behavior, showing minimal differences between the 5- and 6-anchor configurations across all scenarios.

This indicates that the adaptive mechanisms embedded in Nov-EKF compensate for the reduced geometric diversity.

In general, a sufficiently high number of anchors is necessary to maintain robustness in NLoS-prone settings. Among all scenarios, only Scenarios 1 and 8 exhibit limited sensitivity to anchor count, suggesting that at least a minimal set of anchors remains reliably in LoS throughout the trajectory.

A particularly noteworthy behavior appears in Scenario 9, where severe NLoS conditions arise after the occlusion of two anchors. The full Nov-EKF maintains stable performance even with only four anchors, achieving significant improvements over the classic EKF despite having just two anchors in LoS. Conversely, the five-anchor configuration without bias correction performs unexpectedly poorly. This likely results from an overly generalized covariance estimation under severe disturbances, preventing the system from distinguishing clean from corrupted range measurements on the affected anchors.

4.2.4 Conclusions and Future Works

This paper advances robot localization with UWB technology by introducing an adaptive EKF–novelty framework. The proposed method effectively addresses the challenge of unreliable UWB measurements in cluttered indoor environments with frequent NLoS conditions, achieving an average improvement of approximately 22 cm in absolute positioning accuracy.

The key contribution lies in enabling the EKF to dynamically evaluate the reliability of incoming range measurements through covariance and bias terms inferred from the novelty score. Experimental results highlight the substantial reduction in localization error compared to a standard EKF baseline. Furthermore, the ablation study provides insight into the influence of the number of UWB anchors and the role of bias estimation, revealing how these factors can significantly enhance or degrade the robustness of the system.

A limitation of the proposed framework concerns its generalization capability: the learned nominal data distributions are tied to fixed anchor placements. Consequently, when the system is deployed in a new environment or when anchor positions change significantly, retraining is required to establish new nominal conditions.

However, this limitation is mitigated by the fast and label-free training process of the unsupervised novelty autoencoder.

Future work will explore online or continual training strategies to enable the model to automatically adapt to changing environments. Additionally, the EKF may be replaced with more advanced tracking approaches, such as factor-graph optimization or neural architectures that integrate extended temporal sequences, addressing the EKF's limited prediction horizon. We also plan to investigate sensor fusion strategies and generative modeling techniques to further improve the estimation and predictive capabilities of the novelty score.

Chapter 5

Conclusion

The works presented in this thesis investigate different applications in which novelty detection represents the core paradigm underlying each proposed algorithm. The first part of the thesis focuses on industrial environments, with particular attention to bearing, which constitutes the main object of study.

The first work demonstrates that novelty detection can be effectively applied using relatively simple models, such as a Linear Regressor, by exploiting the concept of overfitting as an advantage rather than a limitation. When properly trained, even a simple linear model can successfully identify deviations from normal operating conditions, thus performing novelty detection in an effective manner.

An important aspect of the first two works presented in this thesis is the complementary analysis of supervised and unsupervised machine learning approaches. The first study adopts a supervised learning framework, while the second focuses exclusively on unsupervised machine learning models. Together, these contributions provide a broad overview of novelty detection techniques based on both supervised and unsupervised paradigms. In particular, the second work proposes a benchmark for novelty detection using only unsupervised methods. Although the two studies employ different preprocessing strategies, the results obtained are highly promising, especially in scenarios characterized by limited data availability.

The second part of the thesis is devoted to the application of novelty detection in the robotics domain. In this context, a deep learning approach is adopted, with autoencoders serving as the primary models for novelty detection. The first work in this section introduces the overall localization framework, laying the foundation for

the complete pipeline. The subsequent work extends this framework by integrating a Kalman Filter, resulting in a complete and practical localization solution that can be readily adopted by practitioners.

The central contribution of this thesis is the demonstration that novelty detection represents a flexible and easily deployable paradigm, even in complex systems operating under constrained computational resources. Despite its effectiveness, novelty detection is still not widely adopted in industrial environments. This thesis aims to bridge that gap by proposing a preliminary framework that can complement traditional Predictive Maintenance strategies, acting as an intermediate layer capable of detecting anomalies and improving the understanding of machine-generated signals that are often underutilized.

As demonstrated throughout this work, novelty detection performs effectively across multiple domains thanks to its adaptability to different types of sensors and data sources. Moreover, the use of simple and computationally efficient models shows that even non-expert users can integrate this paradigm to begin extracting meaningful insights from machine data, thereby leveraging the full potential of multi-sensor information.

Chapter 6

Future Works

Future work will explore several research directions in which novelty detection remains the central paradigm. One promising avenue involves the application of novelty detection within the context of neuromorphic computing, a rapidly emerging domain in artificial intelligence. Neuromorphic systems convert signals from the physical domain into spiking representations, which can be processed using spiking neural networks specifically designed to handle event-based data. The primary advantage of neuromorphic computing lies in its high computational efficiency, enabled by massive parallelism during inference. This capability allows such systems to perform inference more rapidly than conventional architectures while consuming significantly less power. As a result, neuromorphic approaches are particularly well suited for deployment on embedded devices, fully exploiting the benefits of edge computing.

Another application domain of interest for novelty detection is agriculture. In this context, the goal is to monitor plant health and detect changes over time using visual data acquired through cameras. In addition to standard imaging systems, emerging multispectral and hyperspectral cameras can capture information beyond the visible spectrum, enabling the extraction of vegetation indices that are highly informative for plant analysis. A major challenge in this sector is the scarcity of ground-truth data for conditions such as plant diseases, hydration levels, and stress factors. Novelty detection techniques, trained on data representing normal plant conditions, could therefore play a crucial role in identifying abnormal behaviors and supporting the

incremental construction of more comprehensive datasets for precise plant health monitoring.

References

[1] Laura Swanson. Linking maintenance strategies to performance. *International journal of production economics*, 70(3):237–244, 2001.

[2] Umberto Albertin, Giuseppe Pedone, Matilde Brossa, Giovanni Squillero, and Marcello Chiaberge. A real-time novelty recognition framework based on machine learning for fault detection. *Algorithms*, 16(2), 2023.

[3] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[4] Roza Dastres Mohsen Soori, Behrooz Arezoo. Artificial intelligence, machine learning and deep learning in advanced robotics, a review. *Cognitive Robotics*, Volume 3, 2023, Pages 54-70.

[5] Syed Mohtashim Mian, Mohammad Shuaib Khan, Mohd Shawez, and Amandeep Kaur. Artificial intelligence (ai), machine learning (ml) deep learning (dl): A comprehensive overview on techniques, applications and research directions. pages 1404–1409, 2024.

[6] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep learning*.

[7] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[8] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[9] F Rosenblatt. Principles of neurodynamics (spartan, new york). *Principles of Neurodynamics*, 1962.

[10] B. Widrow and M. E Hoff. Adaptive switching circuits. *1960 IRE WESCON Convention Record*, 4:96–104.

[11] M Minsky and S Papert. Perceptrons," mit press, cambridge, ma, 1969.

[12] Bruno A Olshausen and David J Field. How close are we to understanding v1? *Neural computation*, 17(8):1665–1699, 2005.

[13] Laurie Von Melchner, Sarah L Pallas, and Mriganka Sur. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, 404(6780):871–876, 2000.

[14] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.

[15] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[17] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[18] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[19] David S Touretzky and Geoffrey E Hinton. Symbols among the neurons: Details of a connectionist inference architecture. In *IJCAI*, volume 85, pages 238–243, 1985.

[20] Donald Hebb. The organization of behavior. emphnew york, 1949.

[21] Geoffrey E Hinton. Distributed representations. 1984.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[23] Yann LeCun. Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models). 1987.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[25] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[26] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[27] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[28] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961.

[29] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. 1985.

[31] Umberto Albertin, Ariel Priarone, Carlo Cena, Mauro Martini, and Marcello Chiaberge. Unsupervised novelty detection methods benchmarking with wavelet decomposition. In *2024 8th International Conference on System Reliability and Safety (ICSRS)*, pages 198–207, 2024.

[32] Industry Week Emerson. How Manufacturers Achieve Top Quartile Performance. Available online: https://partners.wsj.com/emerson/unlocking-performance/how-manufacturers-can-achieve-top-quartile-performance/ (accessed on 31 May 2017).

[33] Deloitte. Predictive Maintenance and the Smart Factory. 2021. Available online: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/process-and-operations/us-cons-predictive-maintenance.pdf (accessed on 10 June 2021).

[34] Hitoshi Tsunashima. Condition monitoring of railway tracks from car-body vibration using a machine learning technique. *Applied Sciences*, 9(13):2734, 2019.

[35] Adrian Stetco, Fateme Dinmohammadi, Xingyu Zhao, Valentin Robu, David Flynn, Mike Barnes, John Keane, and Goran Nenadic. Machine learning methods for wind turbine condition monitoring: A review. *Renewable energy*, 133:620–635, 2019.

[36] Rosmaini Ahmad and Shahrul Kamaruddin. An overview of time-based and condition-based maintenance in industrial application. *Computers & industrial engineering*, 63(1):135–149, 2012.

[37] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, and Dong-Hua Zhou. Remaining useful life estimation–a review on the statistical data driven approaches. *European journal of operational research*, 213(1):1–14, 2011.

[38] David He, Ruoyu Li, and Junda Zhu. Plastic bearing fault diagnosis based on a two-step data mining approach. *IEEE Transactions on Industrial Electronics*, 60(8):3429–3440, 2012.

[39] Jinjiang Wang, Shaopeng Liu, Robert X Gao, and Ruqiang Yan. Current envelope analysis for defect identification and diagnosis in induction motors. *Journal of Manufacturing Systems*, 31(4):380–387, 2012.

[40] Francesco Del Buono, Francesca Calabrese, Andrea Baraldi, Matteo Paganelli, and Francesco Guerra. Novelty detection with autoencoders for system health monitoring in industrial environments. *Applied Sciences*, 12(10):4931, 2022.

[41] Clauber Gomes Bezerra, Bruno Sielly Jales Costa, Luiz Affonso Guedes, and Plamen Parvanov Angelov. An evolving approach to unsupervised and real-time fault detection in industrial processes. *Expert Systems with Applications*, 63:134–144, 2016.

[42] Moisés A Oliveira, Eduardo F Simas Filho, Maria CS Albuquerque, Ygor TB Santos, Ivan C Da Silva, and Cláudia TT Farias. Ultrasound-based identification of damage in wind turbine blades using novelty detection. *Ultrasonics*, 108:106166, 2020.

[43] Hai Qiu, Jay Lee, Jing Lin, and Gang Yu. Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics. *Journal of sound and vibration*, 289(4-5):1066–1090, 2006.

[44] Adrián Vega, Ignacio Crespo-Martínez, Ángel Guerrero-Higueras, Claudia Álvarez Aparicio, Vicente Matellán, and Camino Fernández. Malicious traffic detection on sampled network flow data with novelty-detection-based models. *Scientific Reports*, 13, 09 2023.

[45] Jeffrey Schein, Steven T Bushby, Natascha S Castro, and John M House. A rule-based fault detection method for air handling units. *Energy and buildings*, 38(12):1485–1492, 2006.

[46] Rolf Isermann. Model-based fault-detection and diagnosis–status and applications. *Annual Reviews in control*, 29(1):71–85, 2005.

[47] Jihoon Chung, Bo Shen, and Zhenyu James Kong. Anomaly detection in additive manufacturing processes using supervised classification with imbalanced sensor data based on generative adversarial network. *J. Intell. Manuf.*, 35(5):2387–2406, 6 2023.

[48] Fa Zhu, Wenjie Zhang, Xingchi Chen, Xizhan Gao, and Ning Ye. Large margin distribution multi-class supervised novelty detection. *Expert Systems with Applications*, 224:119937, 2023.

[49] Carlo Cena, Umberto Albertin, Mauro Martini, Silvia Bucci, and Marcello Chiaberge. Physics-Informed Real NVP for Satellite Power System Fault Detection. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2024.

[50] Umberto Albertin, Alessandro Navone, Mauro Martini, and Marcello Chiaberge. Semi-supervised novelty detection for precise ultra-wideband error signal prediction. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 3719–3724, 2024.

[51] Mohammad Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M. Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):859–874, 2011.

[52] Mohammed Chalouli, Nasr-eddine Berrached, and Mouloud Denai. Intelligent health monitoring of machine bearings based on feature extraction. *Journal of Failure Analysis and Prevention*, 17(5):1053–1066, 10 2017.

[53] Yuyan Zhang, Xinyu Li, Liang Gao, and Peigen Li. A new subset based deep feature learning method for intelligent fault diagnosis of bearing. *Expert Systems with Applications*, 110:125–142, 2018.

[54] Qicai Zhou, Hehong Shen, Jiong Zhao, Xingchen Liu, and Xiaolei Xiong. Degradation state recognition of rolling bearing based on k-means and cnn algorithm. *Shock and Vibration*, 2019(1):8471732, 2019.

[55] Luis A. Pinedo-Sánchez, Diego A. Mercado-Ravell, and Carlos A. Carballo-Monsivais. Vibration analysis in bearings for failure prevention using cnn. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 42(12):628, 11 2020.

[56] Cecilia Gattino, Elia Ottonello, Mario Baggetta, Roberto Razzoli, Jacek Stecki, and Giovanni Berselli. Application of ai failure identification techniques in condition monitoring using wavelet analysis. *The International Journal of Advanced Manufacturing Technology*, 125(9):4013–4026, 04 2023.

[57] J. Lee, H. Qiu, G. Yu, J. Lin, and Rexnord Technical Services. Bearing data set. IMS, University of Cincinnati, NASA Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA, 2007.

[58] Case Western Reserve University. Bearing data center: Seeded fault test data. http://data-acquisition.case.edu/. Accessed: 2024-06-05.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[60] Lanjun Wan, Gen Zhang, Hongyang Li, and Changyun Li. A novel bearing fault diagnosis method using spark-based parallel aco-k-means clustering algorithm. *IEEE Access*, 9:28753–28768, 2021.

[61] Chen Lu, Yang Wang, Minvydas Ragulskis, and Yujie Cheng. Fault diagnosis for rotating machinery: A method based on image processing. *PLOS ONE*, 11(10):1–22, 10 2016.

[62] Jakub Spytek, Adam Machynia, Kajetan Dziedziech, Ziemowit Dworakowski, and Krzysztof Holak. Novelty detection approach for the monitoring of structural vibrations using vision-based mean frequency maps. *Mechanical Systems and Signal Processing*, 185:109823, 2023.

[63] Markus Breunig, Peer Kröger, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000.

[64] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9):1641–1650, 2003.

[65] David A. Clifton, Peter R. Bannister, and Lionel Tarassenko. Learning shape for jet engine novelty detection. In Jun Wang, Zhang Yi, Jacek M. Zurada, Bao-Liang Lu, and Hujun Yin, editors, *Advances in Neural Networks - ISNN 2006*, pages 828–835, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[66] Kemilly Dearo Garcia, Mannes Poel, Joost N. Kok, and André C. P. L. F. de Carvalho. Online clustering for novelty detection and concept drift in data streams. In *Progress in Artificial Intelligence*, pages 448–459, Cham, 2019. Springer International Publishing.

[67] Andrea Eirale, Mauro Martini, and Marcello Chiaberge. Human-centered navigation and person-following with omnidirectional robot for indoor assistance and monitoring. *Robotics*, 11(5):108, 2022.

[68] Christian Tamantini, Francesco Scotto di Luzio, Francesca Cordella, Giuseppe Pascarella, Felice Eugenio Agro, and Loredana Zollo. A robotic healthcare assistant for covid-19 emergency: A proposed solution for logistics and disinfection in a hospital environment. *IEEE Robotics & Automation Magazine*, 28(1):71–81, 2021.

[69] Alessandro Navone, Mauro Martini, Andrea Ostuni, Simone Angarano, and Marcello Chiaberge. Autonomous navigation in rows of trees and high crops with deep semantic segmentation. In *2023 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2023.

[70] Mauro Martini, Simone Cerrato, Francesco Salvetti, Simone Angarano, and Marcello Chiaberge. Position-agnostic autonomous navigation in vineyards with deep reinforcement learning. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 477–484, 2022.

[71] Christian Gehring, Péter Fankhauser, Linus Isler, Remo Diethelm, Samuel Bachmann, Marcel Potz, Lars Gerstenberg, and Marco Hutter. Anymal in the field: Solving industrial inspection of an offshore hvdc platform with a quadrupedal robot. In *Field and Service Robotics: Results of the 12th International Conference*, pages 247–260. Springer, 2021.

[72] Mary B Alatise and Gerhard P Hancke. A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, 8:39830–39846, 2020.

[73] Alessandro Navone, Mauro Martini, Simone Angarano, and Marcello Chiaberge. Online learning of wheel odometry correction for mobile robots with attention-based neural network. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–6, 2023.

[74] Ke Wang, Sai Ma, Junlan Chen, Fan Ren, and Jianbo Lu. Approaches, challenges, and applications for deep visual odometry: Toward complicated and emerging areas. *IEEE Transactions on Cognitive and Developmental Systems*, 14(1):35–49, 2020.

[75] Surbhi Gupta, R Sangeeta, Ravi Shankar Mishra, Gaurav Singal, Tapas Badal, and Deepak Garg. Corridor segmentation for automatic robot navigation in indoor environment using edge devices. *Computer Networks*, 178:107374, 2020.

[76] Mahmoud Elsanhoury, Petteri Mäkelä, Janne Koljonen, Petri Välisuo, Ahm Shamsuzzoha, Timo Mantere, Mohammed Elmusrati, and Heidi Kuusniemi. Precision positioning for smart logistics using ultra-wideband technology-based indoor navigation: A review. *IEEE Access*, 10:44413–44445, 2022.

[77] Sinan Gezici, Zhi Tian, Georgios B Giannakis, Hisashi Kobayashi, Andreas F Molisch, H Vincent Poor, and Zafer Sahinoglu. Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks. *IEEE signal processing magazine*, 22(4):70–84, 2005.

[78] Daquan Feng, Chunqi Wang, Chunlong He, Yuan Zhuang, and Xiang-Gen Xia. Kalman-filter-based integration of imu and uwb for high-accuracy indoor positioning and navigation. *IEEE Internet of Things Journal*, 7(4):3133–3146, 2020.

[79] Panagiotis T Karfakis, Micael S Couceiro, David Portugal, and Rui Cortesão. Uwb aided mobile robot localization with neural networks and the ekf. In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 93–99. IEEE, 2022.

[80] Klemen Bregar and Mihael Mohorčič. Improving indoor localization using convolutional neural networks on computationally restricted devices. *IEEE Access*, 6:17429–17441, 2018.

[81] Fuhu Che, Qasim Zeeshan Ahmed, Jaron Fontaine, Ben Van Herbruggen, Adnan Shahid, Eli De Poorter, and Pavlos I Lazaridis. Feature-based generalized gaussian distribution method for nlos detection in ultra-wideband (uwb) indoor positioning system. *IEEE Sensors Journal*, 22(19):18726–18739, 2022.

[82] Simone Angarano, Vittorio Mazzia, Francesco Salvetti, Giovanni Fantin, and Marcello Chiaberge. Robust ultra-wideband range error mitigation with deep learning at the edge. *Engineering Applications of Artificial Intelligence*, 102:104278, 2021.

[83] Abhishek Goudar and Angela P Schoellig. Online spatio-temporal calibration of tightly-coupled ultrawideband-aided inertial localization. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1161–1168. IEEE, 2021.

[84] Changqiang Wang, Aigong Xu, Jian Kuang, Xin Sui, Yushi Hao, and Xiaoji Niu. A high-accuracy indoor localization system and applications based on tightly coupled uwb/ins/floor map integration. *IEEE Sensors Journal*, 21(16):18166–18177, 2021.

[85] Arne Niitsoo, Thorsten Edelhäußer, Ernst Eberlein, Niels Hadaschik, and Christopher Mutschler. A deep learning approach to position estimation from channel impulse responses. *Sensors*, 19(5), 2019.

[86] Christian Gruhl, Bernhard Sick, and Sven Tomforde. Novelty detection in continuously changing environments. *Future Generation Computer Systems*, 114:138–154, 2021.

[87] Yuanyuan Zhao, Huijuan Hao, Yu Chen, and Yu Zhang. Novelty detection and fault diagnosis method for bearing faults based on the hybrid deep autoencoder network. *Electronics*, 12(13):2826, 2023.

[88] Maximilian Stahlke, Sebastian Kram, Felix Ott, Tobias Feigl, and Christopher Mutschler. Estimating toa reliability with variational autoencoders. *IEEE Sensors Journal*, 22(6):5133–5140, 2022.

[89] Jaron Fontaine, Matteo Ridolfi, Ben Van Herbruggen, Adnan Shahid, and Eli De Poorter. Edge inference for uwb ranging error correction using autoencoders. *IEEE Access*, 8:139143–139155, 2020.

[90] Fazeelat Mazhar, Muhammad Gufran Khan, and Benny Sällberg. Precise indoor positioning using uwb: A review of methods, algorithms and implementations. *Wireless Personal Communications*, 97(3):4467–4491, 2017.

[91] Alwin Poulose, Žiga Emeršič, Odongo Steven Eyobu, and Dong Seog Han. An accurate indoor user position estimator for multiple anchor uwb localization. In *2020 international conference on information and communication technology convergence (ICTC)*, pages 478–482. IEEE, 2020.

[92] Zahid Farid, Rosdiadee Nordin, and Mahamod Ismail. Recent advances in wireless indoor localization techniques and system. *Journal of Computer Networks and Communications*, 2013(1):185138, 2013.

[93] Juan Antonio Corrales, Francisco A Candelas, and Fernando Torres. Hybrid tracking of human operators using imu/uwb data fusion by a kalman filter. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 193–200, 2008.

[94] Alessandro Benini, Adriano Mancini, and Sauro Longhi. An imu/uwb/vision-based extended kalman filter for mini-uav localization in indoor environment using 802.15. 4a wireless sensor network. *Journal of Intelligent & Robotic Systems*, 70:461–476, 2013.

[95] B Venkata Krishnaveni, K Suresh Reddy, and P Ramana Reddy. Indoor tracking by adding imu and uwb using unscented kalman filter. *Wireless Personal Communications*, 123(4):3575–3596, 2022.

[96] Xiaotong Ye and Yu Zhang. Research on uwb positioning method based on deep learning. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, pages 1505–1508. IEEE, 2020.

[97] Alwin Poulose and Dong Seog Han. Feature-based deep lstm network for indoor localization using uwb measurements. In *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 298–301. IEEE, 2021.

[98] Alwin Poulose and D.S. Han. Uwb indoor localization using deep learning lstm networks. *Applied Sciences*, 10(18):6290, 2020.

[99] Alwin Poulose, Odongo Steven Eyobu, Myeongjin Kim, and Dong Seog Han. Localization error analysis of indoor positioning system based on uwb measurements. In *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 84–88, 2019.

[100] Doan Tan Anh Nguyen, Han-Gyeol Lee, Jingon Joung, and Eui-Rim Jeong. Convolutional neural network-based uwb system localization. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 488–490. IEEE, 2020.

[101] Changhui Jiang, Jichun Shen, Shuai Chen, Yuwei Chen, Di Liu, and Yuming Bo. Uwb nlos/los classification using deep learning method. *IEEE Communications Letters*, 24(10):2226–2230, 2020.

[102] Justin Cano, Yi Ding, Gaël Pages, Eric Chaumette, and Jerome Le Ny. A robust kalman filter based approach for indoor robot positionning with multipath contaminated uwb data. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.