Internet of Things 23 (2023) 100829

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

IoT-based expert system for fault detection in Japanese Plum leaf-turgor pressure WSN☆

Arturo Barriga [a,*], José A. Barriga [a], María José Moñino [b], Pedro J. Clemente [a]

[a] Quercus Software Engineering Group[1], Department of Computer Science, Universidad de Extremadura[2], Avenida de la Universidad s/n, Cáceres (10003), Spain

[b] Area of Agronomy of Woody and Horticultural Crops, Extremadura Scientific and Technological Research Centre (CICYTEX), Guadajira (06187), Badajoz, Spain

ARTICLE INFO

Keywords:
Internet of things
Leaf-turgor pressure sensors
Machine learning
Precision agriculture
Sensor faults

ABSTRACT

Industry 4.0 involves the digital transformation of industrial sectors. Given the current climate change scenario and the scarcity of water in semi-arid regions, this digital transformation has to take into account the sustainable use of water. In agriculture, one of the most water-intensive sectors, to optimise the use of water, precision irrigation techniques are being applied. As a result of the digital transformation of agriculture, a key aspect for the application of these precision irrigation techniques, the crop water stress, can be predicted from a Wireless Sensor Network (WSN) of leaf-turgor pressure sensors. However, these sensors often fail, introducing errors in the data, which could lead to inaccurate application of precision irrigation techniques compromising crops and yields. So, sensor fault identification is a must. Nevertheless, sensor fault identification is a tedious and costly task that requires an expert to manually review all sensors and each of their measurements over the last 24 h. In this work, with the aim of digitally transforming this task, an IoT-based expert system is proposed. By means of a novel learning model, this system is capable of identifying sensor faults with 84.2% f1-score and 0.94 AUC ROC. Note that to train this learning model, only real-world data gathered from an experimental plot has been used. In addition, the real-world application of the IoT-based expert system in this plot is shown and discussed. Furthermore, a novel methodology that summarises the main findings and techniques applied in this study is also illustrated.

1. Introduction

Industry 4.0 involves the digital transformation of different industrial sectors, such as livestock [1], agriculture [2], automotive [3], construction [4], etc. Nevertheless, given the current climate change scenario, and the scarcity of water in semi-arid regions, the digital transformation has to take into account the sustainable use of water [5].

☆ This work was funded by project TED2021-129194B-I00 funded by MCIN/AEI/10.13039/501100011033 and for European Union NextGenerationEU/PRTR; the Government of Extremadura, Council for Economy, Science and Digital Agenda under the grant GR21133 and the projects IB20058, and by the European Regional Development Fund (ERDF).

* Corresponding author.
E-mail addresses: arturobc@unex.es (A. Barriga), jose@unex.es (J.A. Barriga), mariajose.monino@juntaex.es (M.J. Moñino), pjclemente@unex.es (P.J. Clemente).

[1] http://quercusseg.unex.es

[2] https://ror.org/0174shg90

https://doi.org/10.1016/j.iot.2023.100829
Received 17 April 2023; Received in revised form 17 May 2023; Accepted 22 May 2023
Available online 30 May 2023

2542-6605/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

In agriculture, the sustainable use of water is key [6]. Indeed, in countries such as Spain agriculture is the sector with the greatest water consumption, accounting for more than 70% of water extractions from rivers, reservoirs and aquifers [7]. Furthermore, water is a limiting resource in semi-arid regions, which together with the current climate change scenario, is fostering a context of uncertainty and major challenges concerning the sustainability and viability of existing agroecosystems. However, water scarcity can be mitigated by its sustainable use through the study and application of precision irrigation techniques. In this regard, techniques such as deficit irrigation techniques allow obtaining the maximum yield of a crop from the available water [8].

To apply these precision irrigation techniques, it is necessary to have a precise knowledge of the water stress of the crops. In this regard, crop water stress identification is a process that has traditionally been carried out manually, but it can be digitally transformed and automated by means of the Internet of Things (IoT) and Machine Learning (ML). In this sense, the authors of [9] present a solution based on a WSN of leaf-turgor pressure sensors and a ML model able to identify the water stress of crops from the readings of these leaf-turgor pressure sensors.

Nevertheless, since the solution described above is based on an IoT system, it has some limitations that are inherent to several IoT systems, such as the need of handling sensor faults. Furthermore, in the context of precision irrigation, the identification of sensor faults is a critical task since their readings could have a negative impact on the precision irrigation systems that use them as inputs, thus compromising crop yields.

Currently, the identification of leaf-turgor pressure sensor faults is carried out by experts by hand and involves daily analysis of the data gathered by each sensor over the last 24 h. This process is tedious, time-consuming and costly, particularly when the crop is extensive, thus limiting the scalability and feasibility of proposals such as [9]. For these reasons, this communication proposes an IoT-based expert system able of identifying leaf-turgor pressure sensor faults, thus automating and digitally transforming this process.

To carry out this proposal, a learning model able to identify if a sensor is failing is trained by means of a set of experiments. Note that this model has been trained from the leaf-turgor pressure data gathered by sensors during one year's fruit ripening cycle.

In this sense, with the aim of giving a practical approach to this communication, this model has been integrated into an IoT system, illustrating its real-world application. The proposed IoT system is designed bearing in mind the context where it could be deployed, thus addressing some of the open challenges that IoT is currently facing, such as *interoperability*, *energy awareness* or *scalability* [10].

Finally, to the best of our knowledge, most of the existing works in the literature concerning the identification of sensor faults are based on the introduction of artificially generated faults (in healthy datasets) [11]. Since the training of the learning model presented in this proposal uses only real data, this communication also shows a real-world application of the findings on sensor faults identification of these proposals. Thus, showing the value of these findings in the identification of leaf-turgor pressure sensor faults.

The main contributions of this communication are listed below:

- A novel ML model able to identify leaf-turgor pressure sensor faults. This model presents an 84.2% f1-score and a 0.94 Area Under the Curve (AUC) Receiver Operating Characteristic (ROC), being feasible to replace experts in terms of leaf-turgor pressure sensor fault identification.
- A novel methodology that summarises the main findings and techniques applied in this study. This methodology could be easily extrapolated to similar problems involving sensors and time series, thus reducing development efforts and costs.
- An IoT-based expert system that integrates the trained learning model, illustrating its real-world application. Furthermore, this IoT-based expert system is designed bearing in mind aspects such as interoperability, energy awareness and scalability.
- A real-world application of the findings of related work on sensor fault detection, assessing its effectiveness in identifying faults in a leaf-turgor pressure WSN.

The rest of the communication is structured as follows: Section 2 analyses several related works. Section 3 presents the materials and methods applied to train and validate the learning models. Section 4 illustrates the results and experiments carried out. Section 5 proposes a methodology to address similar problems that involve sensors and time series. Section 6 presents the integration of the learning model in a real IoT system. Section 7 conducts a discussion regarding the findings of this paper. Finally, Section 8 considers some future works and concludes the paper.

2. Related works

Sensor fault detection is a key process when dealing with WSNs. Particularly in critical applications such as military operations [12], disaster prediction and management [13], intruder detection [14], elderly monitoring [15] or vital signs monitoring in hospitalised patients [16]. Thus, the improvement or automation of sensor fault detection has been a subject of study over time.

In 2006 and 2009 respectively, two proposals with a major impact on the automatic detection of sensor faults emerged [17,18]. These two proposals are based on algorithms known as *Neighbours-based* algorithms. Neighbours-based algorithms assume that, in general, the nodes of a WSN are close together in space and there is a relation between sensor data from devices in a geographically close proximity. Moreover, the data obtained at an instant of time is related with previous and succeeding measurements. Thus, Neighbours-based algorithms use these spatial–temporal correlations to detect outliers and anomalous behaviour [19].

Thus, in [17] a distributed neighbour-based algorithm to identify faulty sensors is proposed and assessed. The execution of this algorithm is as follows: first, labels all nodes in the network as possibly correct or possibly faulty. To carry out this labelling, the

Table 1
Key elements of the related works summarized.

Article Ref	Computing	Approach	Dataset	Source of faults	System performance
Chen et al. [17]	Distributed	Neighbours	Artificially generated data from a simulated scenario composed of 1024 sensor nodes randomly deployed in a region of size 32 × 32 units	Artificially injected	Performance metric: Faulty sensor detection accuracy; Results: 7 neighbours and 25% fault ratio: 97%; 10 neighbours and 25% fault ratio: 99%
Jiang [18]	Distributed	Neighbours	Artificially generated data from a simulated scenario composed of 200 sensor nodes randomly deployed in a region of size 30 × 30 units	Artificially injected	Performance metric: Faulty sensor detection accuracy; Results: 5 neighbours and 25% fault ratio: 98.5%; 10 neighbours and 25% fault ratio: 99.2%
Zidi et al. [20]	Centralised	Machine learning	Relative humidity and air temperature measurements. Data were obtained in 2010 by researchers at the University of North Carolina at Greensboro	Artificially injected	Performance metric: Faulty sensor detection accuracy; Results: SVM with 25% fault ratio: 99.9%; NB with 25% fault ratio: 98.0%; Hidden Markov Models 25% fault ratio: 96.0%
Noshad et al. [21]	Centralised	Machine learning	Relative humidity and air temperature measurements. Data were obtained in 2010 by researchers at the University of North Carolina at Greensboro	Artificially injected	Performance metric: Faulty measurement detection accuracy; Results: SVM with 25% fault ratio: 92.0%; RF with 25% fault ratio: 95.0%
Yuan et al. [22]	Centralised	Machine learning	Air temperature, relative humidity, light and battery voltage. The data were collected from 54 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004	Artificially injected	Performance metric: Measurement classification accuracy; Results: SVM: 82.0%; NB: 83.5%; GBDT: 91.0%
Zhang et al. [23]	Centralised	Machine learning	Fifteen signals from wind turbines monitored by sensors, including wind velocity at hub height, rotor angular velocity or generator angular velocity, among others. Data were obtained from the National Renewable Energy Laboratory (USA)	Artificially injected	Performance metric: Faulty measurement detection accuracy; Results: SVM: 96.1%; RF: 99.9%; XGBoost with RF: 99.9%
Jin et al. [24]	Centralised	Statistics	Randomly generated data from a simulated WSN topology composed of 100 sensor nodes	Artificially injected	Performance metric: Mean deviation, among others, to estimate the similarity of a WSN compared to others WSNs of known health status (proportion of faulty nodes). Results: Kuiper test: 0.1 when comparing a WSN with five faulty nodes to a WSN without faults; Kolmogorov–Smirnov test: 0.05 in the same situation
Our proposal	Centralised	Machine learning	Leaf-turgor pressure and leaf temperature. Data were obtained from 18 leaf-turgor pressure/temperature sensors of a WSN deployed in a Japanese plum tree farm. The experimental field was carried out for six months with late-maturing Japanese plum trees cv. *Angeleno* and *Talete*, located in the Centre for Scientific and Technological Research of Extremadura (CICYTEX) - La Orden in Badajoz (Spain)	Real	Performance metric: Measurement classification accuracy, precision, recall, f1-score and AUC-ROC; Results: SVM: 84.9% of accuracy, 84.9% of precision, 84.9% of recall, 84.2% of f1-score and 0.94 of AUC-ROC; NB: 71.4% of accuracy, 73.1% of precision, 71.4% of recall, 69.8% of f1-score and 0.80 of AUC-ROC; Decision Tree (DT): 75.8% of accuracy, 76.8% of precision, 75.8% of recall, 75.1% of f1-score and 0.76 of AUC-ROC; Logistic Regression (LR): 74.1% of accuracy, 74.5% of precision, 74.1% of recall, 73.0% of f1-score and 0.82 of AUC-ROC; K-Nearest Neighbours (kNN) (k=7): 84.8% of accuracy, 85.9% of precision, 84.8% of recall, 84.2% of f1-score and 0.91 of AUC-ROC

algorithm assesses the differences of measured values at the same instant of time between neighbouring nodes. Then, if the node's measurements do not exceed a threshold compared to the majority of measurements of the neighbours' nodes, the node is labelled as possibly correct, otherwise, possibly incorrect. Then, each node is compared again with its neighbours, assessing whether neighbours are possibly correct or incorrect. Based on the possibly correct or incorrect neighbours, the algorithm finally labels each node as correct or incorrect. To test their proposal, the authors applied it to a simulated WSN of 1024 nodes with erroneous random nodes. The system proved to be able to identify faults with high accuracy (see Table 1, *system performance* column). However, performance worsens exponentially as the ratio (percentage) of erroneous nodes increases.

In [18], an improvement to the previous algorithm is presented. Thus, in this proposal, the second part of the algorithm is improved, i.e. when it compares each node with its neighbours to finally classify them as correct or incorrect. In this sense, it applies a less strict condition to determine the final classification of each node, resulting in fewer nodes being misdiagnosed as faulty. To test their proposal, the authors carry out several experiments where they applied their proposal on a simulated WSN of 200 sensors with different ratios of faulty nodes. The experiments show that the performance is improved compared to the previous version with high error rates, identifying more than 94% of faulty nodes compared to the previous version which detected 83% for a high error rate of 30% of faulty nodes.

Subsequently, ML gained popularity in virtually all research fields, due to its broad scope of application and its effectiveness in solving problems. Therefore, most current work relies on ML to identify whether a sensor is failing or not.

Thus, in [20], a WSN fault detection system based on a learning model trained with the Support Vector Machine (SVM) classifier is proposed. This learning model is able to identify and classify different kinds of sensor faults, specifically random faults, offset faults, gain faults, stuck-at faults, and out-of-bounds faults. To train the learning model, the authors use a humidity and temperature dataset published by the University of North Carolina, injecting into it the above-mentioned kinds of errors. To test their system, the authors carry out several experiments considering different error rates, classifying the measurements with 99% of accuracy in all cases.

In [21], a WSN fault detection system based on ML is proposed. This system is able to identify several types of sensor faults such as offset, gain, stuck-at, out-of-bounds, spike, and data loss faults. To train the learning models, the same dataset used in the related work described above is used in this work. Again, it is the authors who inject the sensor fault data into the dataset. In this

case, sensor fault data are injected randomly at different rates (10%, 20%, 30%, 40%, and 50%). To train the learning model, the authors use and compare the performance of several algorithms such as SVM, multilayer perceptron, convolutional neural networks, Random Forest (RF), stochastic gradient descent and probabilistic neural networks. The RF algorithm is reported as the best with a faulty measurements identification accuracy of 90% for almost all kinds of faults and rates.

The communication presented in [22] carries out a comparative analysis of SVM, Naive Bayes (NB) and Gradient Boosting Decision Tree (GBDT) for data fault detection in WSNs. To train the learning models, it is used a public dataset from 54 sensors deployed in the Intel Berkeley Research lab (2004). These devices collected humidity, temperature, light, and voltage values once every 31 s. Regarding sensor fault data, the authors injected three different types of faults: noise fault, gain fault and stuck-at fault. Thus, after training the learning models the communication reports GBDT as the best model with an average accuracy of 90% classifying all different kinds of faults.

In [23], a data-driven design for fault detection of wind turbines using RF and XGboost is presented. To train the learning models, the authors use a dataset from The National Renewable Energy Laboratory (USA). In this work, not only sensor failures but also actuator failures are detected. Thus, ten different fault scenarios are defined, including six sensor faults and four actuator faults. Again, it is the authors who inject sensor fault data into the dataset. On the other hand, in the dataset's pre-processing stage, the RF algorithm is applied to assess and select the most representative dataset's features. Then, when features are selected, the fault classifier is trained with the XGBoost algorithm. The resulting model reports an average accuracy of 90% classifying each kind of fault.

The works described above are some of the current proposals for sensor fault identification. As mentioned above, since ML has become popular, ML approaches are the most common to solve the problem of sensor fault detection. However, there are other current approaches based on other kinds of techniques that are also effective solving this problem.

In this regard, the communication presented in [24] presents a statistical approach for WSN fault diagnosis based on the Autoregressive model [25], using the Kuiper test [26] and the Kolmogorov–Smirnov test [27], which are statistical methods used to determine whether two given distributions are similar or significantly different. With the results of the above-mentioned tests, it is theoretically possible to compare a WSN with other WSNs whose health states are known (proportion of faulty nodes) to assess the similarity between them, thus being able to estimate the health states of a WSN. The authors use several metrics such as mean deviation to evaluate the performance of both tests in estimating the similarity between WSNs with known health status. After assessing the results achieved, the authors concluded that both tests are able to evaluate the health conditions of WSNs, however, the Kuiper statistic shows better performance than the Kolmogorov–Smirnov statistic. The main drawback of this approach is that it is not possible to find out the exact number of faulty nodes.

Finally, note that Table 1 shows a summary of the related works described together with the proposal presented in this communication.

As witnessed by the related work described above, sensor fault detection has been and is currently a matter of research interest. However, most studies in the literature are theoretical and generalist. These studies do not focus on fault detection of a specific sensor and do not apply to a real WSN, furthermore, these proposals only consider artificially injected faults. On the other hand, the proposal presented in this communication uses a dataset that comes from a real WSN of leaf-turgor pressure sensors, including the real sensor faults. So, there is no need to inject sensor faults artificially. Moreover, in order to validate the learning model, data from this WSN is used, thus corroborating its effectiveness in a real WSN.

In short, the difference between our proposal and those presented in the related works is that our proposal gathers all the knowledge that the different related works have reached, and uses it to apply it to a real WSN. Thus, solving a real problem of sensor fault detection in WSNs composed of leaf-turgor pressure sensors.

3. Material and methods

In this Section, the material and methods used are illustrated. In this regard, Section 3.1 describes the countryside context and presents the leaf-turgor pressure sensors. Next, Section 3.2 addresses how the leaf-turgor pressure data has been gathered and presents the dataset. Finally, Section 3.3 shows the pre-processing techniques, ML algorithms and validation metrics applied to train the learning models.

3.1. Countryside context and leaf-turgor pressure sensors

The experimental field was carried out for six months with late-maturing Japanese plum trees cv. *Angeleno* and *Talete*, located in the CICYTEX - La Orden Research Center in Badajoz (Spain). The experimental plot consisted of nine trees with two leaf-turgor pressure sensors installed on each tree, distributed along the orchard (Fig. 1).

On the other hand, a leaf-turgor pressure sensor consists of a pressure chip embedded in a gel and limited by a plastic chamber. Surrounding the chamber is placed a metal ring which, together with a counter magnet, is fixed to the tree leaf (see Fig. 2). The photosynthesis process of the plant with the rise and fall of the sun causes cyclical increases and decreases of the leaf-turgor pressure level over time. In addition, changes in the plant's water stress directly affect leaf size and behaviour, which are recorded by the chip. The sensor turns these changes into an electrical signal that sends to a communication box continuously. The communication box is equipped with Bluetooth and every five minutes sends the readings to a gateway that stores the information and once per hour sends a data packet to a cloud where it is stored and processed.

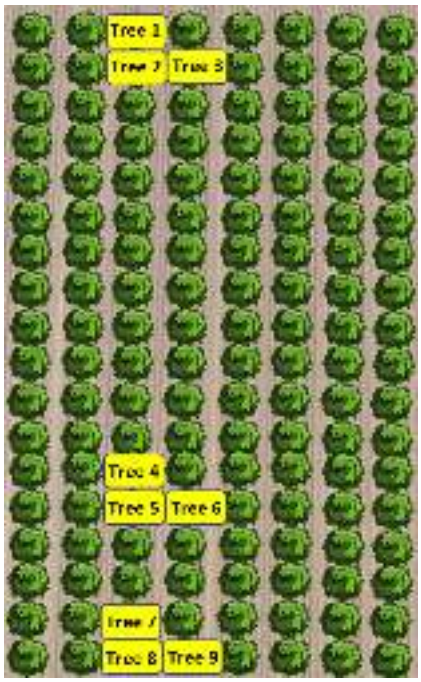


Fig. 1. Experimental plot. It consists of nine trees, each one with two leaf-turgor pressure sensors installed.

Fig. 2. Sensor detail installed on the plum trees.

3.2. Leaf measurements and IoT-sensor data gathering

Nine trees were monitored installing two leaf-turgor pressure sensors per tree. The leaf-turgor pressure sensors were installed on May 13th and removed on October 5th and the frequency of data gathering was set at 5 min. Note that leaf-turgor pressure sensors also measure the temperature of the leaf. Fig. 1 shows the distribution of monitored trees throughout the farm.

On the other hand, Fig. 3 shows the well-defined cycles (24 h) that leaf-turgor pressure presents as a result of the process of photosynthesis. These cycles show how leaf-turgor pressure increases during the day, as solar radiation increases, and decreases in the evening, as solar radiation decreases, remaining constant during the night. Other parameters such as crop's water-stress can also affect the behaviour of leaf pressure [28]. So, plum's leaf-turgor pressure curves could present different distributions and/or behaviours. Figs. 3 shows 30 random days of correct leaf-turgor pressure measurements.

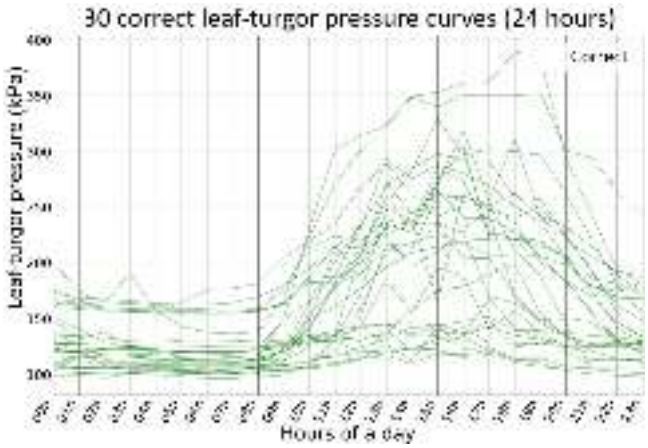


Fig. 3. 30 randomly selected leaf-turgor pressure curves (24 h). All measurements have been classified as correct by the experts.

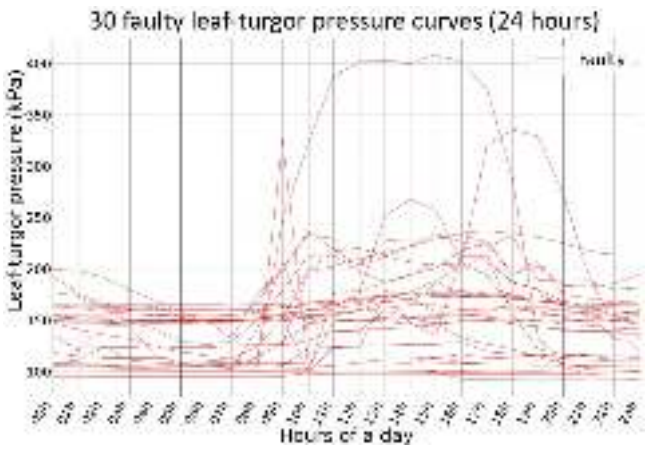


Fig. 4. 30 randomly selected leaf-turgor pressure curves (24 h). All measurements have been classified as incorrect (faulty sensors) by the experts.

Furthermore, Fig. 4 shows 30 random days of incorrect leaf-turgor pressure measurements. Fig. 4 evidences that faulty sensors present more heterogeneous and arbitrary measurements than correct ones. In this regard, several kinds of well-known errors found in the literature can be identified, such as stuck-at faults, i.e., days where the leaf-turgor pressure barely varies, or spike faults, i.e., days where the rate of change of the measured time series with the predicted time series is higher than the acceptable change trend [19], among other kinds of faults. However, there are also several instances where both the measurements of the faulty sensors and the correct ones are similar.

For these reasons, the identification of faulty measurements is a task to be carried out by experts. In this way, experts inspect leaf-turgor pressure curves on a daily basis with the aim of identifying incorrect measurements. This inspection is carried out by displaying leaf-turgor pressure charts where the last 24 h of measurements of each sensor are shown. By analysing these charts, experts can identify whether a sensor is behaving abnormally, and then associate a binary error code to each measurement depending on whether the sensor was considered correct (1) or faulty (0). Furthermore, since some correct measurements could be identified as errors, and vice versa, experts physically check the sensors on a daily basis with the aim of ensuring the reliability of their classification (correct or incorrect measurement).

In the course of this process, the experts have, over a period of one year, elaborated the dataset used to train the learning models, which is shown in Table 2.

Finally, note that, since this kind of sensor also gathers leaf-temperature data, leaf-temperature behaviour has been also analysed with the aim of identifying sensor faults. However, as experts stated, no patterns have been found in the leaf temperature data that would help to identify sensor faults.

Table 2
Dataset description.

Feature	Description	Type
$sensorId$	sensor ID	String
$timestamp$	timestamp	Date
P_{leaf}	leaf-turgor pressure (kPa)	Float
T_{leaf}	leaf temperature (°C)	Float
$ErrorCode$	sensor error code	Boolean

Table 3
Re-framed time series to supervised (two variables and time steps).

var1(t-2)	var2(t-2)	var1(t-1)	var2(t-1)	label
1.0	51.0	2.0	52.0	0
2.0	52.0	3.0	53.0	0
3.0	53.0	4.0	54.0	1
4.0	54.0	5.0	55.0	1

3.3. Data pre-processing techniques, ML algorithms and learning models validation

This Section focuses on describing the dataset pre-processing carried out (Section 3.3.1), the ML algorithms applied to train the learning models (Section 3.3.2), and the validation metrics used to measure the performance of the learning models from different perspectives (Section 3.3.3).

3.3.1. Dataset pre-processing

The dataset pre-processing phase is performed before the training of the learning models and aims to generate reliable input data. So, the main purpose of this stage is to increase the performance of the learning models in terms of classification accuracy, time in building a classifier, the size of the classifier, etc. [29,30].

Regarding pre-processing techniques, two types of pre-processing techniques can be usually distinguished: (1) Data Reduction and (2) Data Projection. *Data Reduction* includes those techniques that focus on modifying data features in order to optimise data quality. *Data projection* focuses on the necessary transformations of the raw data into an optimised and feasible representation for each particular learning algorithm [29].

Data Reduction. The techniques applied in this context are *UnderSampling*, *OverSampling* and *Mislabels Correction*. *UnderSampling* is the process of decreasing the amount of majority target instances or samples [31]. *OverSampling* means increasing the volume of instances of a particular class [32,33]. Finally, mislabels occur when an observation is incorrectly labelled, *Mislabels Correction* is performed by flipping the label [34].

Data Projection. The techniques applied in this context are *Min-Max Normalisation* and *Re-frame Time Series as Supervised Learning*. *MinMax Normalisation* is a technique able to scale data in a range bounded by a predefined minimum and maximum limit. [35]. As per *Min-Max Normalisation* technique see Eq. (1).

$$A' = \left(\frac{A - min\ value\ of\ A}{max\ value\ of\ A - min\ value\ of\ A} \right) \times (D - C) + C \tag{1}$$

Where A' is *Min-Max Normalised* data, $[C, D]$ is the predefined boundary and A is a data within the range of original data.

On the other hand, *Re-frame Time Series as Supervised Learning* is applied to transform time series forecasting and classification problems into supervised learning problems for their use with ML algorithms [36].

A time series is a sequence of numbers that are ordered by a time index. A supervised learning problem is comprised of input variables X and an output variable Y, and an algorithm can be used to learn the mapping function from the input to the output. The goal is to approximate the underlying true mapping so that when there is new input data X, the output Y for that new input can be predicted [36], see Eq. (2).

$$Y = f(X) \tag{2}$$

Re-frame Time Series as Supervised Learning consists of using the previous measurements of N time steps as the input variables X and the value of the next time step or a label as the output variable Y. Table 3 shows an example of a classification problem in which measurements of two variables from two previous time steps are considered to determine a label.

3.3.2. Algorithms applied

In order to carry out the experiments proposed in this work, several ML algorithms have been applied: *k-Nearest Neighbours*, *Support Vector Machine*, *Naive Bayes*, *Decision Tree* and *Logistic Regression*.

- *K-Nearest Neighbours (kNN)* [37] is based on determining the similitude between examples. Thus, the classifier compares their attributes-based descriptions. The fact of each example is represented by a point in an *n-dimensional* space which makes it possible to calculate the geometric distance between any pair of examples. The closer to each other examples are in the instance space, the greater their mutual similarity. The classifier identifies not only one neighbour but also k neighbours. So, k-NN classifier is a similarity classifier where k is the number of the voting neighbours.

- *Support Vector Machine (SVM)* is a supervised ML that analyses data for classification and regression analysis which belongs to the kernel-based algorithms [38]. SVM is an algorithm for maximising a particular mathematical function with respect to a given collection of data [39]. So, the target is to determine the vector support and their margins. The algorithm should identify the support vector that maximises the margin. Consequently, training an SVM involves searching for a separating hyperplane that leads to the maximum margin as this will best separate the levels of the target feature [40].
- *Naive Bayes (NB)* can predict class membership probabilities, such as the probability that a given sample belongs to a particular class. Bayesian classifiers are based on Bayes' theorem. Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes [41]. The NB classifier performs remarkably well even when the underlying independence assumption is violated [42].
- *Decision Tree (DT)* algorithm is part of the supervised learning algorithm family, and its main objective is to construct a training model that can be used to predict the class or value of target variables through learning decision rules inferred from the training data. The DT algorithm can be used to solve regression and classification problems [43].
- *Logistic Regression (LR)* is a multivariable method devised for dichotomous outcomes. It is a standard statistical classification method which is particularly appropriate for models involving binary classification problems. It has been widely applied due to its simplicity and great interpretability [44].

3.3.3. Learning model validation metrics

In order to assess the performance of the trained learning models, metrics such as *Accuracy*, *Precision*, *Recall*, *F1_score*, *Matthews correlation coefficient* or *Cohen's kappa statistic* [45] have been calculated. *Accuracy* is computed as the number of correctly classified data over the total number of data [46], see Eq. (3). True Positive (TP) and True Negative (TN) mean the number of correctly classified as positive or negative. False Positive (FP) means that a negative instance is predicted as positive, and False Negative (FN) means the opposite (predicted negative when the instance is positive).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

Precision or Confidence (as it is called in Data Mining) denotes the proportion of Predicted Positive cases that are correctly Real Positives [47], that is the number of True Positives divided by the sum of True Positives and False Positives, see Eq. (4).

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Recall or Sensitivity (as it is called in Psychology) is the proportion of Real Positive cases that are correctly Predicted Positive [47], that is the true positive rate (the number of True Positives divided by the sum of True Positives and False Negatives), see Eq. (5).

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

F1_score is calculated from *Precision* and *Recall*. It is the harmonic mean of the precision and recall [45], see Eq. (6).

$$F1_score = 2 \times \frac{precision \times recall}{precision + recall} \tag{6}$$

Matthews Correlation Coefficient (MCC) is a validation metric that, when applied to binary classification problems, returns the Phi-Coefficient [48,49]. It is a value between −1 and +1, where −1 indicates a perfect negative correlation between predictions and actual labels, 0 indicates no correlation (i.e. the model predicts randomly) and +1 indicates a perfect positive correlation [45]. MCC is defined as Eq. (7).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{7}$$

Cohen's kappa statistic (Kappa) is a validation metric that takes into account the probability of obtaining the correct classifications by chance [50]. It is defined as Eq. (8).

$$Kappa = \frac{P_0 - P_e}{1 - P_e} \tag{8}$$

In Eq. (8), P_0 is the proportion of correct classifications, i.e. accuracy, and P_e is the probability of performing correct classifications by chance. P_e can be computed as Eq. (9), where n is the number of total instances [45].

$$P_e = \frac{TP + FP}{n} \times \frac{TP + FN}{n} + \frac{FN + TN}{n} \times \frac{FP + TN}{n} \tag{9}$$

Like MCC, Kappa also ranges from −1 to +1, where 0 means random classification and +1 means perfect classification.

Additionally, well-known measures such as AUC ROC [51] will be used to measure the performance of the ML models proposed. ROC related *Sensitivity* and *1 - Specificity* [40]. It should be noted that *Sensitivity* is calculated as *Recall*, see Eq. (10).

$$Sensitivity = \frac{TP}{TP + FN} \tag{10}$$

Specificity is *Recall* measures with negative examples, see Eq. (11).

$$Specificity = \frac{TN}{TN + FP} \tag{11}$$

Table 4
Experiments carried out to allow the identification of the best performing pre-processing conditions.

Experiment ID	Description
Experiment 00	Applying the techniques of related works, two different approaches can be distinguished. (A): Models are trained using the original dataset and no pre-processing techniques are applied. (B): Applying *Re-frame Time Series as Supervised Learning* (three time steps) to the original dataset.
Experiment 01	Applying hourly *UnderSampling*, *Re-frame Time Series as Supervised Learning* (24 h) and *MinMax Normalisation* by sensors as pre-processing techniques to the original dataset.
Experiment 02	Applying *MinMax Normalisation* by time ranges within each sensor and *Mislabels Correction* to the Experiment 01 dataset.
Experiment 03	Applying *OverSampling* to minority class in the Experiment 02 dataset.

The ROC curve summarises all the confusion matrices that each threshold produced. That is, the ROC curve is obtained by moving the model threshold between [0-1] and obtaining the values of *Sensitivity and 1-Specificity* for each threshold. The *ROC index* or AUC measures the area underneath a ROC curve [40].

It is possible to generate ROC curves from a K-fold cross-validation process and analyse the variation of the curve at each fold. In addition, from all the ROC curves of each fold, it is possible to calculate the arithmetic and weighted average ROC curve and thus the arithmetic and weighted average AUC.

Finally, note that no neuronal networks have been applied. In this regard, the firsts steps in training the model involved the use of recurrent neural networks such as Long Short-Term Memory (LSTM) networks (given the recurrent nature of the problem); and Extreme learning machine neuronal networks, in view of the good results obtained in works involving sensor data related to agriculture, as the current problem [52–54]. The results obtained using these neuronal networks reported similar predictive performance to those presented above. However, neuronal networks have three major disadvantages compared to these: (1) Slower model training, (2) Greater complexity in terms of model training (network architecture development and parameter configuration) (3) Additional complexity to replicate the training of the models due to (1) and (2). One of the main drawbacks of most neural network optimisation methods is their low efficiency; more complex methods tend to achieve better results, but this is offset by a disproportionate increase in computational costs [55].

In view of these disadvantages and the fact that the application of such algorithms does not add extra value to this communication, the authors have considered that the inclusion of this stage of the research in the article is fruitless.

Finally, several clustering techniques have also been unsuccessfully assessed to solve the problem, e.g. K-Means and Density-based spatial clustering of applications with noise (DBSCAN). As with the algorithms discussed above, the authors have decided not to include this phase of the research in the paper, as it does not provide any additional value.

4. Experiments carried out for tuning the ML models and results

This section shows the experiments carried out to train the learning models. The aim of these experiments is to increase the performance of the trained learning models. To achieve it, each experiment is focused on the application of different pre-processing techniques. Note that the experiments described in this section are a selection/summary of all those that have been carried out, including those from which it has been possible to improve the performance of learning models. Table 4 summarises the main experiments carried out.

Before addressing the description of the experiments, this section shows how the dataset has been used and how the performance metrics have been applied to measure the results obtained in each experiment.

4.1. Dataset and validation metrics in experiments

Next, both the dataset and the application of validation metrics used throughout the set of experiments are described.

4.1.1. Dataset in experiments

The original dataset is formed by the features defined in Table 2 and by 974.493 rows. This dataset is the result of monitoring plum crops for one year's fruit ripening cycle following the methodology described in Sections 3.1 and 3.2.

In order to use this dataset to train learning models, it is necessary to split the dataset into a training and a validation set. There are several techniques to carry out this splitting and the most common is to randomly choose a set of rows and use them as validation set [56]. However, the dataset provided in this communication presents some peculiarities, so some considerations have to be taken into account before applying this technique to it.

Splitting the data set randomly means that data relating to readings from the same sensor could be assigned to both the training set and the validation set. As each sensor presents a particular behaviour, this scenario could lead to train models that only correctly classify the measurements of sensors that have similar behaviour to the sensors on which the models have been trained, i.e., not being able to generalise.

In this regard, not all sensors, even if they belong to the same tree, are located on similar leaves or share the same meteorological conditions. A small difference such as the cardinal orientation of the sensor may determine different exposure to sunlight or wind on the leaf and thus to a different leaf-turgor pressure behaviour.

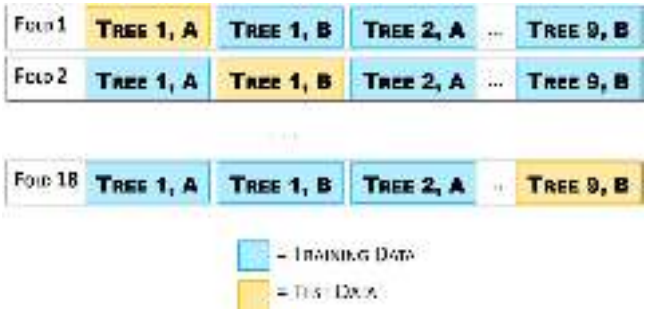


Fig. 5. K-Fold validation split by sensors.

For these reasons, the data relating to the readings of the same sensor are handled as an inseparable set. Thus, to randomly split the dataset, instead of randomly choosing a set of rows as described in [56], the whole data of a sensor is randomly assigned together to either the training set or the validation set.

So, validating with readings from sensors that are not in the training set implies that the models will be able to apply what they have learned to unknown situations, i.e., to classify the correct or incorrect status of sensors attached to leaves of trees that do not have similar behaviour to the devices with which the model has been trained.

Note that these considerations are made because the results achieved in the first stages of training, when the dataset was randomly divided regardless of whether the readings from the same sensor belonged to both the validation or training set, led to the undesired scenario described above, i.e. the models do not generalise as expected.

4.1.2. Application of the validation metrics

In order to achieve reliable results in terms of performance measurements, cross-validation is applied [57]. Taking into account the considerations made in Section 4.1.1, the cross-validation is carried out using in each iteration the data from one sensor for testing and the data from the rest for training, as shown in Fig. 5. It is worth mentioning that the dataset used for training and testing at each fold is not only composed of the faulty records of the sensors but an equal number of correct records (as it is the majority class) is sampled in order to balance the classes.

The results achieved in each iteration are weighted, i.e. multiplied by the size of the sensor dataset reserved for validation. It has been considered suitable to weigh the results as there are sensors with significant variations in the number of errors made. In this regard, it is considered inappropriate for a sensor with a single fault well identified by the learning models, that could achieve a 100% of accuracy, to have the same weight in the final performance assessment as a sensor with 50 errors and 85% accuracy.

Taking the above-mentioned into consideration, the following Eq. (12) indicates how to calculate a performance metric (e.g. accuracy, precision, recall, f1-score and the AUC-ROC) with the sensor-weighted cross-validation method.

$$Metric = \frac{1}{nF + nC} \times \sum_{i=1}^{sensors} Metric_i \times (nF_i + nC_i) \tag{12}$$

Where $Metric$ is the weighted average of the performance metric throughout the cross-validation process, nF and nC are the total number of faulty and correct records of the dataset respectively, $sensors$ represents the set of the 18 sensors included in the dataset, $Metric_i$ is the performance reported by the validation metric in the cross-validation iteration i, and nF_i and nC_i are the number of faulty and correct records respectively of the sensor reserved for testing purposes in the cross-validation iteration i.

In other words, the expression explained above calculates the weighted average of the performance metric throughout the cross-validation process. Thus, at each iteration of the cross-validation process, the performance obtained is multiplied by the size of the sensor dataset reserved for validation ($nF_i + nC_i$). Finally, the resulting amount of the 18 iterations is divided by the total data size ($nF + nC$).

4.2. Experiment setup and experimentation

The motivation, reasoning, set-up, experimentation and results achieved in each experiment are illustrated below.

4.2.1. Experiment 00: First approach, applying the techniques of related works.

This first experiment aims to carry out a first approach to the problem, clarifying whether it is possible to identify faulty sensors in a real application case by means of the findings and pre-processing techniques used in related works, which are theoretical.

As discussed in depth in Section 2, in these related works the faults are artificially injected and usually do not apply to specific sensors but are generalist. The most recent ones, employ ML algorithms to generate classification models capable of classifying the measurements as correct or incorrect. All of them hardly apply any pre-processing techniques to the data, however, they achieve very high performances in detecting erroneous measurements.

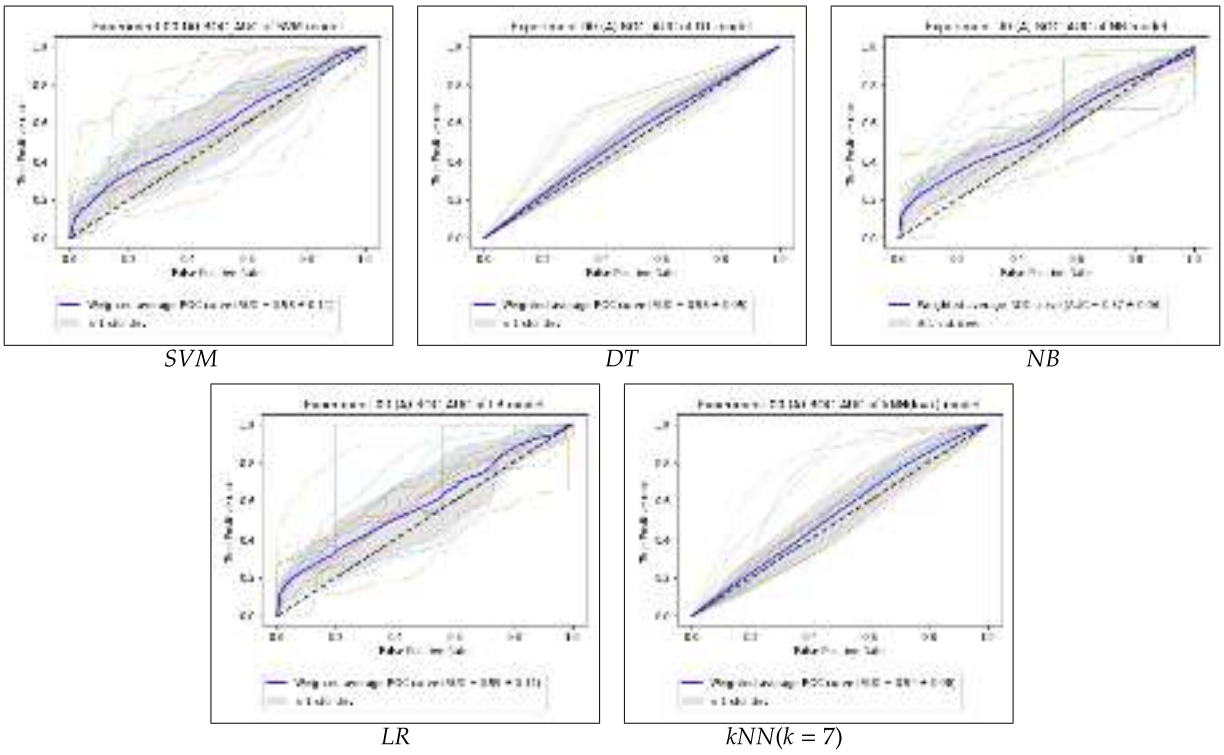


Fig. 6. ROC curve and AUC for the ML models developed in Experiment 00 (A).

Table 5
Experiment 00 (A) results. The values given for the metrics are the weighted average of all iterations of the cross-validation.

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC	MCC	Kappa
SVM	56.2%	61.6%	56.2%	51.3%	0.58	0.17	0.12
DT	52.7%	52.8%	52.7%	51.7%	0.53	0.05	0.05
NB	55.8%	58.5%	55.8%	53.3%	0.57	0.14	0.12
LR	53.9%	55.5%	53.9%	52.5%	0.59	0.09	0.08
kNN(k=7)	52.6%	52.7%	52.6%	51.7%	0.54	0.05	0.05

In [22], measurements from temperature and voltage sensors are used and no pre-processing techniques are applied to the data before feeding it into the ML algorithms, the authors achieve around 90% of accuracy in classifying the measurements as correct or incorrect. Moreover, in [23], signals from sensor-monitored wind turbines are used, faults are injected and data are fed into the ML algorithms. Only Random Forest is applied for the selection of the most representative features; no further preprocessing techniques are specified, however, the authors report 99.9% of accuracy in detecting incorrect measurements.

Thus, the aim of the first part of Experiment 00, i.e. Experiment 00 (A), is to follow the approach of these two related works, where almost no pre-processing techniques are applied to the data. Therefore, in Experiment 00 (A) the ML models are trained and validated following the cross-validation process explained in Section 4.1.2 without applying any pre-processing techniques. Each record is composed of only three features, i.e. two input features, a leaf-turgor pressure measurement and a leaf temperature measurement at a specific time instant, together with the class label or output feature (correct or incorrect).

Table 5 shows the Experiment 00 (A) results. Note that in this table, the neighbours of kNN (k) are set to seven (k=7). This is the result of a grid search process [58] that has been carried out to identify the optimal number of neighbours in kNN. Thus, after applying this process, seven neighbours (k=7) were found to be optimal for maximising the performance of the kNN models. On the other hand, the weighted average of ROC curves and AUC values can be observed in Fig. 6. Moreover, Fig. 15 included in the appendix (Appendix) shows not only the weighted average but also the performance of each individual sensor through the cross-validation process.

In view of the results achieved (see Table 5), it is shown that the classification performance of the measurements is very poor following the strategies of [22,23]. This is probably because it is a real use case with real faults, and more data pre-processing techniques are needed to increase the performance. However, no random classification is being performed, e.g. 50% of accuracy, as around 56% of this metric is obtained in the case of SVM or NB.

On the other hand, there are related works in the literature that already include *Re-frame Time Series as Supervised Learning*, such as [20,21], where this pre-processing technique is applied. In both studies, data from two humidity and temperature sensors at the

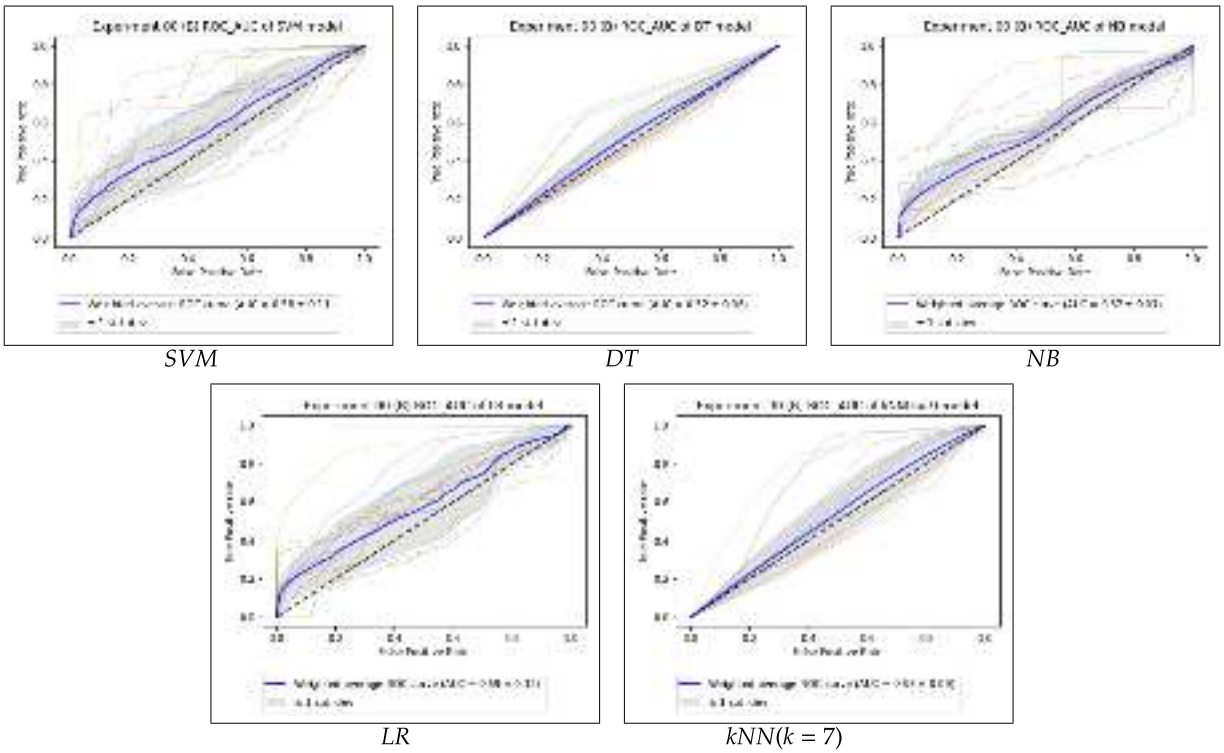


Fig. 7. ROC curve and AUC for the ML models developed in Experiment 00 (B).

Table 6
Experiment 00 (B) results. The values given for the metrics are the weighted average of all iterations of the cross-validation.

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC	MCC	Kappa
SVM	56.1%	61.2%	56.1%	51.2%	0.58	0.16	0.12
DT	52.4%	52.4%	52.4%	51.1%	0.52	0.04	0.04
NB	55.7%	58.3%	55.7%	53.1%	0.57	0.14	0.11
LR	53.7%	55.2%	53.7%	52.3%	0.59	0.09	0.07
kNN(k=7)	52.5%	52.6%	52.5%	51.6%	0.53	0.05	0.05

University of North Carolina at Greensboro are used and three consecutive time steps are employed to generate the time series, thus generating records of 12 input features or dimensions, i.e., two temperature and humidity measurements at each time step, since there are two sensors, for three consecutive time instants. No further pre-processing techniques are applied in these studies, however, in both cases, an accuracy of around 95% is achieved in the detection of incorrect measurements.

Therefore, in the second part of this experiment, i.e. Experiment 00 (B), the aim is the same as in the first part, but now *Reframe Time Series as Supervised Learning* is applied in the same manner described in the two related works [20,21] discussed above. Since this is the only pre-processing technique used in both related works, in Experiment 00 (B) the measurements are fed into the ML algorithms without further pre-processing. The generated records have seven dimensions, i.e. six input features, which are the leaf-turgor pressure and leaf temperature of three consecutive time instants, together with the class label or output feature.

Note that, as in related works, the generated time series are composed of three measurements or time steps spaced a few minutes or even seconds apart. This leads to the leaf-turgor pressure and leaf temperature values being practically identical throughout the time series, however, this has been done on purpose as the aim of this first experiment is to follow as similarly as possible the techniques applied in the related works, assessing its applicability to a real use case such as the one in this paper.

Table 6 shows the Experiment 00 (B) results. The weighted average of ROC curves and AUC values can be observed in Fig. 7. Moreover, Fig. 16 included in the appendix (Appendix) shows not only the weighted average but also the performance of each individual sensor through the cross-validation process.

The results of Experiment 00 (B) are quite similar to those of Experiment 00 (A). A clearly insufficient performance is achieved, however, the classification is not random, as about 56% of accuracy is achieved for SVM and NB in classifying the time series as correct or incorrect.

This first approach has shown that the few pre-processing techniques applied in the related works are insufficient in a real application case such as the one addressed in this paper. The related works are theoretical and generalist and faults are artificially

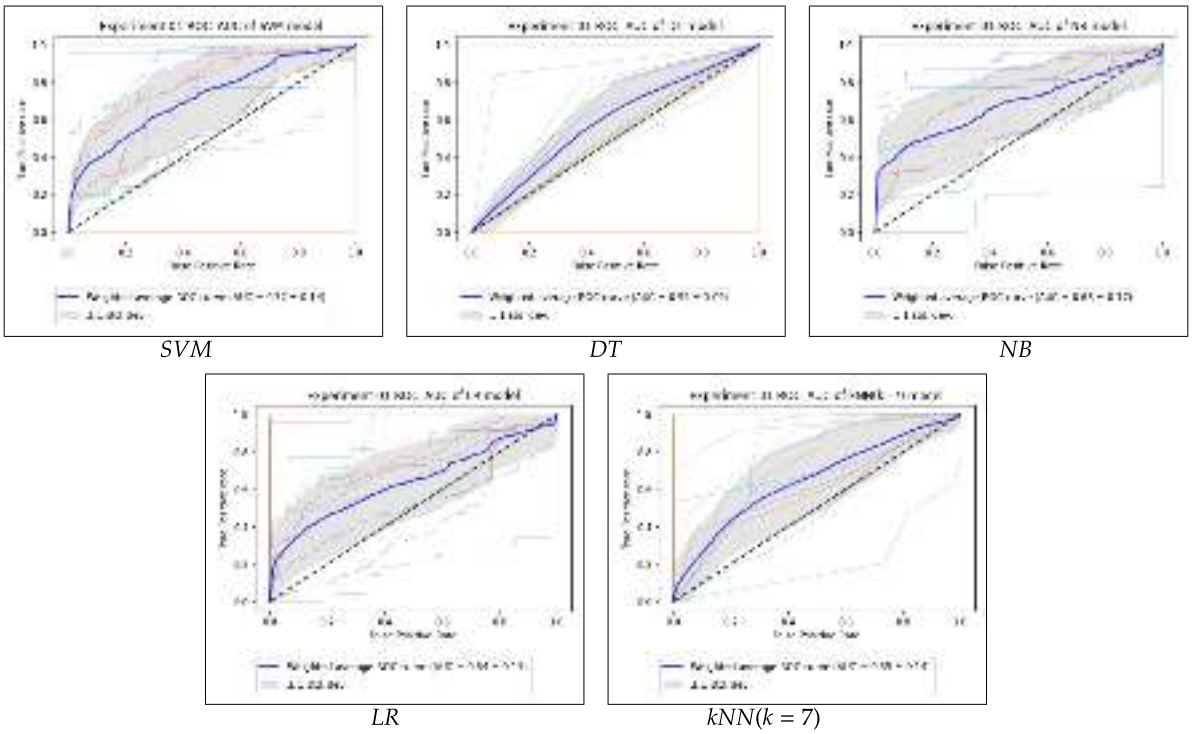


Fig. 8. ROC curve and AUC for the ML models developed in Experiment 01.

injected, which explains such a drastic loss of performance when applied to the case under study, which is real. However, the knowledge and techniques used in the related works seem to be useful in the present case as no random classification has been performed, which means that the models have been able to learn some patterns from the data. In this regard, it is expected that based on the knowledge of the related work and by applying new preprocessing techniques, as well as better adapting them to the problem under study, the performance will be improved, which is what is done in the following experiments.

4.2.2. Experiment 01: Incorporation and adaptation of basic pre-processing techniques to our real problem

In view of the poor performance results achieved in Experiment 00, where the findings and the few pre-processing techniques used in the related work were applied, the aim of this experiment is to increase the performance of the learning models by including new data pre-processing techniques, as well as better adapting them to the real problem under study. Furthermore, this experiment also aims to build a first pre-processed dataset that can be used as a basis for the rest of the experiments.

As described in Section 4.1.1, the original dataset consists of 974,493 rows. This is the result when sensors publish a measurement every two-three minutes during one year's fruit ripening cycle. With the aim of determining whether lower data resolution, i.e. fewer measurements per hour, provides the same knowledge to the learning models, the *undersampling* technique has been applied.

In this regard, the best results in terms of the performance of the learning models were obtained when using one row per hour. After the hourly *undersampling*, the dataset reduced the number of rows from 974,493 to 59,534.

In terms of training, this reduction means more feasible training in terms of the application of further pre-processing techniques, the use of less computational power, and high speed to train models. Regarding the deployment of the model in an IoT system, this reduction implies lower energy consumption of the devices in terms of gathering and publishing data, lower bandwidth usage, lower energy and computational costs of the component involved in pre-processing the data collected by sensors (to provide reliable inputs to the model), lower energy and computational costs in the execution of the model, etc.

Since experts use the daily (24-hour) leaf-turgor pressure curves to determine whether a sensor is performing correctly or not (Section 3.2), the training of the models has been approached in the same way, including not only the leaf-turgor pressure but also the leaf temperature measurements as input features for the algorithms. Thus, once the hourly *undersampling* was applied, the *re-frame of time series as supervised learning* has been made using the measurements of 24 time steps. These time series have 48 input variables (X), i.e. the leaf-turgor pressure and leaf temperature of 24 h, and one value that represents the error code of that 24 measurements as the output variable (Y). Note that the time series where the 24 h do not share the same error code has been deleted.

Finally, the *MinMax normalisation* of the magnitudes according to the maximum and minimum values measured by each sensor has been performed. This is because, as experts reported, each magnitude moves in slightly different ranges for each sensor.

Table 7
Experiment 01 results. The values given for the metrics are the weighted average of all iterations of the cross-validation.

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC	MCC	Kappa
SVM	62.4%	64.4%	62.4%	60.7%	0.72	0.27	0.25
DT	58.0%	58.5%	57.9%	55.9%	0.58	0.16	0.16
NB	63.3%	64.1%	63.3%	62.1%	0.68	0.27	0.27
LR	58.5%	58.7%	58.5%	57.4%	0.64	0.17	0.17
kNN(k=7)	59.8%	59.8%	59.8%	58.3%	0.65	0.20	0.20

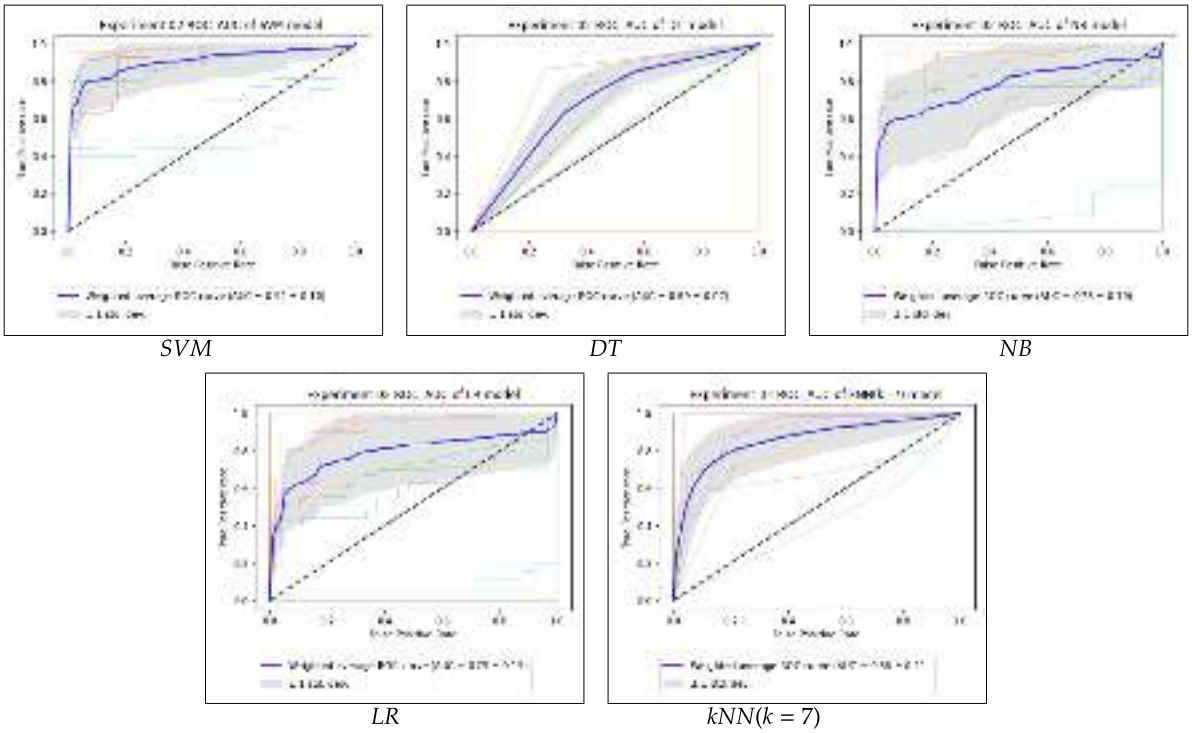


Fig. 9. ROC curve and AUC for the ML models developed in Experiment 02.

In short, in this first experiment, the data have been reduced from 974,493 to 59,534 rows, re-frame to time-series (dimensionality of 48 variables, i.e., the leaf temperature and leaf-turgor pressure of 24 h, and a label indicating whether the series is faulty or not) and *MinMax* normalised.

Table 7 shows the Experiment 01 results. The weighted average of ROC curves and AUC values can be observed in Fig. 8. Moreover, Fig. 17 included in the appendix (Appendix) shows not only the weighted average but also the performance of each individual sensor through the cross-validation process.

In view of the results, this second experiment evidences that the problem can be tackled by ML techniques and the dataset gathered. This is evidenced by the fact that about 60% of *f1-score* was obtained by classifying the test data, achieving an 8% increase in the value of this metric compared to Experiment 00, which was around 52%. Accuracy values have also increased for all algorithms by about 6% compared to the previous experiment.

On the other hand, one of the most relevant performance metrics in this problem is the AUC ROC, which determines the ability of models to discriminate between classes, i.e. the ability of the model to classify incorrect measurements as incorrect and correct measurements as correct (thus avoiding false positives/negatives) [59]. AUC ROC is key in this problem because when a sensor is identified as faulty, experts have to go to the country to inspect and fix the faulty sensor. Therefore, low AUC ROC, or in other words, the frequent occurrence of false positives, implies that experts would have to go to the country unnecessarily frequently, which is an undesirable scenario.

Currently, an AUC ROC of 0.72 has been achieved for the case of SVM (an improvement of 0.14 points over Experiment 00). This AUC ROC performance is not yet acceptable, however, in view of the rest of the performance metrics, the AUC ROC achieved could be considered high. Therefore, it could be that as the other performance metrics improve, the AUC ROC will also improve.

4.2.3. Experiment 02: In-depth dataset analysis and learning model performance improvement

The aim of this experiment is the improvement of the performance of the learning models, as the performance achieved in *Experiment 01* is not enough to replace experts.

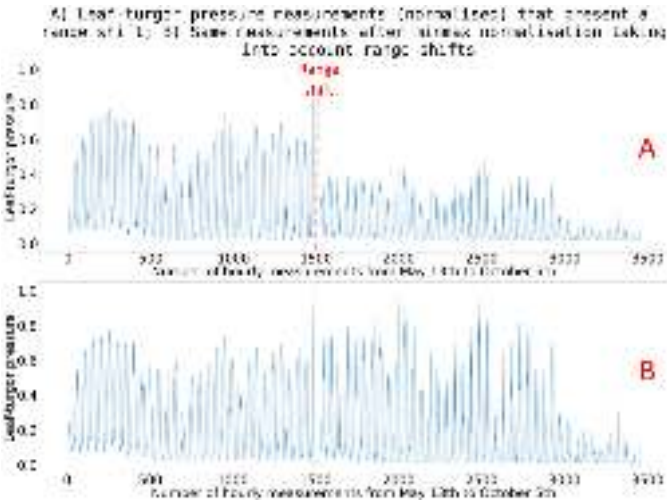


Fig. 10. Leaf-turgor pressure measurements from a sensor normalised without taking into account range shifts (A), and normalised taking into account the range shifts (B).

Table 8
Experiment 02 results. The values given for the metrics are the weighted average of all iterations of the cross-validation.

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC	MCC	Kappa
SVM	81.5%	81.7%	81.5%	80.8%	0.91	0.63	0.63
DT	70.7%	71.9%	70.7%	70.1%	0.68	0.39	0.37
NB	71.1%	73.0%	71.1%	69.7%	0.78	0.44	0.42
LR	69.7%	71.0%	69.7%	67.9%	0.79	0.40	0.39
kNN(k=7)	78.0%	78.7%	78.0%	77.1%	0.86	0.57	0.56

In this regard, analysing the dataset, it is observed that sensors' measurements present range shifts in their measurements (see Fig. 10). As can be seen in Fig. 10, these range shifts occur abruptly at specific time instants. Analysing the data in more depth, it was found that these time instants match the time instants at which a sensor is detected as faulty. According to expert opinion, these range shifts are the result of fixing the sensor. Thus, after a sensor fault, experts place the sensor again correctly, often in a different position or even changing the leaf, which shifts the range in which the sensor was gathering its measurements.

Therefore, in this second experiment, an improved *MinMax normalisation* of the measurements of each sensor is performed taking into account the above-described range shifts.

Furthermore, analysing the dataset, it has been noticed that when sensors' measurements are identified as faulty, several previous days of correct measurements have also been labelled as faulty. After seeking expert opinion, it was reported that since a sensor is manually identified as faulty, the previous days' measurements are also labelled as incorrect to ensure the correctness of the data for subsequent studies. Furthermore, it has been noticed that when a faulty sensor is fixed, experts also label all measurements of that sensor for that day as incorrect.

This is a key aspect to take into account as introducing mislabelled measurements into ML algorithms might have a negative impact. So, *Mislabels correction* has been applied in this experiment by removing all the measurements that are susceptible of have being mislabelled.

Table 8 shows the Experiment 02 results. The weighted average of ROC curves and AUC values can be observed in Fig. 9. Moreover, Fig. 18 included in the appendix (Appendix) shows not only the weighted average but also the performance of each individual sensor through the cross-validation process.

Thus, applying the above pre-processing techniques, this second experiment achieves a performance of 80.8% of *f1-score* in the case of SVM, an 18.7% more than the best result reported by *Experiment 01* (62.1% of *f1-score* in the case of NB and kNN(k=7)). Furthermore, the AUC ROC achieved is 0.91, also in the case of SVM (0.19 more than the best case reported by *Experiment 01*). According to experts' opinion, this increased performance brings learning models closer to an expert in terms of identifying sensor faults.

Therefore, in view of the results obtained, this model could replace the experts in the identification of sensor faults.

4.2.4. Experiment 03. Fine-tuning of the model.

This experiment aims to further increase the performance achieved in *Experiment 02*. In this regard, there are techniques that have not yet been applied, such as data-augmentation techniques (*Oversampling*), which could increase the performance of the models in problems where the number of samples is low.

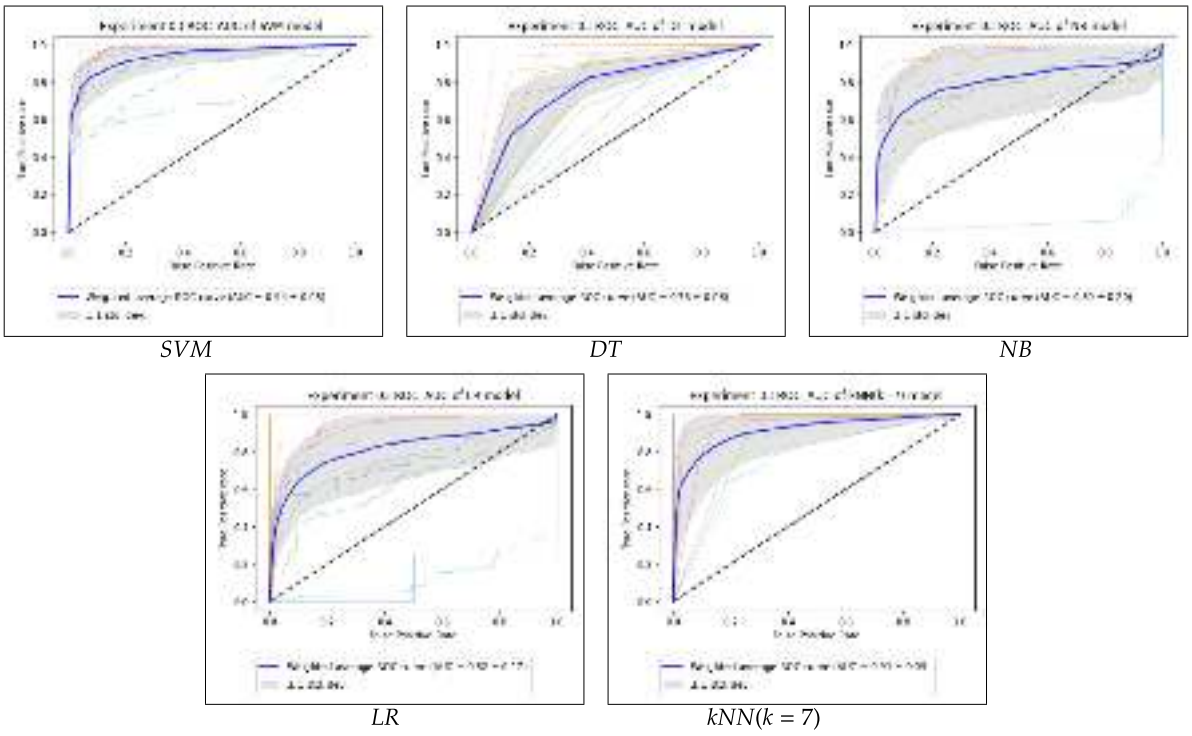


Fig. 11. ROC curve and AUC for the ML models developed in Experiment 03.

Table 9
Experiment 03 results. The values given for the metrics are the weighted average of all iterations of the cross-validation.

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC	MCC	Kappa
SVM	84.9%	84.9%	84.9%	84.2%	0.94	0.71	0.70
DT	75.8%	76.8%	75.8%	75.1%	0.76	0.54	0.53
NB	71.4%	73.1%	71.4%	69.8%	0.80	0.45	0.43
LR	74.1%	74.5%	74.1%	73.0%	0.82	0.49	0.48
kNN(k=7)	84.8%	85.9%	84.8%	84.2%	0.91	0.71	0.70

Thus, *data-augmentation* techniques have been applied. These techniques have been considered particularly interesting for two reasons: (1) The first reason refers to the common benefit of increasing data when dealing with a ML problem, i.e. increasing the number of instances of a particular class allows models to gain more knowledge of this class, learning better and performing better at classifying instances of this class [60]. (2) Secondly, note that the learning models are not only trained and tested with faulty records, but an equal number of correct records is taken, thus providing balanced sets with respect to the two possible classes (faulty measurement and correct measurement). Since incorrect measurements are the minority class, when sampling an equal number of correct records to balance the classes there are a lot of correct measurements that are not included (around 97% of the total). Therefore, increasing the number of incorrect records also means increasing the percentage of correct records that are considered to train and test the learning models. In this way, the algorithms are fed with a wider variety of correct measurements.

The applied data augmentation technique consists of increasing and decreasing incorrect measurements slightly in a random way, generating new measurements similar to the previous one, thus giving the models more variety of possible incorrectly behaving measurements. In this regard, the best results were achieved by generating three new faults from each original fault.

Table 9 shows the Experiment 03 results. The weighted average of ROC curves and AUC values can be observed in Fig. 11. Moreover, Fig. 19 included in the appendix (Appendix) shows not only the weighted average but also the performance of each individual sensor through the cross-validation process.

In this final experiment, an 84.2% of *f1-score* is achieved with the SVM algorithm, a 3.4% more than in *Experiment 02* (80.8% in the case of SVM). This performance also improves AUC, achieving a 0.94 of AUC.

To sum up, the best learning model presents an 84.2% *f1-score* and 0.94 AUC, making it feasible to replace experts, thus automating the process of identifying faulty sensors.

In the following sections, a novel methodology for tackling similar problems involving time series and sensor data will be presented. Furthermore, it will be illustrated how to integrate the resulting ML model into an IoT system so that it can be already deployed and exploited for sensor fault detection in plum tree orchards.

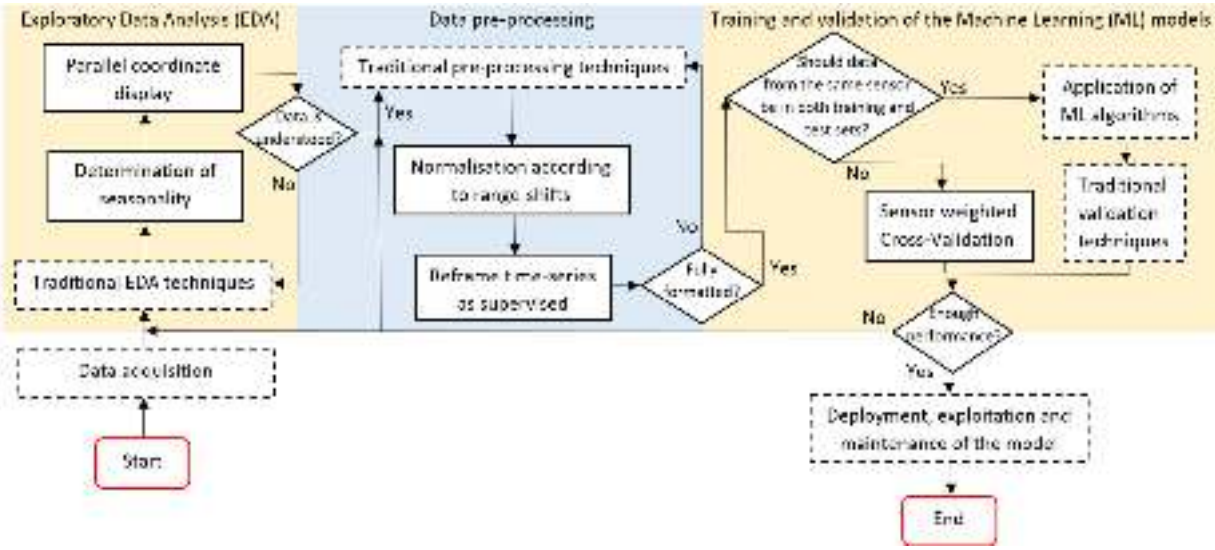


Fig. 12. Flowchart relating the proposed methodology to the main phases of a data science project.

5. Proposed methodology for sensor fault detection in time series

In this paper, not only a learning model for sensor fault detection has been developed together with an IoT system, but also an entire approach for sensor time series classification has been illustrated in a real use case. Since the techniques and knowledge achieved in this work could help to successfully tackle similar problems that integrate time series and sensors with less time and effort, this section presents a methodology that summarises such key ideas and techniques.

Fig. 12 relates the proposed methodology to the main phases of a traditional data science project. Note that in Fig. 12, the traditional steps of a data science project are shown with a dotted border, in contrast to the novel steps proposed in the methodology, which are shown with a solid border. These novel steps proposed in the methodology affect most phases of a data science project, ranging from the exploratory data analysis to the training and validation of ML models (see Fig. 12). The steps of the proposed methodology are explained in detail below:

- **Determining the seasonality of the measured magnitude.** One of the first steps in tackling a problem involving times series is to determine the seasonality time period of the measured magnitude, i.e. the time period in which optional patterns of behaviour are repeated. In the case of the problem addressed in this article, the seasonality time is 24 h since as can be seen in Fig. 3, the leaf-turgor pressure increases during the day with the rise of the sun and decreases at night, which corresponds to the tree's photosynthesis cycles. In addition, the other measured magnitude is the leaf temperature, which is also directly related to the daily cycles. Moreover, as the experts said, they displayed 24-hour charts to label the measurements as correct or incorrect. In other problems, seasonality periods can range from a few minutes to weeks, months or even years.
- **Parallel coordinate display.** To further time series analysis, plots with parallel coordinates can be generated. This paper includes two charts of this nature in Figs. 3 and 4, with 30 days from 00 a.m. to 11 p.m. of measurements classified as correct and incorrect by the experts, respectively. With these charts, it is possible to notice particular patterns and behaviours in the data that provide information and a better understanding of the problem. Note that the visualisation of this type of chart is extremely useful for determining the seasonality time discussed in the previous point.
- **Normalisation according to range-shifts.** As discussed in Experiment 02 in Section 4, a normalisation according to features and even differentiating between sensors is not always enough. It is possible that different ranges of values may be measured for the same magnitude within a sensor (see Fig. 10). In the particular case of this work, ranges shifts occur when CICYTEX workers detect faults in the sensors and change the position or even the leaf where the sensor is attached. This is a non-trivial issue, and the normalisation according to each range shift has produced in this work a substantial improvement in the performance of the learning models. Note that in other problems this situation could also occur, especially with the emerging digital transformation of the agricultural sector with greater monitoring of crops with sensors attached to leaves or tree trunks, where the amplitude of the measured magnitude can be dependent on the exact point where it is placed, as occurs in this work.
- ***Re-frame Time Series as Supervised Learning.*** After proper analysis and normalisation of the dataset, a time series classification or forecasting problem can be approached as a supervised ML problem by reformatting the data, using the previous measurements of N time steps as the input variables (X) and the value of the next time step or a label as the output variable (Y) [36]. For the particular case of this paper *Re-frame Time Series as Supervised Learning* is applied considering 24 h or time steps of leaf temperature and leaf-turgor pressure measurements as input variables (48 features), together with a class label

that indicates whether the time series of measurements has been classified as correct or incorrect by the experts. Table 3 shows an illustrative example of the records for a classification problem in which measurements of two variables from two previous time steps are considered to determine a label.

- **Split by sensors to train and validate models.** As discussed in Section 4.1.1, learning models are trained and validated so that data from the same sensor are never in both stages. Note that in a tree, leaves are oriented towards different cardinal points, have different exposure to the sun or may be older or younger, among other factors, this results in each sensor showing particular leaf-turgor pressure behaviours. Validating with data from sensors not used for training ensures that the performance metrics obtained correspond to the actual performance of the models, since in future it will have to classify measurements from sensors attached to leaves that have never been seen before in the training phase. This is something that can occur in other problems involving time series, especially in the agricultural sector with sensors on crop leaves, therefore validation of models with data from sensors other than those used to train them may also be necessary to ensure that models are able to generalise properly.
- **Sensor weighted cross-validation.** As discussed in the previous point, in order to properly calculate the performance metrics of the models it is sometimes necessary to validate with data from sensors that have never been seen in the training phase. Thus, an alternative version of k-fold cross-validation is proposed in this article (see Section 4.1.2), where the division between folds is made by sensors, always saving the measurements of one sensor for testing and training with the measurements of the remaining ones. Therefore, there are as many folds as sensors. In each fold, the metrics obtained are weighted to calculate the average, given that there are sensors with more data and faults than others. This validation process can be applied to many other problems, where segregating by sensors for training and testing is necessary to ensure that the models generalise properly. Furthermore, AUC ROC charts can be computed by following the same sensor cross-validation process. See Fig. 11 where the AUC ROC folds of Experiment 03 are displayed for every single fold or sensor, and the weighted average ROC curve is calculated and plotted thicker in blue.

As shown, many of the techniques and knowledge achieved in this work can be applied to similar problems involving sensors and time series. In this section, a complete methodology has been proposed to tackle this type of problems, ranging from the analysis of the time series by visualising parallel coordinate charts and determining the seasonality of the measured magnitude, through the transformation of the data so that time series classification and forecasting problems can be tackled as supervised ML problems, to the validation of the results, including as well an alternative cross-validation process that separates training and test sets by sensors in each fold. Therefore, many related future works involving time series could be based on the proposed methodology and thus reduce the development cost in time and effort, as well as improve the results in terms of performance.

6. IoT-based expert system for leaf-turgor pressure sensor fault detection

This section proposes an IoT-based expert system that integrates the learning model trained through the experiments described in Section 4. In this regard, this section is structured as follows: Section 6.1 describes the motivation for developing the IoT system. Section 6.2 identifies some design requirements that the proposed system has to reach. Section 6.3 presents the design, main layers and components of the system. Finally, Section 6.6 conducts a discussion about the trade-offs of the system design.

6.1. IoT-based expert system. Motivation

In this communication, a learning model that is able to identify faulty leaf-turgor pressure sensors has been developed. However, to take advantage of this learning model, it has to be integrated into the WSN of leaf-turgor pressure sensors. Therefore, the main motivation for developing this IoT system is to integrate and take advantage of the trained learning model in a real system. Thus, also showing how the learning model can be deployed as part of an IoT system, its role within the system, interactions with the other devices or software components, etc.

6.2. Iot-based expert system. Design requirements

The IoT currently has several open challenges that should be taken into account in the development of any IoT system. Furthermore, depending on the IoT system, addressing some of these open challenges is a must. Thus, the open challenges that have been considered key to the IoT system that has been developed in this communication are: (1) Interoperability, (2) Energy awareness, and (3) Scalability.

In the context of IoT, interoperability is understood as the property that allows an IoT system to operate with a third party, such as another IoT system, an application, etc. [10]. This open challenge has to be addressed in the proposed system, as the sensor fault detection system could be a part of a larger IoT system that requires such fault detection service (e.g. a smart irrigation system). Therefore, the IoT system has to be interoperable.

On the other hand, given the current scenario of climate change [61], the challenges to be overcome by many countries towards the exclusive use of *green energies* [62], the limitations of the IoT itself, such as the low battery life of some IoT devices [63] (as in the case of leaf-turgor pressure sensors), and also the context of our research, energy consumption needs to be optimised as much as possible. For these reasons, the IoT system has to be energy-aware.

Finally, scalability can be defined as the property that ensures that, regardless of the size of the IoT system (number of devices connected to it), the performance of the IoT system will not be negatively affected [10]. Since the IoT system that has been developed could be deployed on small or large agricultural farms, the system has to be scalable.

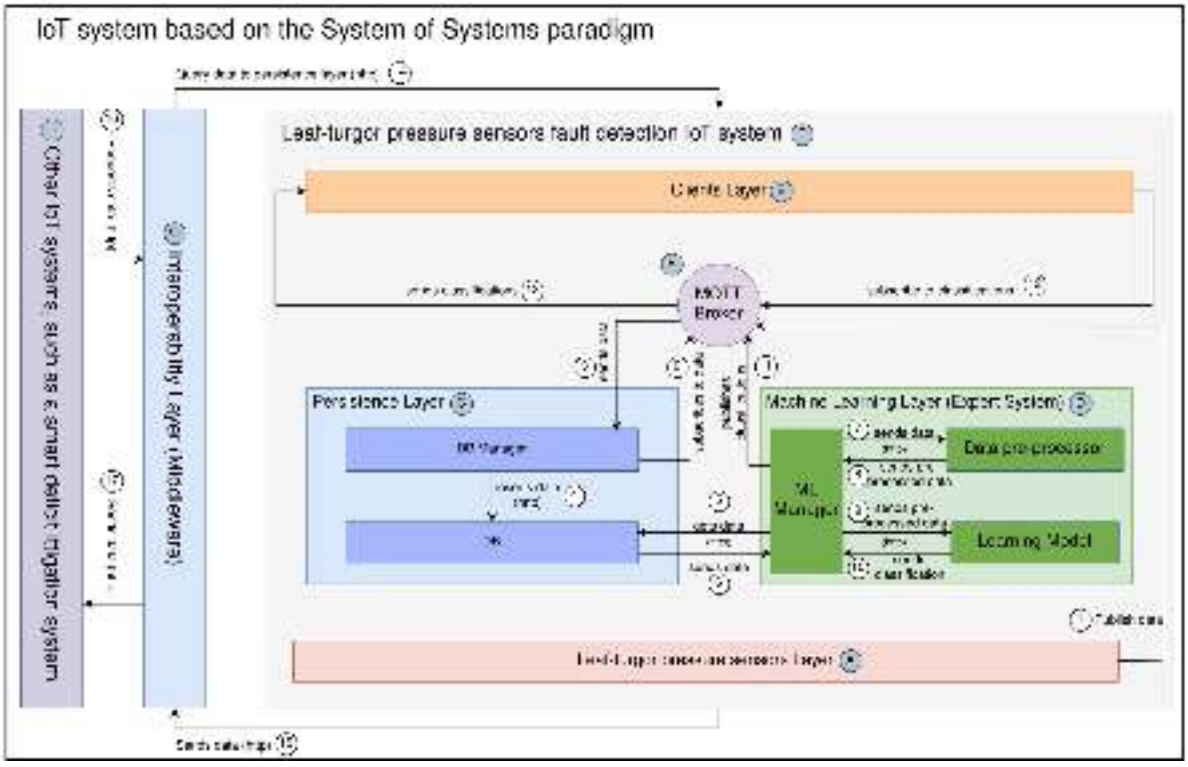


Fig. 13. Software architecture of the IoT-based expert system.

6.3. Iot-based expert system. Design, layers and components

This section presents the proposed IoT system's design, main layers and components. For the sake of clarity, each layer is presented below specifying its aim and components. Thus, layers, their components and interrelationships are described. Note that, to describe the elements better, the numerical and alphabetical labels shown in Fig. 13 are used below as references in the text. These references are used for layers and components by means of the expression `[layer or component name]` Ⓧ, where x is the label associated with the `[layer or component name]` in Fig. 13. References are used for relationships by means of the expression (relationship ⓝ), where n is the label associated with the relationship in Fig. 13, avoiding its name.

- **`Leaf-turgor pressure sensors Layer`** Ⓐ
  Aim. This layer represents the WSN of leaf-turgor pressure sensors deployed on the plum tree farm. The purpose of this layer is to feed measurements into the system, where each individual measurement will be classified as correct or faulty.
  Interrelationships. The sensors from the `Leaf-turgor pressure sensors layer` periodically publish their measurements in the `MQTT Broker` Ⓑ(relationship ①).
- **`MQTT Broker`** Ⓑ
  Aim. The purpose of this component is to allow communication between the other layers and components of the IoT system through the Message Queuing Telemetry Transport (MQTT) protocol, based on the publish–subscribe pattern. In the proposed IoT system in Fig. 13, the message broker that implements the MQTT protocol Eclipse Mosquitto [64] is used.
  Interrelationships. As the central element driving the communication, the `MQTT Broker` has several interrelationships. These interrelationships will be explained below in each component and layer description.
- **`Persistence Layer`** Ⓒ
  Aim. The main purpose of this layer is to provide persistence to all data sent by the `Leaf-turgor pressure sensors layer` Ⓐ. Thus, this layer provides persistence to both, raw and labelled sensor measurements (as correct or incorrect). Also allows other components to query such data.
  Components:
  - `DB Manager`. The aim of this component is to receive and insert into a database all the raw sensor measurements published in the `MQTT Broker` Ⓑby the `Leaf-turgor pressure sen- sors layer` Ⓐ, as well as labelled sensor

measurements published in the MQTT Broker Ⓑby the Machine Learning layer (Expert System) Ⓓ. To do that, first, it subscribes to the topics provided by the MQTT Broker Ⓑwhere the Leaf-turgor pressure sensors layer Ⓐpublish their measurements (relationship ②). Each time a sensor measurement is published in the MQTT Broker Ⓑthe DB Manager receives it (relationship ③) and inserts the sensor reading into the database (MongoDB) as a JSON document (relationship ④). In addition, it also subscribes to the topic of the MQTT Broker Ⓑwhere the Machine Learning layer (Expert System) Ⓓpublish labelled sensor measurements (relationships ②and ⑪), and stores received sensor measurements labelled as correct or incorrect in a different collection in the database (relationship ④).

- **DB**. The purpose of this component is to store sensor measurements and machine learning classifications. Thus, the raw measurements sent by the sensors are stored in one collection, and the measurements labelled as correct or incorrect by the Machine Learning layer (Expert System) Ⓓare stored in a separate collection. In the proposed IoT system in Fig. 13, the NoSQL database MongoDB [65] is used.

- **Machine Learning Layer (Expert System)** Ⓓ
  Aim. The purpose of this layer is to classify and label as correct or incorrect the raw sensor measurements sent by the Leaf-turgor pressure sensors layer Ⓐ, as well as to publish the labelled sensor measurements in the MQTT Broker Ⓑ.
  Components:
  - **Data pre-processor**. The purpose of this component is to perform all the necessary pre-processing techniques to adapt the sensor measurements to the learning models. Thus, reliable and properly formatted data is provided to the learning models. In this way, the learning models are able to classify the sensor measurements as correct or incorrect. Note that the pre-processing techniques applied are those previously discussed in Section 3.3.1.
  - **Learning Model**. This component hosts the best-performing learning model developed in the experiments (Section 4). The aim of this component is to classify the data published by the Leaf-turgor pressure sensors layer Ⓐas correct or incorrect. Thus, through the HyperText Transfer Protocol (HTTP), the service receives the sensors' measurements already pre-processed from the Data pre-processor component. Then, it returns the sensors' measurements together with the classification performed by the model (0=incorrect; 1=correct).
  - **ML Manager**. This is the core component of this layer. Its purpose is to manage the flow of data between the above-described components. Thus, the ML Manager periodically queries the latest measurements from the raw or unlabelled collection of the DB in the Persistence layer Ⓒ(relationships ⑤and ⑥). These raw sensor measurements are sent to the Data Processor component (relationship ⑦) which applies all the necessary transformations and pre-processing techniques to the data and sends it back to the ML Manager (relationship ⑧). Then, the properly pre-processed and formatted data is sent to the Learning Model component (relationship ⑨). In this way, the sensors measurements are classified and the result (labelled measurements) is returned to the ML Manager (relationship ⑩), which publishes it in the MQTT Broker Ⓑ(relationship ⑪).

- **Clients Layer** Ⓔ
  Aim. The purpose of this layer is to represent the target users of the IoT system, who exploit the learning model to detect faults in the leaf-turgor pressure sensors deployed in plum tree farms. Note that target users can be people as well as other computer systems.
  Interrelationships. The Clients layer is subscribed to the classification reports in the MQTT Broker Ⓑ(relationship ⑫). So, every time the Machine Learning layer Ⓓpublishes a new labelled measurement, it will be received by the Clients layer Ⓔ(relationship ⑬). Note that with this approach the target users do not have to worry about the internal logic of the system that works as a black box for them. Once the sensors publish their raw measurements in the appropriate topic of the MQTT Broker Ⓑ, they are only responsible for subscribing to the topic in the MQTT Broker Ⓑwhere the labelled sensor measurements are published.
- **Leaf-turgor pressure sensors fault detection system** Ⓕ
  Aim. This box encompasses all the elements previously described and can be considered as a single entity. It is the IoT-based expert system for fault detection in leaf-turgor pressure sensors that integrates and exploits the best learning model developed in the experiments (Section 4). This system persistently stores the sensor measurements and allows them to be queried, both raw and labelled (classified as correct or incorrect). Thus, it allows third parties applications or users to query such data through the Interoperability layer Ⓖ.
  Interrelationships. Its interrelationships are discussed below, as they are related to the Interoperability layer Ⓖ.
- **Interoperability Layer (Middleware)** Ⓖ
  Aim. The purpose of this layer is to allow any application or user to query and consume sensor records from the Leaf-turgor pressure sensors fault detection system Ⓕ, both raw, i.e. the data directly stored from sensors measurements, and labelled (as correct or incorrect), i.e. the data after the application of the pre-processing techniques and the learning model.
  Interrelationships. Thus, this layer acts as middleware (REST API) that allows any application or user that is not able to use the MQTT protocol to query data from the IoT system's persistent storage (relationships ⑯, ⑭and ⑮). Thus, after receiving a request, the middleware returns such data to the user or application that requested it (relationship ⑰).
- **Other IoT systems** Ⓗ

Aim. This layer represents all IoT systems that could take advantage of the `Leaf-turgor pressure sensor fault detection IoT system`Ⓕthrough the `Interoperability layer (Middleware)` Ⓖ. Note that the `Interoperability layer (Middleware)` Ⓖensures that any third IoT system, regardless of its particular communication protocols and properties, can consume the proposed IoT system for fault detection. An example could be a smart irrigation system, which detects faults automatically without the need for an expert to manually check the sensors.

Finally, note that the proposed system integrates MQTT, which is associated with an event-driven architecture (EDA), with REST functionality. The reason for not adopting a purely EDA or purely REST solution is described below.

An event-driven architecture allows data to be managed by using publish–subscribe communication protocols, which help us to improve internal components' reusability, design low coupling systems and improve aspects such as fault tolerance, energy-awareness and scalability [66–68]. On the other hand, REST APIs make it possible to describe single entry points for service provisioning/consumption, either between system components or for third-party components that could be integrated with the system.

Thus, both paradigms can (should) be used together, using each of them according to the application context in order to take advantage of their strengths. So, in this work, the following application criteria have been followed: EDA if (1) The data does not have a specific recipient, (2) No response is expected from the recipients and (3) The data is not a response to a request. REST if (1) The data has a specific recipient, (2) A response is expected from the recipients or (3) The data is a response to a request.

6.4. Infrastructure of the proposed IoT-based expert system

In this section, the infrastructure of the proposed IoT-based expert system is described. Thus, this section addresses aspects such as the network topology adopted, the protocol stacks used, the hardware employed for each node of the system, the technologies that support the WSN architecture (Fig. 13 Ⓐ) as well as the platforms that support the deployment of the rest of the system, i.e. `Persistence`, `ML` and `Interoperability` layers (Fig. 13, labels Ⓒ, Ⓓand Ⓖrespectively). Note that, for the sake of clarity, a graphical summary of this section is shown in Fig. 14.

Thus, first of all, before describing the infrastructure design of the system, the system model, i.e. the characteristics and assumptions of the system that need to be taken into account throughout this design process, has been defined.

- Scalability is one of the requirements of the system as described in Section 6.2. So, large-scale WSNs are expected.
- The leaf-turgor pressure sensor layer (WSN, Fig. 13 Ⓐ) is deployed in a different location from the `ML` layer (where predictions are made, Fig. 13 Ⓓ). So, the Internet is required to connect the WSN to the Base Station (BS), i.e. the platform on which the `ML` layer is deployed (Microsoft Azure [69]).
- Sensors are only needed at the defined control points (see Fig. 1). Thus, the leaf-turgor pressure sensors are deployed in the same area but grouped in these control points.
- Sensors gather data at a fixed rate (five minutes).
- Sensor fault identification is carried out at a fixed rate (hourly).
- Sensors are homogeneous, i.e. have the same computing power, energy consumption, battery life, processes and data gathering rate. In this regard, note that the sensors' hardware is constrained.
- Internet access is provided through WiFi (IEEE 802.11 [70]).
- Sensors do not have a WiFi card, although they come equipped with a Bluetooth 4.2 LE (BLE) module.
- Energy-awareness is one of the requirements of the system as described in Section 6.2. So, energy-efficient protocols are required.
- Both leaf-turgor pressure sensors and the Base Station (BS), are static after deployment.

Thus, from this system model and studies such as [71] or [72], it has been considered that the most appropriate WSN topology for the proposed system is the Cluster-based topology. In this way, each control point, where sensors are grouped, is managed as a cluster of the WSN.

Concerning the communication between the WSN and the BS, there are several key aspects related to the system model to take into consideration before its design. Firstly, leaf-turgor pressure sensors lack WiFi, which is required to connect to the Internet and send the collected data directly to the BS. Secondly, energy-awareness is crucial in this design stage because of the sensors' hardware constraints and the aim of designing a green IoT system (Section 6.2). Thirdly, sensors come with a BLE module to transmit the gathered data.

Bearing in mind these aspects and analysing BLE features such as its energy-awareness in low-rate data transmissions, in sleep mode and when establishing communication [73,74], it has been considered appropriate to use this technology for the transmission of the collected data.

Then, to enable the sending of the data gathered to the BS, a new device with BLE (in order to collect the data gathered by sensors) and WiFi (in order to connect to the Internet), has to be added to the WSN. According to these requirements, the included device is the RaspBerry Pi 3 B+ [75], which has been added to each cluster of the WSN with the role of Cluster Head (CH). So, every five minutes (sensor gathering rate) the CH collects (via BLE) the data gathered by each sensor belonging to its cluster. Note that to carry out this transmission, the BLE protocol stack defined in [76] is used. On the other hand, once per hour (prediction rate of the expert system), the CHs aggregate the collected data and send it to the BS. As for this transmission, it is performed through

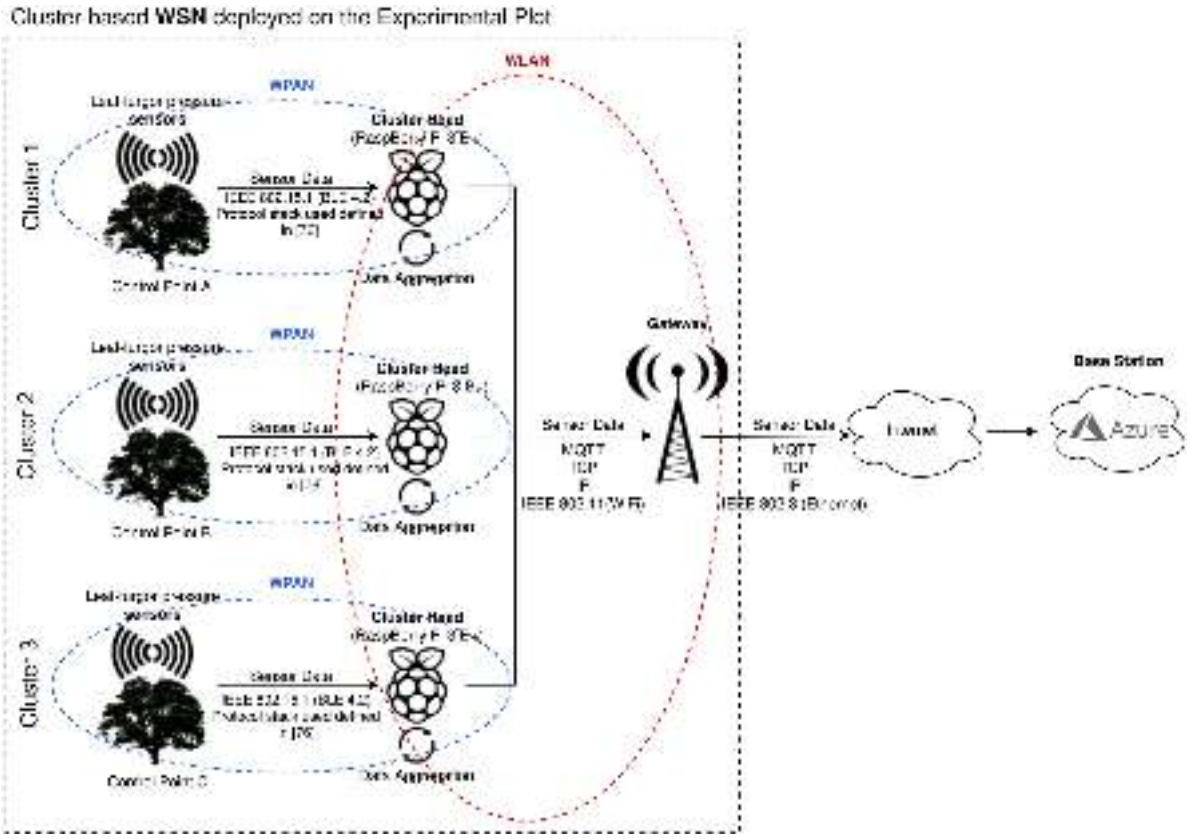


Fig. 14. Infrastructure of the IoT-based expert system.

MQTT, which is applied over the TCP protocol through the Internet (IP protocol). In this regard, according to the aforementioned, the RaspBerries access to the Internet through WiFi.

On the other hand, the infrastructure design of the rest of the system, i.e. the BS, could be summarised in Microsoft Azure and Kubernetes [77]. Thus, each component belonging to the `Persistence`, ML and `Interoperability` layers together with the `MQTT Broker` (Fig. 13, labels Ⓒ, Ⓓ, Ⓖand Ⓑrespectively) are wrapped in a Docker container, deployed on Microsoft Azure and orchestrated by Kubernetes. Regarding the communication between these components, they use both MQTT and HTTP over TCP/IP to interact among them (see Fig. 13). As for the protocols used below TCP/IP, in this case, it will depend on Microsoft Azure.

6.5. Computational complexity analysis of the proposed IoT-based expert system

The computational complexity of a system [78] is usually expressed by the well-known Big-O notation [79]. In short, the Big-O notation represents the worst-case computational complexity of a system by determining how its time requirements increase as the size of the inputs grows and approaches infinity [79].

In this section, an analysis from the Big-O notation perspective about the computational complexity of the proposed IoT-based expert system is carried out. On the one hand, the computational complexity related to the training and running (i.e. make a classification) of the learning models is addressed. On the other hand, the computational complexity of each layer of the proposed IoT-based expert system is assessed. In this case, the impact of each learning model on these components (when deployed together) is also explored.

6.5.1. Computational complexity of the ML algorithms

Firstly, the computational complexity of the learning models is addressed. Although these algorithms belong to the `Machine Learning layer (Expert System)` Ⓓ, they are analysed separately from the rest of the system due to several reasons such as their relevance within the proposed IoT-based expert system, the presence of additional variables that impact its complexity (e.g. the number of support vectors in SVM), and the type of analysis, which is carried out from two different perspectives, training and running complexity.

Thus, Table 10 shows the computational complexity of each ML algorithm applied in this manuscript (see Section 3.3.2). As mentioned above, this complexity is analysed from two perspectives, training and running complexity. The training complexity is

Table 10
Computational complexity of the applied ML algorithms [80,81].

Algorithm	Training complexity	Running complexity
SVM	$\mathcal{O}(n^2 f + n^3)$	$\mathcal{O}(n_{sv} f)$
DT	$\mathcal{O}(n^2 f)$	$\mathcal{O}(f)$
NB	$\mathcal{O}(nf)$	$\mathcal{O}(f)$
LR	$\mathcal{O}(nf)$	$\mathcal{O}(f)$
kNN	$\mathcal{O}(knf)$	$\mathcal{O}(nf)$

Table 11
Computational complexity of the proposed `Leaf-turgor pressure sensors fault detection system` Ⓕ and each of its layers: `Leaf-turgor pressure sensors layer` Ⓐ, `MQTT Broker` Ⓑ, `Persistence layer` Ⓒ, `Machine Learning layer (Expert System)` Ⓓ and `Interoperability layer (Middleware)` Ⓖ.

Layer	Computational complexity
Ⓐ	$\mathcal{O}(n)$
Ⓑ	$\mathcal{O}(n)$
Ⓒ	$\mathcal{O}(nlog(n))$
Ⓓ	$MAX[\mathcal{O}(nf), Learning\ Model\ \mathcal{O}]$
Ⓕ	$MAX[\mathcal{O}(nf), Learning\ Model\ \mathcal{O}]$
Ⓖ	$\mathcal{O}(log(n))$

Table 12
Computational complexity of Machine Learning layer (Expert System).

`Machine Learning layer (Expert System)` Ⓓ		
$MAX[\mathcal{O}(nf), Learning\ Model\ \mathcal{O}]$		
`ML Manager`	`Data pre-processor`	`Learning Model`
$MAX[\mathcal{O}(nf), Learning\ Model\ \mathcal{O}]$	$\mathcal{O}(nf)$	Depending on the algorithm (see Table 10)

related to the time required for the ML algorithms to train the classification models. On the other hand, the running complexity is related to the time required to perform classifications using these models once they have been trained.

Regarding the variables used to determine the computational complexity of these ML algorithms, n is the number of instances that form the training dataset, f is the number of features of each instance, k is the number of neighbours specified in kNN and n_{sv} is the number of support vectors specified in SVM. Note that these variables represent the inputs to the ML algorithms, whose growth affects the computational complexity of the training and running processes of the learning models. Finally, it should be noted that the computational complexity of these ML algorithms has already been addressed in the literature [80,81].

6.5.2. Introduction to the computational complexity of the iot-based expert system and notation used

The computational complexity of an algorithm in terms of Big-O notation can be determined by analysing its code. Specifically, from the structures that scale in number of operations as the size of the input variables grows, i.e. loops and nested loops [79,82]. This analysis has been carried out for each component that forms the proposed IoT-based expert system. Table 11 shows the computational complexity determined for each system layer. To carry out this analysis, a Big-O notation similar to that used in Section 6.5.1 has been applied. Nevertheless, the concept that the variables represent changes depending on the context (Layer) in which they are applied. So, the notation used and its meaning in each case is presented below.

In this section, $\mathcal{O}(f)$ is equal to $\mathcal{O}(mt)$, where m represents the number of measured magnitudes (e.g. leaf-turgor pressure and leaf temperature) and t represents the number of time steps considered (e.g. 24 h). Note that $\mathcal{O}(mt)$ is equal to $\mathcal{O}(f)$ since the number of features (f) in each time series is the number of measured magnitudes (m) multiplied by the number of time steps (t). On the other hand, sensors gather data at a fixed gathering rate (e.g. every five minutes). Therefore, if sensors grow, the amount of data gathered grows in the same proportion. Moreover, the number of time series that can be generated grows in the same proportion as both. As these three variables (number of sensors, number of recorded measurements and number of final time series) grow in the same proportion, they can be reduced to the same variable n, which represents all three interchangeably.

6.5.3. Computational complexity of the Machine Learning layer

Table 12 shows the computational complexity of the `Machine Learning layer` Ⓓ. To determine this computational complexity, the computational complexity of each of the components of this layer has been assessed.

Firstly, the computational complexity of the `Data Preprocessor` grows as the number of monitored magnitudes (m) and the number of time steps considered (t) for each record to preprocess (n) increases $\mathcal{O}(nmt) = \mathcal{O}(nf)$. Note that the computational complexity of this component is not affected by the learning models, as the required pre-processing techniques are the same for all of them.

Secondly, as the `Learning Model` component represents the selected learning model to be deployed as part of the IoT-based expert system, its computational complexity is the complexity of the selected learning model (see Table 10).

Thirdly, regarding the `ML Manager` component, it manages the flow of the data in the `Machine Learning layer` Ⓓ. So, it requests periodically the data gathered by the sensors from the DB. Later, it sends this data to the `Data Preprocessor`. Once the preprocessed data is received, it requests the `Learning Model` to get the classifications and publish them to the `MQTT Broker` Ⓑ. Therefore, the complexity of this component is determined by the component with the highest complexity on which it depends.

In this regard, note that only the ML layer is affected by the learning model selected to be deployed as part of the IoT-based expert system.

6.5.4. Computational complexity of the Persistence layer

The purpose of the `Persistence layer` Ⓒis to provide persistence to both, raw and labelled sensor measurements (as correct or incorrect). In this regard, note that to optimise the insertion and retrieval of the data, a date-based index has been included in the DB. Therefore, the number of operations required to insert or retrieve a measurement (by means of a dichotomous search by date) will grow in a logarithmic order, i.e. $\mathcal{O}(log(n))$. So, as the tasks of the DB `Manager` are limited to inserting and retrieving data by date from the database, the computational complexity of this component is $\mathcal{O}(nlog(n))$.

6.5.5. Computational complexity of the Leaf-turgor pressure sensors layer

The `Leaf-turgor pressure sensors layer` Ⓐrepresents the WSN deployed in the experimental plot, which is composed of leaf-turgor pressure sensors clusters and Cluster Heads. Thus, each cluster sends its measurements to its Cluster Head, which aggregates and publishes them to the `MQTT Broker` Ⓑ. Therefore, the amount of operations that this layer has to perform grows linearly with the number of sensors (n) deployed. So, the time complexity of the layer is $\mathcal{O}(n)$.

6.5.6. Computational complexity of the Interoperability layer

The `Interoperability layer` Ⓖacts as a middleware (REST API) that allows any application or user to request data from the IoT system's persistent storage. Since the DB has a complexity of $\mathcal{O}(log(n))$, the computational complexity of this layer is $\mathcal{O}(log(n))$.

6.5.7. Computational complexity of the MQTT Broker

Although the MQTT Broker is not a system layer, it is an essential element of the system. So, its computational complexity is addressed in this section.

Thus, the computational complexity of the `MQTT Broker` Ⓑdepends on which has been used. In this work, the Eclipse Mosquitto [64] has been selected and its computational complexity has already been analysed in the literature from several perspectives, such as the growth of response times as the size of the message payload increases [83], as well as the number of publishers increases [84]. In both cases, a linear growth relationship between response time and these magnitudes is identified, i.e. $\mathcal{O}(n)$.

In the proposed system, both the number of publishers could vary (n sensors deployed) and the message payload (amount of data aggregated by each Cluster Head). As the complexity in both cases grows linearly and they involve independent processes, the complexity of the MQTT Broker is $\mathcal{O}(n)$.

Finally, note that the `Machine Learning layer` Ⓓis the component that presents the highest computational complexity. Therefore, the overall computational complexity of the system is determined by the complexity of this component. In this regard, it is crucial to take into account that the computational complexity of this layer varies depending on the selected learning model. On the other hand, also note that the complexity of the client layer has not been analysed. This is because the client layer could involve any type of device or system that consumes the proposed expert system.

6.6. Discussion about the proposed IoT-based expert system

The proposed system is an energy-aware, scalable and interoperable IoT system based on the system of systems paradigm [85]. Given that the previous section did not address how these open challenges have been specifically addressed, this section discusses how these IoT open challenges have been reached with the software architecture of the system and its components.

Regarding the energy awareness of the system, communication is considered one of the key factors in terms of sensor energy consumption [86]. In this sense, the MQTT communication protocol has been chosen since it is one of the most widespread and energy-efficient protocols in the IoT [87]. On the other hand, the number of sensors' publications has been considerably reduced. Before the training of the learning model, the sensors belonging to the sensor layer Ⓐwere configured to publish their data every two or three minutes. However, after the experiments were carried out, the trained learning model showed that it is able to perform with only one measurement from each sensor per hour. This represents a decrease of around 95% in the data that sensors have to

gather and publish, implying significant energy savings. Besides, it also implies a reduction in the use of the bandwidth of the IoT system, which also saves energy.

Regarding scalability, the main elements of the proposed system are designed to be scaled. The `Leaf-turgor pressure layer` Ⓐcan be scaled by simply adding more sensors to it. The `Persistence layer` Ⓒis composed of MongoDB, a NoSQL database designed to be scalable [88]. Concerning the `Machine learning layer` Ⓓ, both the `Data pre-processor` and the `Learning model` components can be replicated, being the `ML manager` which would act as a load balancer in this scenario. Finally, the `Interoperability layer` Ⓖcan be also replicated.

Finally, regarding interoperability, the proposed system is designed following the system of systems paradigm. In this way, the `Leaf-turgor pressure sensors fault detection IoT system` Ⓕcan be integrated or consumed as a service by third-party applications or systems through the `Interoperability layer` Ⓖ. So, third-party systems only have to request the services (faults identified, whole labelled dataset, whole raw dataset, etc.) they need from this layer Ⓖand pre-process the response (if needed) to later consume it.

7. Discussion

Precision irrigation techniques are a vehicle for the sustainable use of water in agriculture. In this regard, techniques such as deficit irrigation techniques need precise knowledge of crop water stress to be suitably applied. Proposals such as [9] provide these precision irrigation techniques with the water stress of crops. However, such solutions are IoT-based and often involve sensors. So, it is mandatory to identify sensor faults since they could compromise the application of precision irrigation techniques, negatively impacting crop yield. Besides, in the context of leaf-turgor pressure sensors, this process is carried out manually by experts. So, it is tedious, time-consuming and costly.

With the aim of improving the digital transformation by using and automating this process in the context of leaf-turgor pressure sensors, this communication presents as main contribution a learning model that is able of identifying leaf-turgor pressure sensor faults with an 84.2% f1-score and 0.94 AUC ROC. According to the experts' opinion, these results imply the feasibility of replacing them in order to carry out this process. Therefore, using this model, manual error identification can be avoided, thus reducing the effort and costs associated with this task. Besides, the digital transformation of this process also increases the feasibility and scalability of proposals such as [9], since they no longer require an expert in terms of fault identification to be successfully applied. However, taking into account that the performance of the learning model is not able to identify 100% if a sensor fails, to achieve the optimum crop yield it is recommended that experts manually inspect the correct status of sensors occasionally. Nevertheless, in this scenario, the workload related to this task has been also substantially reduced.

On the other hand, the training of this model has involved dealing with several challenges. So, it has been considered interesting to develop a methodology taking into account these difficulties. In this regard, this methodology could save effort, time and money for those facing a similar problem. However, it should be noted that the usability of this methodology depends on the scope of the problem to address and its similarity with the problem addressed in this communication.

This communication also presents an IoT system that integrates the fault identification learning model. In this way, it illustrates how the learning model should be deployed and applied by showing its role in the system, interactions with other components of the system, required components for its operation, etc. Furthermore, the design of the IoT system is carried out bearing in mind aspects such as scalability, interoperability and energy awareness. Thus, the use and applicability of the learning model are shown, encouraging those who can benefit from it to integrate it into their systems. Besides, in a simple and sustainable way, due to its interoperability and energy-awareness properties.

Finally, to the best of the authors' knowledge, the related works to date on fault detection in WSNs are generalist and theoretical, i.e., they attempt to detect faults synthetically injected by the authors themselves in the sensor data. A relevant contribution of this paper is that the findings of these related works have been shown to be successfully applied to a real use case, thus corroborating the applicability of the techniques and knowledge achieved by the related works in a real-world application case with real, non-synthetically injected faults. In this way, further work dealing with fault detection in real WSNs can be more reliably supported by these related works.

Nevertheless, the contributions of this communication enable, through technologies such as IoT and ML, the digital transformation and automation of fault detection of leaf-turgor pressure sensors deployed in plum tree farms. Furthermore, both the proposed methodology and the validation of the applicability of the findings of more theoretical studies in a real use case scenario could reduce the efforts and costs of solving similar problems.

As for the limitations of this work, the main limitation is the lack of data from plum tree crops grown in different climates. In different climates, the behaviour of the leaf-turgor pressure curves could vary. So, the performance of the trained learning models might be reduced if they are applied in this context. Nevertheless, even though applied in this scenario, no significant decrease in performance is expected, but some uncertainty is expected in those faults that are more difficult for learning models to identify. On the other hand, another limitation of this work is that it is not possible to state that this proposal is valid for other types of crops. Although the pressure curves of other crops are similar to those of plum trees, no validation tests have been carried out in this respect. Thus, further research is needed in this matter.

8. Conclusions and future works

This section presents the conclusions, as well as some possible lines of future research that can be explored to improve or complement the study reported in this paper.

8.1. Conclusions

Some precision irrigation techniques use input data coming from or based on data gathered by sensors. However, these sensors could fail, leading to an inaccurate application of these techniques. As this scenario could compromise crop yield, identifying faults in these sensors is a must.

Currently, in the context of leaf-turgor pressure sensors, the identification of sensor faults is carried out manually by experts. This process is tedious, time-consuming and costly. So, in order to avoid it, this communication has presented a learning model able to identify leaf-turgor pressure sensor faults with an 84.2% f1-score and a 0.94 AUC ROC, a suitable performance to replace the manual process. Besides, the key aspects related to the experiments carried out to train this model have been identified, thus developing a methodology from them that could improve the investment of time, money and efforts to train models of this scope.

On the other hand, an IoT system that integrates and illustrates the practical use of this learning model has been presented. Furthermore, this system has been designed bearing in mind key aspects such as interoperability, scalability or energy awareness, thus simplifying its integration with other systems and making sustainable use of energy.

Finally, it has been shown that the findings found in more theoretical works (use of artificial data) concerning the identification of sensor faults are applicable to real-world approaches.

8.2. Future works

The main future works under consideration are outlined below:

- Test the performance of the ML models presented in this article for the detection of leaf-turgor pressure sensor faults in a Japanese plum-tree farm, on sensors deployed in other crops, such as almond, mandarin or orange trees.
- If the same performance is not achieved in other crops, generate crop-specific models from the leaf-turgor pressure sensor data of each crop and its faults, in order to optimise the faults detection task.
- Test the performance of the ML models presented in this article with sensor data from different years. Since the models have only been trained with data from one year's fruit ripening cycle.
- Integrate the IoT system for fault detection of leaf-turgor pressure sensors with other IoT systems. For instance, smart irrigation IoT systems in CICYTEX's Japanese plum tree farms.
- Automate the normalisation of data taking into account range shifts. This is a time and effort-consuming manual task, as charts of the measured magnitude (e.g. leaf-turgor pressure as shown in Fig. 10) have to be displayed and analysed to identify the time instants at which range shift occurs. Automating this task would speed up the development times of many similar works, as well as reduce the possibility of errors, optimising an essential phase of the methodology proposed in Section 5.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgements

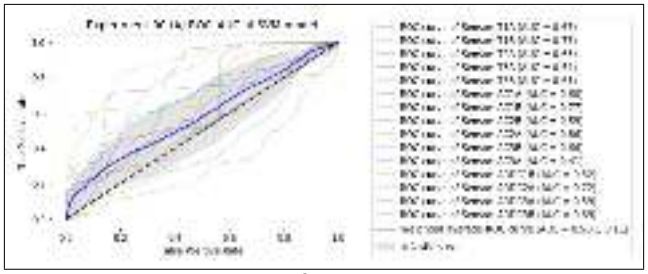
JUNTA DE EXTREMADURA

To Robert BOSCH España S.L.U. for providing the leaf-turgor pressure sensors.

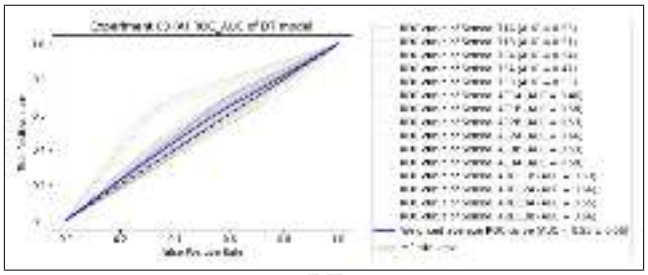
Appendix

AUC ROC charts showing the performance of each sensor throughout the cross-validation process.

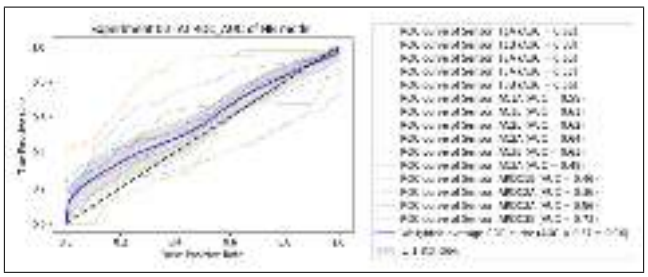
See Figs. 15–19.



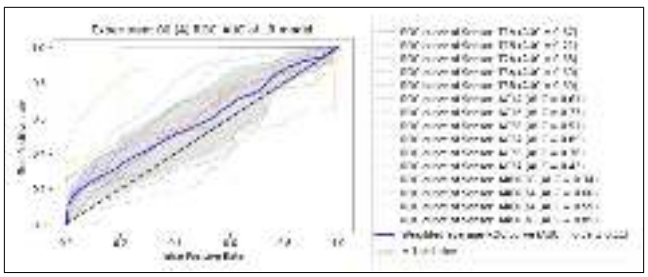
SVM



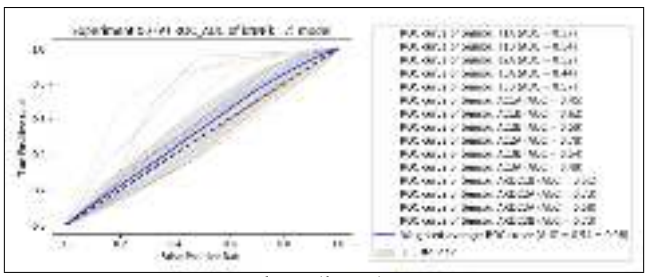
DT



NB

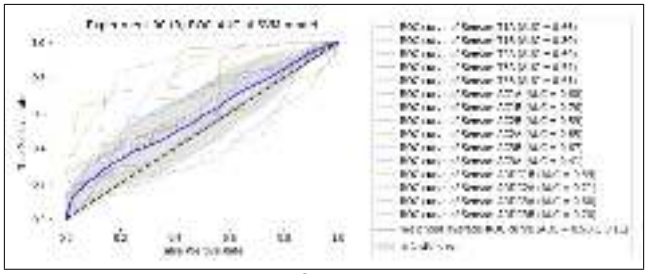


LR

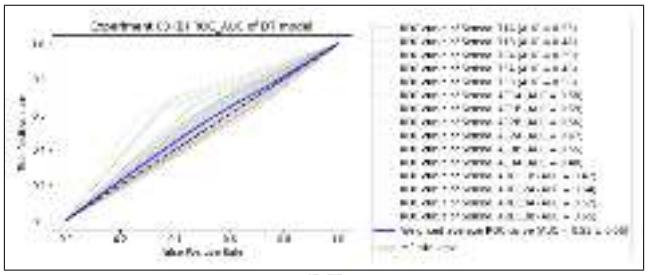


$kNN(k = 7)$

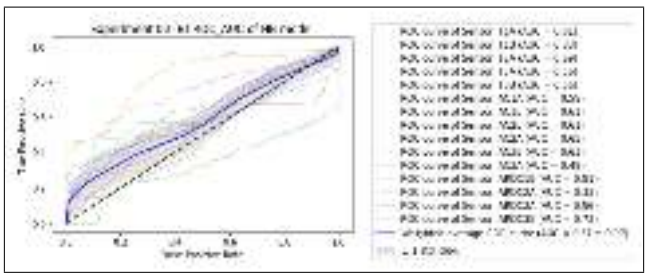
Fig. 15. ROC curve and AUC for the machine learning models developed in Experiment 00 (A).



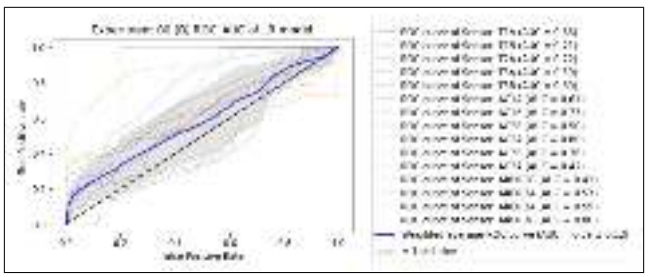
SVM



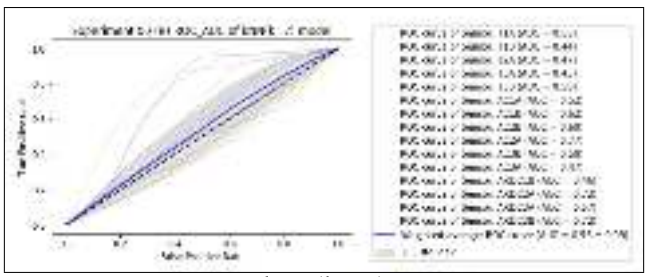
DT



NB

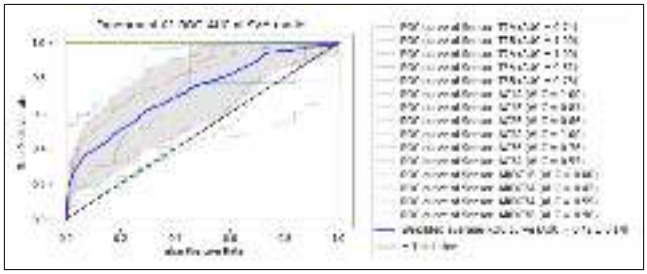


LR

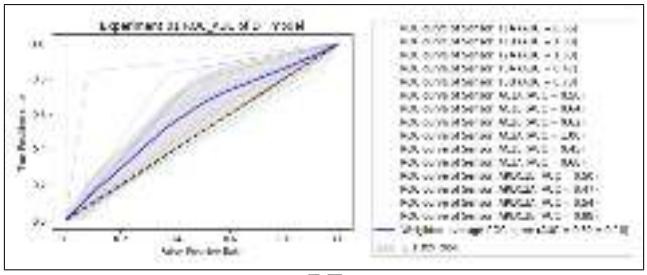


$kNN(k = 7)$

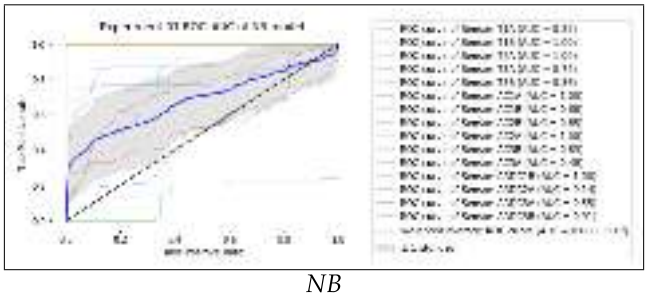
Fig. 16. ROC curve and AUC for the machine learning models developed in Experiment 00 (B).



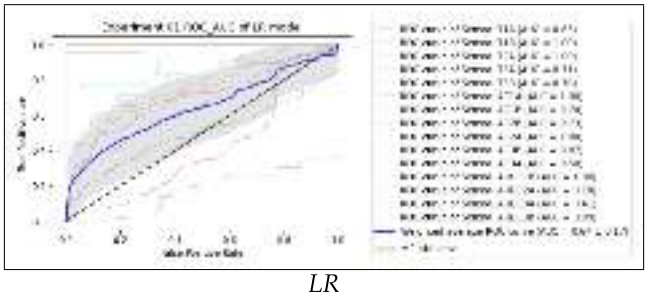
SVM



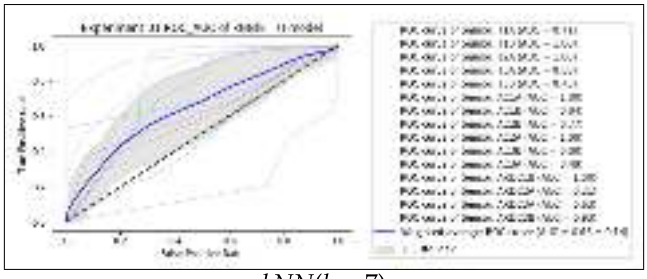
DT



NB

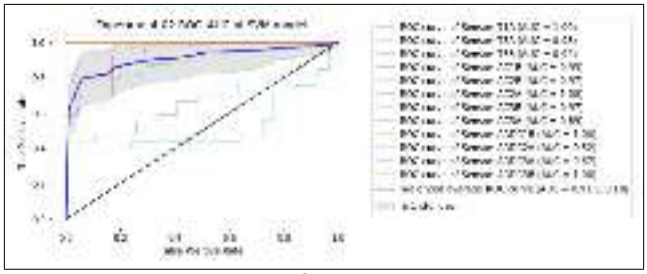


LR

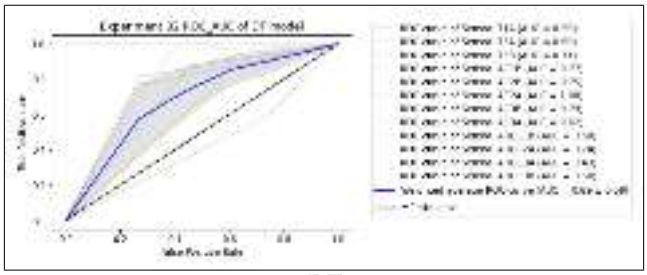


kNN($k = 7$)

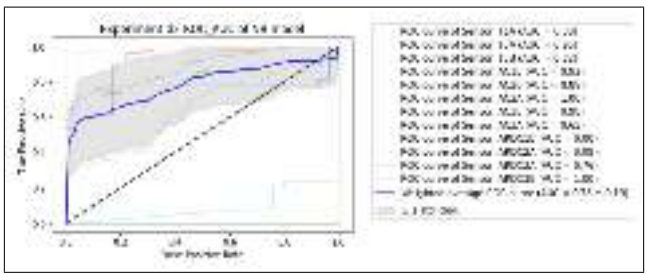
Fig. 17. ROC curve and AUC for the machine learning models developed in Experiment 01.



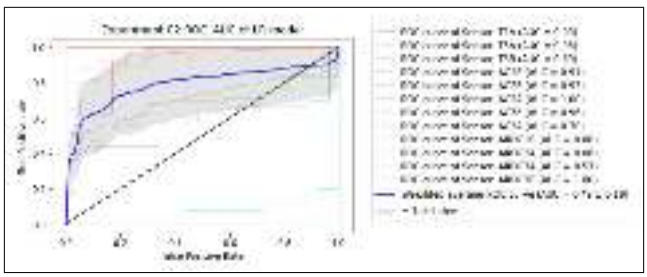
SVM



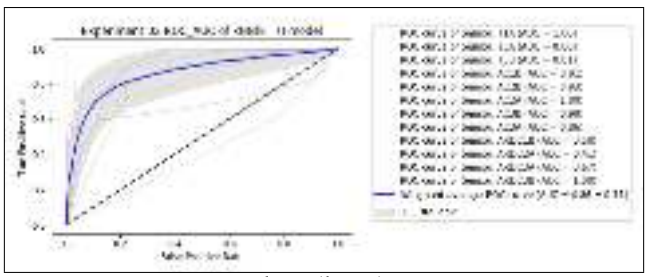
DT



NB

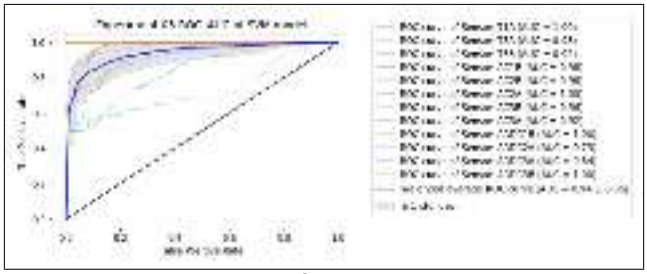


LR

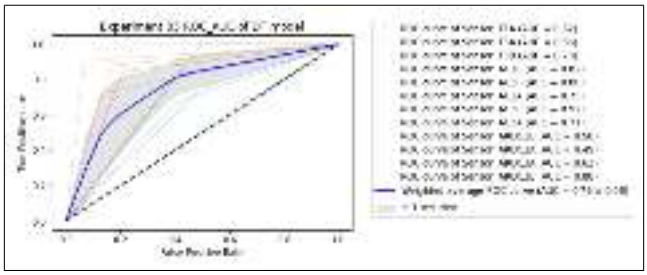


$kNN(k = 7)$

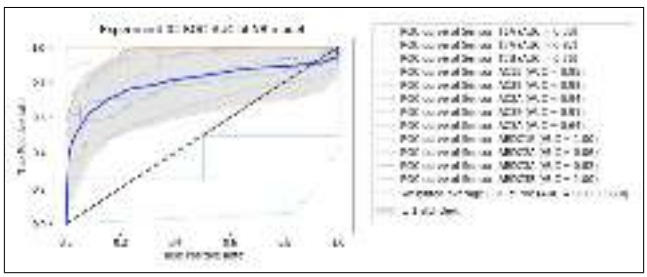
Fig. 18. ROC curve and AUC for the machine learning models developed in Experiment 02.



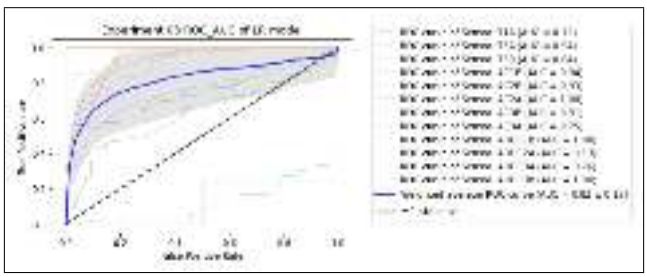
SVM



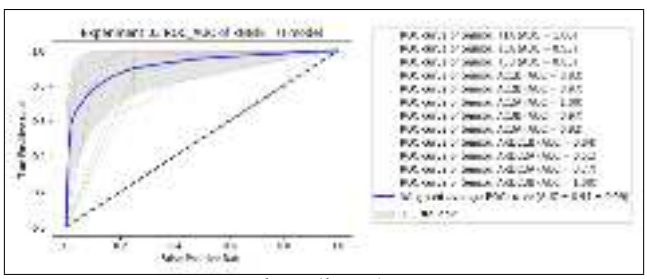
DT



NB



LR



$kNN(k = 7)$

Fig. 19. ROC curve and AUC for the machine learning models developed in Experiment 03.

References

[1] Bernard Ijesunor Akhigbe, Kamran Munir, Olugbenga Akinade, Lukman Akanbi, Lukumon O. Oyedele, IoT technologies for livestock management: A review of present status, opportunities, and future trends, Big Data Cogn. Comput. 5 (1) (2021) 10.

[2] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Tariq Umer, Yousaf Bin Zikria, Role of IoT technology in agriculture: A systematic literature review, Electronics 9 (2) (2020) 319.

[3] Baofeng Ji, Xueru Zhang, Shahid Mumtaz, Congzheng Han, Chunguo Li, Hong Wen, Dan Wang, Survey on the internet of vehicles: Network architectures and applications, IEEE Commun. Stand. Mag. 4 (1) (2020) 34–41.

[4] Roy Woodhead, Paul Stephenson, Denise Morrey, Digital construction: From point solutions to IoT ecosystem, Autom. Constr. 93 (2018) 35–46.

[5] Hans-Otto Pörtner, Debra C. Roberts, H. Adams, C. Adler, P. Aldunce, E. Ali, R. Ara Begum, R. Betts, R. Bezner Kerr, R. Biesbroek, et al., Climate Change 2022: Impacts, Adaptation and Vulnerability, IPCC Sixth Assessment Report, IPCC Geneva, Switzerland, 2022.

[6] Juan F. Velasco-Muñoz, José A. Aznar-Sánchez, Luis J. Belmonte-Ureña, Isabel M. Román-Sánchez, Sustainable water use in agriculture: A review of worldwide research, Sustainability 10 (4) (2018) 1084.

[7] Jaime Espinosa-Tasón, Evolución de la gestión del regadío en España y sus implicaciones ante la escasez del agua, 2022, Universidad de Córdoba, UCOPress.

[8] Iván García-Tejero, Victor Hugo Durán-Zuazo, Javier Arriaga-Sevilla, José Luis Muriel-Fernández, Impact of water stress on Citrus yield, Agron. Sustain. Dev. 32 (3) (2012) 651–659.

[9] Jose A. Barriga, Fernando Blanco-Cipollone, Emiliano Trigo-Córdoba, Iván García-Tejero, Pedro J. Clemente, Crop-water assessment in Citrus (*Citrus sinensis L.*) based on continuous measurements of leaf-turgor pressure using machine learning and IoT, Expert Syst. Appl. (2022) 118255.

[10] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, Mohsen Guizani, Internet of Things architecture: Recent advances, taxonomy, requirements, and open challenges, IEEE Wirel. Commun. 24 (3) (2017) 10–16, http://dx.doi.org/10.1109/MWC.2017.1600421.

[11] Sana Ullah Jan, Young Doo Lee, In Soo Koo, A distributed sensor-fault detection and diagnosis framework using machine learning, Inform. Sci. 547 (2021) 777–796.

[12] Milica Pejanović Đurišić, Zhilbert Tafa, Goran Dimić, Veljko Milutinović, A survey of military applications of wireless sensor networks, in: 2012 Mediterranean Conference on Embedded Computing, MECO, IEEE, 2012, pp. 196–199.

[13] Ahsan Adeel, Mandar Gogate, Saadullah Farooq, Cosimo Ieracitano, Kia Dashtipour, Hadi Larijani, Amir Hussain, A survey on the role of wireless sensor networks and IoT in disaster management, in: Geological Disaster Monitoring Based on Sensor Networks, Springer, 2019, pp. 57–66.

[14] D. Arjun, P.K. Indukala, K.A. Unnikrishna Menon, Border surveillance and intruder detection using wireless sensor networks: A brief survey, in: 2017 International Conference on Communication and Signal Processing, ICCSP, IEEE, 2017, pp. 1125–1130.

[15] Muhammad Irfan, Husnain Jawad, Barkoum Betra Felix, Saadullah Farooq Abbasi, Anum Nawaz, Saeed Akbarzadeh, Muhammad Awais, Lin Chen, Tomi Westerlund, Wei Chen, Non-wearable IoT-based smart ambient behavior observation system, IEEE Sens. J. 21 (18) (2021) 20857–20869.

[16] Cristiano André Da Costa, Cristian F. Pasluosta, Björn Eskofier, Denise Bandeira Da Silva, Rodrigo da Rosa Righi, Internet of health things: Toward intelligent vital signs monitoring in hospital wards, Artif. Intell. Med. 89 (2018) 61–69.

[17] Jinran Chen, Shubha Kher, Arun Somani, Distributed fault detection of wireless sensor networks, in: Proceedings of the 2006 Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks, 2006, pp. 65–72.

[18] Peng Jiang, A new method for node fault detection in wireless sensor networks, Sensors 9 (02) (2009) 1282–1294.

[19] Thaha Muhammed, Riaz Ahmed Shaikh, An analysis of fault detection strategies in wireless sensor networks, J. Netw. Comput. Appl. 78 (2017) 267–287.

[20] Salah Zidi, Tarek Moulahi, Bechir Alaya, Fault detection in wireless sensor networks through SVM classifier, IEEE Sens. J. 18 (1) (2017) 340–347.

[21] Zainib Noshad, Nadeem Javaid, Tanzila Saba, Zahid Wadud, Muhammad Qaiser Saleem, Mohammad Eid Alzahrani, Osama E. Sheta, Fault detection in wireless sensor networks through the random forest classifier, Sensors 19 (7) (2019) 1568.

[22] Ye Yuan, Shouzheng Li, Xingjian Zhang, Jianguo Sun, A comparative analysis of SVM, naive bayes and GBDT for data faults detection in WSNs, in: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C, IEEE, 2018, pp. 394–399.

[23] Dahai Zhang, Liyang Qian, Baijin Mao, Can Huang, Bin Huang, Yulin Si, A data-driven design for fault detection of wind turbines using random forests and XGboost, Ieee Access 6 (2018) 21020–21031.

[24] Xiaohang Jin, Tommy W.S. Chow, Yi Sun, Jihong Shan, Bill C.P. Lau, Kuiper test and autoregressive model-based approach for wireless sensor network fault diagnosis, Wirel. Netw. 21 (3) (2015) 829–839.

[25] Jiří Anděl, Autoregressive series with random parameters, Math. Oper.forsch. Stat. 7 (5) (1976) 735–741.

[26] A.S. Louter, J. Koerts, On the Kuiper test for normality with mean and variance unknown, Stat. Neerl. 24 (2) (1970) 83–87.

[27] Frank J. Massey Jr., The Kolmogorov-Smirnov test for goodness of fit, J. Amer. Statist. Assoc. 46 (253) (1951) 68–78.

[28] Saray Gutiérrez-Gordillo, Leontina Lipan, Víctor Hugo Durán Zuazo, Esther Sendra, Francisca Hernández, Martín Samuel Hernández-Zazueta, Ángel A. Carbonell-Barrachina, Iván Francisco García-Tejero, Deficit irrigation as a suitable strategy to enhance the nutritional composition of hydrosos almonds, Water 12 (12) (2020) 3336.

[29] Sven Crone, Stefan Lessmann, Robert Stahlbock, The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing, European J. Oper. Res. (2006) 781–800, http://dx.doi.org/10.1016/j.ejor.2005.07.023.

[30] Alper Kursat Uysal, Serkan Gunal, The impact of preprocessing on text classification, Inf. Process. Manage. 50 (1) (2014) 104–112.

[31] Roweida Mohammed, Jumanah Rawashdeh, Malak Abdullah, Machine learning with oversampling and undersampling techniques: Overview study and experimental results, in: 2020 11th International Conference on Information and Communication Systems, ICICS, IEEE, 2020, pp. 243–248.

[32] S. Barua, M.M. Islam, X. Yao, K. Murase, MWMOTE–Majority weighted minority oversampling technique for imbalanced data set learning, IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 405–425, http://dx.doi.org/10.1109/TKDE.2012.232.

[33] Aida Ali, Siti Mariyam Shamsuddin, Anca L. Ralescu, Classification with class imbalance problem, Int. J. Adv. Soft Comput. Appl. 5 (3) (2013).

[34] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, Ce Zhang, Cleanml: A benchmark for joint data cleaning and machine learning [experiments and analysis], 2019, p. 75, arXiv preprint arXiv:1904.09483.

[35] S. Gopal Krishna Patro, Kishore Kumar Sahu, Normalization: A preprocessing stage, 2015, CoRR abs/1503.06462.

[36] Jason Brownlee, Introduction to Time Series Forecasting with Python: How to Prepare Data and Develop Models to Predict the Future, Machine Learning Mastery, 2017.

[37] Miroslav Kubat, An Introduction to Machine Learning, Springer, 2017.

[38] Bernhard E. Boser, Isabelle M. Guyon, Vladimir N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, 1992, pp. 144–152.

[39] William S. Noble, What is a support vector machine? Nature Biotechnol. 24 (12) (2006) 1565–1567.

[40] John D. Kelleher, Brian Mac Namee, Aoife D'arcy, Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies, MIT Press, 2020.

[41] K. Ming Leung, Naive Bayesian Classifier, Vol. 2007, Polytechnic University Department of Computer Science/Finance and Risk Engineering, 2007, pp. 123–156.

[42] Daniel Berrar, Bayes' Theorem and Naive Bayes Classifier. Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics, Vol. 403, Elsevier Science Publisher, Amsterdam, the Netherlands, 2018, p. 412.
[43] Bahzad Charbuty, Adnan Abdulazeez, Classification based on decision tree algorithm for machine learning, J. Appl. Sci. Technol. Trends 2 (01) (2021) 20–28.
[44] Abdallah Bashir Musa, Comparative study on classification performance between support vector machine and logistic regression, Int. J. Mach. Learn. Cybern. 4 (1) (2013) 13–24.
[45] Daniel Berrar, Performance Measures for Binary Classification, Elsevier, 2019.
[46] Kinam Park, Youngrok Song, Yun-Gyung Cheong, Classification of attack types for intrusion detection systems using a machine learning algorithm, in: 2018 IEEE Fourth International Conference on Big Data Computing Service and Applications, BigDataService, IEEE, 2018, pp. 282–286.
[47] David Martin Powers, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation, in: Informedness, Markedness and Correlation. Journal of Machine Learning, Bioinfo Publications, 2011.
[48] Joakim Ekström, The phi-coefficient, the tetrachoric correlation coefficient, and the pearson-yule debate, 2011.
[49] Margherita Grandini, Enrico Bagli, Giorgio Visani, Metrics for multi-class classification: An overview, 2020, arXiv preprint arXiv:2008.05756.
[50] Jacob Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Measur. 20 (1) (1960) 37–46.
[51] Daniel Berrar, Peter Flach, Caveats and pitfalls of ROC analysis in clinical microarray research (and how to avoid them), Brief. Bioinform. 13 (1) (2012) 83–97.
[52] Yu Feng, Yong Peng, Ningbo Cui, Daozhi Gong, Kuandi Zhang, Modeling reference evapotranspiration using extreme learning machine and generalized regression neural network only with temperature data, Comput. Electron. Agric. 136 (2017) 71–78.
[53] Amit Prakash Patil, Paresh Chandra Deka, An extreme learning machine approach for modeling evapotranspiration using extrinsic inputs, Comput. Electron. Agric. 121 (2016) 385–392.
[54] Kasra Mohammadi, Shahaboddin Shamshirband, Shervin Motamedi, Dalibor Petković, Roslan Hashim, Milan Gocic, Extreme learning machine based prediction of daily dew point temperature, Comput. Electron. Agric. 117 (2015) 214–225.
[55] Z. Reitermanova, Feedforward neural networks–architecture optimization and knowledge extraction, in: WDS'08 Proceedings of Contributed Papers, 2008, pp. 159–164.
[56] Zuzana Reitermanova, Data splitting, in: WDS, Vol. 10, 2010, pp. 31–36.
[57] Daniel Berrar, Cross-validation, 2019.
[58] Petro Liashchynskyi, Pavlo Liashchynskyi, Grid search, random search, genetic algorithm: A big comparison for NAS, 2019, arXiv preprint arXiv:1912.06059.
[59] Sarang Narkhede, Understanding AUC-ROC curve, Towards Data Sci. 26 (1) (2018) 220–227.
[60] Mayuri S. Shelke, Prashant R. Deshmukh, Vijaya K. Shandilya, A review on imbalanced data handling using undersampling and oversampling technique, Int. J. Recent Trends Eng. Res 3 (4) (2017) 444–449.
[61] Paola Arias, Nicolas Bellouin, Erika Coppola, Richard Jones, Gerhard Krinner, Jochem Marotzke, Vaishali Naik, Matthew Palmer, G.-K. Plattner, Joeri Rogelj, et al., Climate Change 2021: The Physical Science Basis, Contribution of Working Group14 I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change; Technical Summary, 2021.
[62] Campos Inês, Pontes Luz Guilherme, Marín-González Esther, Gährs Swantje, Hall Stephen, Holstenkamp Lars, Regulatory challenges and opportunities for collective renewable energy prosumers in the EU, Energy Policy 138 (2020) 111212.
[63] Cheng Zhuo, Shaoheng Luo, Houle Gan, Jiang Hu, Zhiguo Shi, Noise-aware DVFS for efficient transitions on battery-powered IoT devices, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (7) (2019) 1498–1510.
[64] The MosquittoTeam, Mosquitto, 2022.
[65] MongoDB, MongoDB named as a leader in the forrester wave™: Translytical data platforms, Q4 2022, 2022.
[66] Biswajeeban Mishra, Attila Kertesz, The use of MQTT in M2M and IoT systems: A survey, IEEE Access 8 (2020) 201071–201086.
[67] Tetsuya Yokotani, Yuya Sasaki, Comparison with HTTP and MQTT on required network resources for IoT, in: 2016 International Conference on Control, Electronics, Renewable Energy and Communications, ICCEREC, IEEE, 2016, pp. 1–6.
[68] Bharati Wukkadada, Kirti Wankhede, Ramith Nambiar, Amala Nair, Comparison with HTTP and MQTT in Internet of Things (IoT), in: 2018 International Conference on Inventive Research in Computing Applications, ICIRCA, IEEE, 2018, pp. 249–253.
[69] Google, Kubernetes, 2023.
[70] Guido R. Hiertz, Dee Denteneer, Lothar Stibor, Yunpeng Zang, Xavier Pérez Costa, Bernhard Walke, The IEEE 802.11 universe, IEEE Commun. Mag. 48 (1) (2010) 62–70.
[71] Amin Shahraki, Amir Taherkordi, Øystein Haugen, Frank Eliassen, A survey and future directions on clustering: From WSNs to IoT and modern networking paradigms, IEEE Trans. Netw. Serv. Manag. 18 (2) (2021) 2242–2274, http://dx.doi.org/10.1109/TNSM.2020.3035315.
[72] Quazi Mamun, A qualitative comparison of different logical topologies for wireless sensor networks, Sensors 12 (11) (2012) 14887–14913, http://dx.doi.org/10.3390/s121114887.
[73] Karan Nair, Janhavi Kulkarni, Mansi Warde, Zalak Dave, Vedashree Rawalgaonkar, Ganesh Gore, Jonathan Joshi, Optimizing power consumption in iot based wireless sensor networks using Bluetooth Low Energy, in: 2015 International Conference on Green Computing and Internet of Things, ICGCIoT, IEEE, 2015, pp. 589–593.
[74] Roy Friedman, Alex Kogan, Yevgeny Krivolapov, On power and throughput tradeoffs of wifi and bluetooth in smartphones, IEEE Trans. Mob. Comput. 12 (7) (2012) 1363–1376.
[75] RaspBerry, RaspBerry Pi 3 B+, 2023.
[76] Christopher J. Hansen, Internetworking with bluetooth low energy, GetMobile: Mob. Comput. Commun. 19 (2) (2015) 34–38.
[77] Microsoft, Azure cloud services, 2023.
[78] Christos H. Papadimitriou, Computational complexity, in: Encyclopedia of Computer Science, 2003, pp. 260–265.
[79] Sammie Bae, Sammie Bae, Big-O notation, in: Javascript Data Structures and Algorithms: An Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals, Springer, 2019, pp. 1–11.
[80] Hamoud Younes, Mohamad Alameh, Ali Ibrahim, Mostafa Rizk, Maurizio Valle, Efficient algorithms for embedded tactile data processing, in: Electronic Skin, River Publishers, 2022, pp. 113–138.
[81] Zubeda K. Mrisho, Jema David Ndibwile, Anael Elkana Sam, Low time complexity model for email spam detection using logistic regression, Int. J. Adv. Comput. Sci. Appl. 12 (12) (2021).
[82] S. Gayathri Devi, K. Selvam, S.P. Rajagopalan, An Abstract to Calculate Big O Factors of Time and Space Complexity of Machine Code, IET, 2011.
[83] Yifeng Liu, Eyhab Al-Masri, Slow subscribers: A novel IoT-MQTT based denial of service attack, Cluster Comput. (2022) 1–12.
[84] Kitae Hwang, Jae Moon Lee, In Hwan Jung, Dong-Hee Lee, Modification of mosquitto broker for delivery of urgent MQTT message, in: 2019 IEEE Eurasia Conference on IOT, Communication and Engineering, ECICE, IEEE, 2019, pp. 166–167.
[85] Giancarlo Fortino, Claudio Savaglio, Giandomenico Spezzano, MengChu Zhou, Internet of Things as system of systems: A review of methodologies, frameworks, platforms, and tools, IEEE Trans. Syst. Man Cybern.: Systems 51 (1) (2020) 223–236.
[86] Abdul Waheed Khan, Abdul Hanan Abdullah, Mohammad Hossein Anisi, Javed Iqbal Bangash, A comprehensive study of data collection schemes using mobile sinks in wireless sensor networks, Sensors 14 (2) (2014) 2510–2548, http://dx.doi.org/10.3390/s140202510.
[87] Biswajeeban Mishra, Attila Kertesz, The use of MQTT in M2M and IoT systems: A survey, IEEE Access 8 (2020) 201071–201086, http://dx.doi.org/10.1109/ACCESS.2020.3035849.
[88] MongoDB, Scalability with MongoDB Atlas, 2022.