Ensembles of Incremental Learners to Detect Anomalies in Ad Hoc Sensor Networks

Hedde Bosman[a,b,*], Giovanni Iacca[a], Arturo Tejada[a], Heinrich Wörtche[a], Antonio Liotta[b]

[a] *INCAS[3], P.O. Box 797, 9400AT, Assen, The Netherlands*
[b] *Department of Electrical Engineering, Eindhoven University of Technology, P.O. Box 513, 5600MB, Eindhoven, The Netherlands*

Abstract

In the past decade, rapid technological advances in the fields of electronics and telecommunications have given rise to versatile, ubiquitous decentralized embedded sensor systems with ad hoc wireless networking capabilities. Typically these systems are used to gather large amounts of data, while the detection of anomalies (such as system failures, intrusion, or unanticipated behavior of the environment) in the data (or other types or processing) is performed in centralized computer systems. In spite of the great interest that it attracts, the systematic porting and analysis of centralized anomaly detection algorithms to a decentralized paradigm (compatible with the aforementioned sensor systems) has not been thoroughly addressed in the literature. We approach this task from a new angle, assessing the viability of localized (in-node) anomaly detection based on machine learning. The main challenges we address are: 1) deploying decentralized, automated, online learning, anomaly detection algorithms within the stringent constraints of typical embedded systems; and 2) evaluating the performance of such algorithms and comparing them with that of centralized ones. To this end, we first analyze (and port) single and multi-dimensional input classifiers that are trained incrementally online and whose computational requirements are compatible with the limitations of embedded platforms. Next, we combine multiple classifiers in a single online ensemble. Then, using both synthetic and real-world datasets from different application domains, we extensively evaluate the anomaly detection performance of our algorithms and ensemble, in terms of precision and recall, and compare it to that of well-known offline, centralized machine learning algorithms. Our results show that the ensemble performs better than each individual decentralized classifier and that it can match the performance of the offline alternatives, thus showing that our approach is a viable solution to detect anomalies, even in environments with little a priori knowledge.

Keywords: Anomaly Detection, Wireless Sensor Networks, Online Learning, Incremental Learning, Ensemble Methods

1. Introduction

The Internet of Things (IoT), a decade-long vision of a seamless networking where classic networked systems coexist with ubiquitous devices, is becoming reality [1, 2]. Nowadays, not only computers, tablets, and smart phones, but also vehicles, white goods, and other internet-enabled industrial and domestic apparatuses can be connected into a single, heterogeneous world-wide network. This opens up many possibilities for context-aware applications, such as smart buildings, smart cities, and autonomous distributed systems, among others. Within this context, it is of great interest in many IoT applications to embed in the network the ability of detecting, in an online, and decentralized fashion, anomalies in the sensed or received information. This is the focus of this paper.

One of the first classes of IoT devices are the Wireless Sensor Networks (WSN). These consist of a set of nodes, which are (generally) resource-limited embedded platforms endowed with sensors. Such nodes construct an ad hoc network to communicate with each other and with one or more sink nodes (i.e., nodes connected to a central facility for data storage and analysis). For over a decade, the community around WSN has focused mainly on the optimization of resource usage, e.g., network protocol design. Recently, however, the community's focus is shifting to the applications of WSN. Typical applications can be found in agriculture, where WSN are used to provide detailed insight on soil conditions [3], or in environmental monitoring, where they are used, for instance, to measure the effect of global warming on glaciers [4]. Other application domains include civil engineering (with various successful case studies in infrastructural monitoring [5], optimal tunnel lighting conditions control [6], water distribution network monitoring [7]), and health care (with many applications such as the monitoring of falls, medical intake or medical condition [8]). Lately, WSN are slowly being adopted also in industrial settings [9], although these applications are tightly controlled due to stringent reliability, safety and security requirements.

In many such applications, as well as in other IoT scenarios, the most compelling challenge is often to analyze, possibly in real-time, the big datasets that are generated. For domain specialists, such an analysis could provide new inferred knowledge about the sensed environment/processes, that could in turn be used to, for instance, improve their modeling. However, to make this analysis possible, automated analysis strategies [10] and big data analysis techniques [11] are needed, since such large datasets cannot be processed manually.

One special case of data analysis is the detection of *anoma-*

*Corresponding author
Email address: heddebosman@incas3.eu (Hedde Bosman)

lies, i.e., of specific events or patterns in the data that are unexpected [12]. Although the generic notion of an anomaly is rather intuitive, the specific cause (and nature) of such events varies widely across application domains. For example, in environmental monitoring an anomaly can be due to a sensor fault or an (unpredicted) environmental event [13], while in network intrusion detection, an anomaly can be the result of an intruder (often malicious) in the network [14]. Nevertheless, several generic anomaly detection methods [15] have been designed to find patterns in the data that and recognized when they are unexpectedly broken. Depending on the quantity (and quality) of a-priori information available about the environment or process at hand, these methods include a combination of formal techniques, rules, or data mining techniques from Computational Intelligence [16], Pattern Recognition and Machine Learning [17, 18]. One of the limitations of these techniques, though, is that they often make use of statistical data models (possibly inferred by unsupervised learning) or detection rules that are not always available under limited information conditions. Furthermore, these techniques generally require large amounts of data to be available and stored in memory, and considerable computer processing power. Therefore, anomaly detection based on these methods is typically performed in large, centralized (data mining) computing systems.

Such methods are clearly not compatible with IoT applications for several reasons. On the one hand, there is no natural centralized processing location for the continuously growing number of IoT devices connected to the Internet. Even if one of these (or one powerful computer) were selected as the centralized processing node, the need to transport data from all devices to this node would quickly overwhelm the network communication capacity and increase its response time. On the other hand, most IoT devices are expected to have limited power and computing resources (e.g., memory), so they could hardly take the place of a central computing node (specially in WSN applications). For these reasons, the need and interest for decentralized data processing (including anomaly detection) are steadily increasing (see, e.g., [19, 20, 21]). Decentralization can take place at the networking level (see, e.g., cognitive radio research [22, 23]) or at the application level (e.g., probabilistic inference and regression [24, 25, 26], decision making via fuzzy logics [27] or voting techniques [28]).

In spite of these contributions, few attempts have been made to develop methods for *online* learning of models in networked embedded systems. Online learning is needed to provide networked systems (e.g., WSN) with the ability to adapt to local working conditions without using a priori information. This is specially important for distributed anomaly detection. Among the available methods, some require a hierarchical organization of the network, such as in the multi-level clustering methods proposed in [29, 30]; while others are limited only to a specific class of anomalies [31], or make assumptions on the inter-node correlations and the underlying process under observation [32]. In any case, what is offered by the current literature is often based on a preliminary learning phase to build a model of the monitored system. Such approaches require several offline steps of adaptation to target a specific application domain. To the best of our knowledge, no general-purpose (w.r.t. applications and anomalies), online (i.e., with continuous learning), decentralized anomaly detection framework exists for WSNs.

In this paper, we fill this gap in two ways: 1) we present a general-purpose, online learning, decentralized anomaly detection framework that includes a heterogeneous set of local anomaly detection algorithms (applicable on a node either independently or in the form of an *ensemble*), and whose computational requirements are compatible with the stringent limitations of the embedded platforms typically used in WSNs. We build upon previous preliminary works around individual classifiers [33, 34, 35], elaborate on the challenges and choices of learning methodologies for limited-resource devices, and subject the methods to a significantly larger experimental campaign. 2) We evaluate the performance of such algorithms in contrast to centralized anomaly detection methods. For evaluation purposes, labeled datasets are required, which we obtained through synthetic generation and from real-world applications, developed in-house or available through public sources. Moreover, we review the evaluation methods, which are based on the confusion matrix metrics, describing how we account for false positives caused by anomalies in correlated sensors, and for false positives caused by a delayed detection. Through this broad experimental evaluation, we show that using machine learning in an online decentralized approach is feasible in stringent constraints of embedded systems, that it can provide an automated adaptability to different application domains without manual intervention, and that it can match (using an ensemble) the performance of centralized alternatives.

The remainder of the paper is organized as follows: the next section summarizes the related literature in the areas of data mining and anomaly detection. Section 3 describes our proposed anomaly detection methods, explains how these are combined into an ensemble, and details how the learning algorithms are ported onto resource-constrained sensor nodes. Section 4 illustrates the details (datasets and evaluation metrics) of the experimental setup used to assess the anomaly detection performance. Numerical results are shown in Section 5, together with a thorough discussion of both the individual algorithms and the ensemble performance. Finally, Section 6 presents our conclusions.

2. Related work

Anomaly detection is not limited to WSN. It has also been used in several other domains, as summarized next.

2.1. Overview of anomaly detection

Anomaly detection is increasingly and commonly used to detect anomalous access [36] or fraud [37]; to detect problems in data center performance and, through diagnosis, to provide potential remedies [38]; to detect network intrusions [39]; and to detect targets in military applications [40]. Also, anomaly detection is often used in data-mining systems, e.g. to extract data of interest from large databases of historical data [37, 41].

As noted in the previous section, in all these applications the data processing typically takes place in a centralized system

with abundant processing and storage capacity. In spite of this, the ever increasing amount of data that needs to be processed poses a mayor challenge. Several possible solutions have been advanced by the data mining research community, including parallelization, distribution, and stream processing. Parallel data mining partitions and distributes the global data to different computing nodes, such that, when the node results are aggregated, a global result of the mining process is formed. A well-known example is the Map-Reduce algorithm [42], that maps (partitions) the problem onto computing nodes, and reduces (aggregates) the individual results to the global result. In contrast, distributed processing analyzes data from a local neighborhood using, e.g., data mining techniques. The results can then be aggregated into more global knowledge through hierarchical or central aggregation methods. However, this method does not always provide the same result as global analysis [43]. Finally, stream processing consists of updating a model of the data that arrive at the processing system, while the data are acquired. The model can be updated regularly, by training on batches of new data, by incrementally improving the model to account for the new data, or by other online methods. A good example of this kind of methods is LASVM [44], where the model is represented by support vectors, which are pruned after updates. It is important to note that these methods can decrease computational complexity and memory usage, albeit at slightly reduced classification accuracy.

2.2. Anomaly detection in WSNs

Surprisingly, most anomaly detection methods available in the literature are based on an offline, centralized data processing paradigm, as evidenced in the recent surveys [10, 45, 46]. Under this paradigm, all sensor data are collected at a central storage facility, where are processed via standard data mining techniques, such as an ensemble of classifiers [47]. Only recently, there have been some seminal investigations into distributed methods. We can distinguish methods that apply an offline-trained model from those that learn the model online. Using an offline training phase before deployment allows the usage of relatively complex data-driven models. A good example is given by Chang et al. [48], who train Echo State Networks to detect anomalies in time-series. Another example is an offline-trained ARMA model which is used to detect anomalies, such that the system compares new measurements with the predictions of the model and classifies them as normal based on a neighborhood voting system [49].

There have been some recent investigations into distributed methods with online learning. Many of these are based on Least Squares optimization, described by Gauss and Legendre in the early 19th century [50]. In earlier work, we used the Recursive Least Squares (RLS) method to create a model of linear correlations between sensors for anomaly detection [33]. With little computational complexity, such models can be learned, and kept up to date, also in limited-resource systems. Another option is Piecewise Linear Regression, where the Least Squares technique is used to model a time-series into linear segments: by comparing the current segment to historical segments of, e.g., 24 hours before, anomalies can be detected [13]. However, this method does not take into account correlations between sensors or neighboring nodes.

Moreover, the Least Squares method allows for learning of more complex models. A good example is Extreme Learning Machine (ELM), a method based on a Single Layer Feed-Forward Neural Network (SLFN). Compared to RLS, ELM also allows modeling non-linear relations between, for instance, local sensors. In previous work, we have shown how Online Sequential Extreme Learning Machine (OS-ELM) can successfully be used for anomaly detection on WSN nodes [34].

It should be noted that there are also other machine learning approaches that can be learned online, such as Reinforcement Learning (RL) and clustering [51]. However, an application of these methods to online anomaly detection is not immediate. More specifically, RL, that is successfully used in contexts such as optimal on-off node scheduling [52], or optimal routing [53], requires that the nodes have some active behavior and that they get a reward from the interaction with the environment. However, in monitoring applications, where nodes are typically passive sensing entities, this is not always possible. As for clustering, well known examples include K-Nearest Neighbors (KNN) and K-Means. While for K-NN one needs to store all the data in memory (going beyond the hardware limitations of WSNs), for K-Means one only needs to store the cluster centroids (means). Combined with the possibility to incrementally update the model, K-Means is in fact a suitable candidate for online data mining in WSNs [54], with successful applications for instance in network topology adaptation [55]. On the other hand, in order to apply K-Means to anomaly detection, one needs to distinguish between different clusters (normal vs anomalous) and provide suitable similarity measures, which may require an offline preprocessing phase.

3. Implementation of learning algorithms on resource-constrained embedded nodes

As mentioned before, there are various methods to detect anomalies in WSNs. Here, we will focus on the typical use case for WSNs: the monitoring of environments or processes about which there is (presumably) little a priori information and for which it is impossible to ascertain beforehand what "normal" or "anomalous" is. Further, we concentrate on those detection methods that can be applied online (i.e., without the need of an offline learning phase) and that are characterized by a limited computational footprint, so as to accommodate the stringent hardware limitations found in ad hoc WSN nodes. Thus, we will consider only data-driven anomaly detection methods (based on learning, or self-adaptation), which are easier to apply in these situations than their model-driven counterparts. More specifically, we consider incremental learning-based modeling, as these method requires fewer resources such as memory.

The trade-off, however, is that the methods presented here are better suited for situation where the environment's "normal" state occurs (and is measured) much more often than its anomalous states. In the sequel, we will assume that such conditions

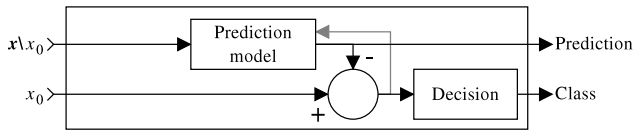

Figure 1: Structure of a multi-dimensional classifier. The difference between a prediction, based on inputs $\mathbf{x} \setminus x_0$, and the measurement in x_0 is classified.

hold so that, provided that sufficient data are acquired, a robust learning mechanism can, on average, learn (i.e., model) quite reliably at least an approximation of the environment's "normal" state. Note that, although this assumption is expected to be met in practice, it makes the evaluation of the anomaly detection performance particularly difficult, since doing so requires either a "skewed" test dataset (i.e., datasets with few anomalies in them) or a very large test dataset that provides enough anomalies to estimate the probabilities of detection (see Section 4.4 for more details).

In the following, we first describe our overall approach (Section 3.1). We detail four specific prediction methods (Sections 3.2-3.5), and then continue to explain how these predictions can be analyzed such that a classification can be made (Section 3.6). Then we explain how individual classifiers can be combined into an ensemble (Section 3.7) and, finally, we define a number of offline baseline methods in order to compare the results of our online methods (Section 3.8).

3.1. Overall approach

The general anomaly detection procedure goes as follows: after data are acquired and an approximate model of the environment has been learned (and adjusted online), the model is used to predict the future state of the environment. A large "deviation" between this prediction and the next measured state signals a potential anomaly (or unmodeled dynamics of the system). The deviation can be stated in several ways. For instance, if the prediction error were to be modeled by a Gaussian random variable ϵ with (unknown) mean μ and standard deviation σ, then one could use statistical tests to ascertain whether or not the measured state of the environment falls outside the 95% confidence interval for its expected value, indicating the presence of an anomaly. Although this general approach to anomaly detection is simple and follows closely the definition of "anomaly" given in the previous section, it has a drawback: its detection accuracy is limited by the quality of the learned model (a "perfect" model of the environment is not available). That is, the detection accuracy depends on the ability of the learning mechanism to build a "good" model.

The approach described above constitutes the building block of the proposed anomaly detection framework and can be depicted as in Figure 1. In this structure, we have a number of inputs $\mathbf{x} \triangleq (x_0, x_1, \cdots, x_d)$, usually a time series of sensor measurements, where x_0 is the *target* value, that is, the value that we want to classify as normal or anomalous. The remaining inputs $x_1, x_2, \cdots, x_d$ (that we indicate with $\mathbf{x} \setminus x_0$) form the basis on which the *prediction model* creates an estimate of the target $\hat{x}_0$ and can be obtained from, for example, other sensors or historic measurements. The prediction error $\epsilon = x_0 - \hat{x}_0$, is then fed to a *decision making component* that can classify x_0 as normal or anomalous based on learned properties of the prediction error.

In our framework, the key feature to overcome the aforementioned drawback is the combination of several prediction models (described below) in two possible configurations. The first is a *fusion* based combination, where not only sensor measurements, but also the predictions from several other models, are the inputs to another prediction model. The second combination configuration is that of an *ensemble*, where the output of several classifiers (each employing different prediction models) is combined to increase accuracy.

As said, in order to construct prediction models of the environment with little or no a priori information, we resort to data-based learning methods. Of these though, only methods whose memory and computational footprint fit within limited hardware resources are suitable for implementation in embedded systems. For example, memory resources available to wireless sensor nodes are in the order of kilobytes, and clock speed is in the order of MHz (with 8 or 16 bit MCUs). Moreover, a WSN node's processing power is limited not only by the speed of its MCU, but also by its capabilities. A common WSN platform such as the TelosB[1] does not have floating-point units. The learning methods that can be adapted to fit these requirements are mostly incremental learning methods, where at each iteration the model is updated to converge to the least squares optimum. While incremental learning methods may incur more computational costs compared to learning from the whole time-series at once (the latter also requiring its storage), the computational and memory cost per new measurement are, in general, low. That, combined with the real-time processing of data, make incremental learning methods ideal for this application.

When a suitable incremental learning method has been found, possible issues related to these limited resources have to be identified. Common limitations are that of memory consumption and integer or fixed-point computations. The former limits, for instance, the number of input dimensions d that can be used, or the number of historical measurement values that can be stored. The fixed-point computations suffer from rounding errors, underflows and overflows, which can make the learning method unstable. Therefore, these problems have to be identified and countered by, for instance, a reset when an overflow is detected, or a variable correction to prevent divisions by zero. For the methods presented below, we have used two open-source fixed-point calculation libraries, `libfixmath` [56] and `libfixmatrix` [57], which allow for 16-bit precision both for the integer and fractional part (denoted as Q16.16). In the next sections, we will present the implementation details of several incremental learning algorithms, addressing in particular the effects of the limited precision on the learning process and how those effects can be overcome.

[1] TelosB is an 8-bit MSP430-based platform with 10KB ROM, 48KB RAM and peripherals such as wireless 2.4 GHz radio (TI CC2420), temperature, humidity (SHT11) and light sensors (Hamamatsu S1087 series).

3.2. Sliding window mean

The simplest prediction method is based on plain statistics. In order to generate these statistics, we use a sliding window approach that keeps in memory a short window of L_h data samples and use standard methods to determine the mean μ and standard deviation σ of such data. The estimate of μ can then be used as the prediction for x_0. Note, however, that due to the risk of overflow in the limited-precision environment, all summands in the sums required to estimate μ and σ are divided by L_h *while* the sums are performed instead of dividing the results *after* the sums are completed. That is, $\hat{x_0} = \sum_{i=0}^{L_h} x_{0,-i}/L_h$, where $x_{0,-i}$ is the i-th historic measurement of x_0.

Additionally, we propose a rule-based decision over the standard deviation σ. That is, we expect the system under measurement to display slow changes, and to display noise in the sensor measurements. This means that a series of consecutive measurements should not have $\sigma \leq \delta$ (where $\delta \to 0$, e.g. $\delta = 1/2^{16}$, the smallest number representable in Q16.16); otherwise, this rule detects a specific type of anomaly, namely a *constant* anomaly (see Sec. 4.1 for further details).

3.3. Recursive Least Squares

Recursive Least Squares (RLS) is a variant of the linear Least Squares estimation method, which recursively attempts to minimize the least square (prediction) error of a linear model $y = \boldsymbol{\beta}\mathbf{x}$, as depicted in Figure 2. The inputs $\mathbf{x}$ can, for example, represent measurements across different sensors taken at one or more instances in time. These are then linearly mapped to the output y, by the weighting of $\boldsymbol{\beta}$. In our approach, we assume that the different sensors (e.g., for temperature or humidity as in a TelosB node) exhibit a degree of correlation. Anomalies can then manifest themselves as breaking this correlation, causing a strong difference between predicted and measured sensor values.

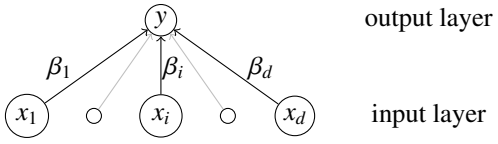

Figure 2: Structure of Recursive Least Squares model.

In earlier work, we showed that RLS can be used as a very lightweight form of learning [33]. As illustrated in Algorithm 1 (for more details consult the work by Bottomley [58]), the method starts by initializing some parameters (the forgetting factor α and the value δ needed to initialize the inverse auto-correlation matrix P) and state variables (the weights $\boldsymbol{\beta}$ and P). Then, for each new iteration, the size of the prediction error, ϵ, is determined first. Following, the variables θ and μ express the updated gain. Next, the direction in which to update β and P is determined using variables K and κ. With these, finally, the weights $\boldsymbol{\beta}$ and inverse auto-correlation matrix P are updated.

The algorithm complexity per iteration is in the order of $O(d^2)$, where d is the dimension of the input vector $\mathbf{x}$. This constitutes a limitation of RLS in resource-constraint systems. Moreover, as we showed in [33], in fixed-point environments

Algorithm 1 Recursive Least Squares

1: Init $\delta > 1, \alpha > 1, \beta_{0,i} = \{0\}, P_0 = \delta I$
2: **for** each sample step t **do**
3: $\quad \epsilon_t \leftarrow y_t - \mathbf{x}_t^\top \beta_{t-1}$
4: $\quad K_t \leftarrow P_{t-1}\mathbf{x}_t$
5: $\quad \mu_t \leftarrow \mathbf{x}_t^\top K_t$
6: $\quad \theta_t \leftarrow \frac{1}{\alpha+\mu_t}$
7: $\quad \kappa_t \leftarrow \theta_t K_t$
8: $\quad \beta_t \leftarrow \beta_{t-1} + \kappa_t \epsilon_t$
9: $\quad P_t \leftarrow \frac{1}{\alpha}\left[P_{t-1} - \theta_t K_t K_t^\top\right]$

this algorithm can be unstable due to aforementioned issues with limited precision. The main problem is the underflow in the computations regarding P_t, which can result in coefficients for a certain x_i to become zero. This has as effect that x_i is not used for correlations in future iterations. Moreover, overflows can occur when the input values are too big, and therefore the inputs have to be scaled. However, based on the work of Bottomley [58], we found that the RLS algorithm can be stabilized by taking a few simple measures:

- Truncating (instead of rounding) the computational results, to bias the error propagation towards stability.
- Use saturating math operators, such that overflows do not wrap around (e.g. positive to negative).
- Biasing the diagonal values of the auto-correlation matrix P to counter underflows which, otherwise, might prevent a zero weight to be updated. That is, $P_n = P_n + (1/2^{16})I$ (where I is identity matrix) after step 9 in Algorithm 1.
- Ensuring that the input is at most of 16 bit precision which, together with the forgetting factor α and scaling after using the Q16.16 representation of `libfixmath`, prevents overflows.

3.4. Extreme Learning Machine

Artificial Neural Networks (ANN), especially Feed-Forward ANN, have been one of the most influential Artificial Intelligence methods in the past decades, due to the universal function approximation property [59]. Traditionally, ANN are trained using the back-propagation method, a gradient descent method that learns the weights and biases of the neurons, typically requiring multiple iterations over a training dataset [60]. More complex learning techniques are also capable of adjusting the structure of the neural network, such as the number of neuron layers and the number of hidden neurons in each layer. Unfortunately, these learning procedures are extremely expensive in computational terms.

In recent years, however, an efficient learning approach for a specific type of ANN, the Single Layer Feed-Forward Neural Network (SLFN), has been receiving more attention. A SLFN has the following structure (see Figure 3):

$$y = f_{\tilde{N}}(\mathbf{x}) = \sum_{i=1}^{\tilde{N}} \beta_i \mathrm{G}(\mathbf{a}_i, b_i, \mathbf{x}), \quad \mathbf{x}, \mathbf{a}_i \in \mathbb{R}^d, \quad b_i \in \mathbb{R}, \qquad (1)$$

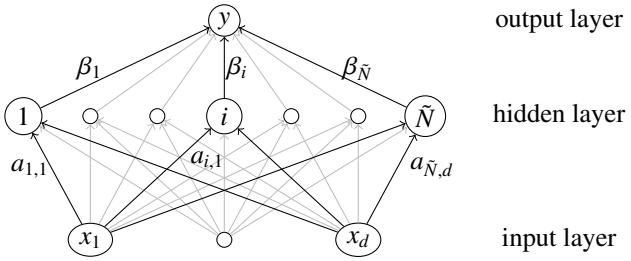

Figure 3: Structure of a Single Layer Feed-Forward Neural Network model.

where $\mathbf{a}_i$ are the input neuron weights, b_i the bias of the ith hidden neuron, $\mathrm{G}(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of that hidden neuron according to activation function $G(\cdot)$, and $\tilde{N}$ is the number of hidden neurons. Also in this case, $\mathbf{x} \in \mathbb{R}^d$ is a vector of input values of dimension d, while β_i is the weight connecting the ith hidden neuron to the output node. The learning approach for an SLFN is then based on a random choice of input weights and biases, leaving only the output weights $\boldsymbol{\beta}$ to be learned. Given a large enough number of hidden neurons $\tilde{N}$, this approach performs on par with other machine learning methods, such as Support Vector Machines. While this approach can also be found in, for example, random vector functional-link neural networks [61], we have chosen to follow the implementation[2] of the Extreme Learning Machine algorithm provided by Huang et al. [62].

The main intuition behind ELM is that, if one sets input weights and biases randomly and N training samples are considered, the hidden neuron outputs can be aggregated into a matrix $\mathbf{H}$ of $N \times \tilde{N}$, where $H_{j,i} = \mathrm{G}(\mathbf{a}_i, b_i, \mathbf{x}_j)$. It is then possible to rewrite eq. (1) as:

$$\mathbf{y} = \boldsymbol{\beta}\mathbf{H}, \tag{2}$$

thus allowing the output weights $\boldsymbol{\beta}$ to be determined analytically using an ordinary Least Squares estimation approach. Over the years, such an approach has shown to be a powerful non-linear mapping method used in classification and regression [63].

Using the Moore-Penrose generalized inverse of the hidden-layer output matrix H, one can find the hidden neuron weights $\boldsymbol{\beta}$, that is, $\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{y}$ is the least squares solution to Equation 2. However, the least squares solution can also be obtained sequentially using RLS. In [34], we have adapted the approach taken by the Online Sequential ELM (OS-ELM) method, proposed by Liang et al. [64]. We showed that this learning method can also be implemented on WSN devices and applied to detect anomalies, by identifying the following issues due to resource constraints. Firstly, as this method uses RLS, the same measures counteract the fixed-point precision issues indicated in Section 3.3 apply to OS-ELM. Next, the activation function $G(\cdot)$ may pose limits on the input values due to the fixed-point precision. In the simplest case, for example, a linear combination of all inputs may overflow the summation variable. To account for this, the inputs have to be scaled accordingly. Lastly, the size of the hidden matrix $\mathbf{H}$ and the variables needed to update this matrix are limited by the available memory, resulting in a limited hidden layer size $\tilde{N}$ and a limited number of inputs d. The algorithm's pseudo-code, consisting of an initialization and a sequential learning phase, is shown in Algorithm 2. For an extensive ELM description, see the work of Liang et al. [64].

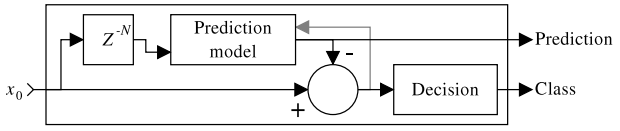

Figure 4: Structure of a single-dimensional time-series classifier. The difference between a prediction, based on a model from the signal delayed by N samples, and the current measurement of the signal x_0 is classified.

Algorithm 2 Extreme Learning Machine

function InitializeELM($\mathbf{x}_{0,\cdots,\tilde{N}}, y_{0,\cdots,\tilde{N}}$)
  for node $i \in 0, \cdots, \tilde{N}$ **do**
    inputs weights $\mathbf{a}_i \leftarrow$ random number between (-1,1)
    inputs biases $b_i \leftarrow$ random number between (-1,1)
  $\boldsymbol{\beta} \leftarrow$ estimate coefficients from initial data $\mathbf{x}_{0,\cdots,\tilde{N}}, y_{0,\cdots,\tilde{N}}$
    using closed form least squares solution
function LearnSequentialELM($\mathbf{x}_t, y_t$)
  Calculate partial hidden layer output matrix $\mathbf{H}_t$
    with hidden neuron outputs $\mathrm{G}(\mathbf{a}_i, b_i, \mathbf{x}_t)$,
  Determine the desired output y_t,
  Compute output weights $\boldsymbol{\beta}_t$ using RLS, given $\mathbf{H}_t$ and y_t
  Apply stability correction for RLS

3.5. Function approximation

The final prediction method considered in our framework is polynomial function approximation (FA). A polynomial function can be approximately fitted to a sliding window of data and can be used in two cases. In the first case, one can use the fitted polynomial to predict future measurement values. In the second case, one can compare the coefficients of the polynomial fitted to one segment to those from polynomials fitted to other segments, yielding a piece-wise linear anomaly detection method [13]. Here, we have opted for the first case by implementing a method called SwiftSeg[3] on fixed-point embedded systems, which uses bases of orthogonal polynomials to incrementally fit a polynomial function to a data window [65]. That is, given orthogonal basis $p_k(x)$, the fitted function $p(x) = \sum_{k=0}^{K} a_k / \|p_k\|^2 p_k(x)$. The pseudo-code with fixed-point corrections is shown in Algorithm 3. Its computational complexity is in the order of $O(deg^2 + L_s)$, while the memory footprint is in the order $O((deg + 2)^2 + L_s)$ (where deg is the degree of the polynomial used, and L_s the window size). Hence, this method is suitable for online embedded function approximation.

The update step recursively calculates the coefficients of each basis polynomial; the estimation step evaluates all the basis polynomials with the current coefficients. In the latter, however, the accumulating variables that are used in this evaluation

[2] http://www.ntu.edu.sg/home/egbhuang/elm_codes.html

[3] http://www.ies-research.de/Software

Algorithm 3 Function Approximation

function InitializeFA(K, N)
  init update coefficients for F_k
    based on max K and window size N
  $\alpha \leftarrow 0$
function UpdateFACoefficients(y_{t+1})
  for degree $k \in 0, \ldots, K$ **do**
    $\alpha_{k,t+1} \leftarrow \alpha_{k,t} + F_k(y_{t+1}, y_t, \alpha_{k-1,t+1}, \ldots, \alpha_{0,t+1})$
    if $\alpha_{k,t+1}$ is overflown **then**
      reset variables and restart with buffered window
function Estimate(t)
  $q \leftarrow 0$
  for degree $k \in K, \ldots, 0$ **do**
    $q \leftarrow combine(q, p_k(t))$
  if q is overflown **then**
    reset variables and restart with buffered window
  return q

can run into the limits of the fixed-precision representation and saturate (or overflow). In such case, the model is re-initialized with the previously buffered values. Note that higher degree polynomials require a higher number of accumulating variables and run a higher risk of fix-precision overflow. Hence, here we will limit our analysis to first degree polynomials.

In our earlier work, we showed that using such a function approximation method, in combination with a decision component to detect anomalies from prediction errors (depicted in Figure 4), performs on par with the aforementioned RLS approach (that has only sensor values as inputs), if one-step-ahead prediction is used [35]. Moreover, we showed that since this method only models a single time-series, it can complement methods such as RLS and OS-ELM, which instead accept multiple inputs that stem, for instance, from multiple sensors instead of historical values.

3.6. Prediction error analysis

As mentioned before, a critical step in anomaly detection is the analysis of the properties of the prediction error ϵ. If a perfect model of the environment were available, this prediction error would be expected to be zero under normal environmental conditions. In practice, however, perfect models are not available, particularly in the case of a limited-resource platforms such as found in WSNs: often instead, the only characteristics that can be estimated are the prediction error's mean μ and standard deviation σ, using an Exponentially Weighted Moving Average (EWMA) [66]. Ideally though, good practical models relying on simple statistics should still be able to yield high-accuracy predictions, thus leading to "almost zero" or "low" prediction errors under normal conditions.

Once the statistics of the prediction error are established (either via analysis or by assuming that it is normally distributed), one can calculate the prediction error's p-value, that is, the probability of obtaining a sample at least as far from the mean as the current sample. Note that the lower the p-value, the more likely the prediction error signals an anomaly. Thus, after acquiring sufficient data to accurately estimate the prediction error's statistical parameters, the decision component (see Figure 1) can determine the prediction error's p-value, compare it to a user-defined confidence level (e.g. 95%), and determine whether or not it can be regarded as anomalous. In this way, the end-user sets a fixed confidence threshold, while the p-value is estimated adaptively, depending on the underlying distribution of the prediction error.

In general, for algorithms such as EWMA, it is difficult to set a single learning rate that is suitable for all applications. Thus, in our framework we re-sample and smooth the standard deviation to account for anomalies of longer duration. That is, after each $N_{resample}$ (e.g., 128) new prediction errors, we take the current σ and smooth it using a second EWMA, in order to create a slower moving σ_l, that is more robust to changes. With this slower changing σ_l we determine a second p-value, and we choose the minimum of the p-values resulting from the normal and slow adapting the probability density function. For completeness, we report the pseudo-code of the adaptive threshold detection in Algorithm 4. Further details can be found in [33].

This simple, yet effective, adaptive decision-making scheme can be easily applied to the prediction methods described before. Each of those four methods (sliding window mean, RLS, OS-ELM, and function approximation), together with EWMA-based decision making, form a one-class classifier, see respectively the "Prediction model" and "Decision" blocks in Figures 1 and 4. The added benefit of classifying over prediction errors, is that an anomalous value can be replaced by this prediction.

Algorithm 4 Adaptive Threshold Detection

initialize μ_t, $\sigma_{t,s}$, $\sigma_{t,l}$, F_s, F_l, λ_s and λ_l.
function AnalyzePredictionError(ϵ_t)
  $\mu_t \leftarrow$ EWMA_update(μ_{t-1}, ϵ_t)
  if initializing or no anomaly **then**
    $\sigma_{t,s} \leftarrow$ EWMA_update($\sigma_{t-1,s}, (\epsilon_t - \mu_{t-1,s})$)
  if t mod $N_{resample} == 0$ **then**
    if initializing or no anomaly **then**
      $\sigma_{t,l} \leftarrow$ EWMA_update($\sigma_{t-1,l}, \sigma_{t,s}$)
  if initializing **then**
    determine multipliers F_s such that
      all past ϵ are within 95% of $\mathcal{N}(\mu_t, F_s\sigma_{t,s})$
    determine multipliers F_l such that
      all past ϵ are within 95% of $\mathcal{N}(\mu_t, F_l\sigma_{t,l})$
    return 1.0
  else
    $p_s \leftarrow$ p-value of ϵ_t given $\mathcal{N}(\mu_{t-1}, F_s\sigma_{t-1,s})$
    $p_l \leftarrow$ p-value of ϵ_t given $\mathcal{N}(\mu_{t-1}, F_l\sigma_{t-1,l})$
    return $\min(p_s, p_l)$

3.7. Combining multiple predictors or classifiers

Although the above described prediction-based classifiers can be applied individually, it is well established that an ensemble of learners can improve the reliability and performance of classification [67]. Moreover, multiple sources of information can also be fused in a single model to increase its accuracy

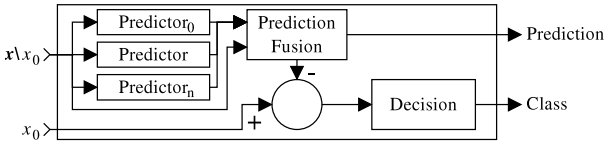

Figure 5: Structure of prediction fusion classifier. The predictions of several individual predictors are fused with sensor measurements in another prediction model. A measurement is then classified w.r.t. this prediction using the same decision making component as in Figure 1.

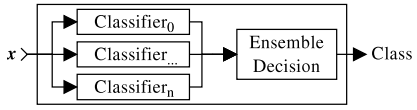

Figure 6: Structure of an ensemble of classifiers. The final decision on *class* is based upon the outputs of the different classifiers.

[68]. In our framework, we consider two alternative combination schemes, based on fusion of predictors and ensemble of classifiers, graphically visualized in Figure 5 and 6, respectively.

The first scheme is composed of individual predictors (see Sections 3.2-3.5), each connected to the same measurement inputs. These measurements and the predictions generated by the individual predictors are then taken as the inputs of the second stage composed of a multi-dimensional input predictor, such as RLS or OS-ELM, which generates a new prediction, $\hat{x}_0$, given a target sensor value x_0. The second stage prediction error can then be analyzed as described in Section 3.6.

Alternatively, the ensemble scheme takes into account the classification based on multiple individual prediction errors. In the simplest case, heuristics can be used. For instance, given the a-priori knowledge that the rule-based *constant* classifier shows quite good performance, one can heuristically determine a simple ensemble decision, such that the output of the *constant* classifier is preferred over that from another classifier, say, the RLS-fusion classifier. This heuristic ensemble can be expressed with a simple rule, as illustrated in Algorithm 5.

Algorithm 5 Heuristic ensemble

```
if constant anomaly then
    p-value ← Constant classifier p-value
else
    p-value ← RLS-fusion classfier p-value
```

A more generic approach is to assign to each individual predictor's classification result an "outlier" score that reflects the confidence level on the classification result, and then combine the classification results in a meaningful way. This often requires the normalization of the outlier scores and the selection of an appropriate combination function [69]. Since in our approach each classifier outputs a p-value, which indicates classifier's confidence that a sample belongs to the normal environmental state, there is no need for normalization. Thus, we can combine these p-values using several well-known ensemble combination methods: *majority voting*, *mean*, *median*, *minimum* or *maximum* p-value selection. We expect that the median and majority voting ensemble produce very similar results according to the median voter theorem [70], which states that a majority vote will produce the preferred outcome of the median voter. Furthermore, because we are using the p-values as outlier scores, we can directly use Fisher's method [71] as a combination method by computing the following sum:

$$-2\sum_{i=1}^{k}\ln(p_i) \sim X^2_{2k},$$

where k is the number of classifiers, p_i denotes p-value produced by classifier i, and $\sim X^2_{2k}$ denotes that the sum has a Chi square probability distribution with $2k$ degrees of freedom. Because we are using different learning techniques, underlying the generation of the p-values from the classifiers, we have implicitly assumed that the tests are independent. However, attention should be paid when dependent tests are combined. In turn, Fisher's method yields another p-value, which can be eventually used to classify the input as anomalous or normal.

In this work, we evaluate the performance of these ensembles and that of each individual classifier, implemented in a resource-limited wireless sensor network platform. The data flow and methods under consideration are shown in Figure 7. The constant rule, sliding window mean and function approximation classifiers act on a small window of historic data $x_{win,0}$, while the RLS and OS-ELM classifiers use $\mathbf{x} \setminus x_0$ to predict and classify x_0. The RLS and OS-ELM fusion classifiers take as inputs not only the raw sensor data, but also the predictions generated by sliding window mean and function approximation. The prediction errors are analyzed as described in section 3.6 yielding p-values that can in turn be used to classify x_0 or be used as input for the ensemble combination methods.

Finally, the ensemble methods take p-values from the classifiers and combine them into a final decision. In previous work, we used the heuristic rule and Fisher's method on a partial set of classifiers. In this work, we extend this initial work by including all the individual classifiers in the inputs for the ensembles and by using the other combination methods mentioned earlier. All these combinations are shown in Figure 7, where the red arrows show the p-value/classification outputs of each individual classifier and ensemble of classifiers.

3.8. Baselines

To compare the above online methods, we contrast them to their offline counterparts. In particular, we have a rule-based baseline, which combines the same constant classifier, as outlined above, with the computation of a p-value over the difference of the sensor signal ($\delta x = x_{0,t} - x_{0,t-1}$). With those, the true mean μ and variance σ of δx can be computed, because we have all data in memory, and with that the p-value.

Furthermore, we use the offline counterparts of RLS and OS-ELM, being Linear Least Squares Estimation (LLSE) and ELM in the same structure as displayed in Figure 1, to learn linear and SLFN models of the relation between the target sensor and the remaining sensors in the system. The prediction errors are then

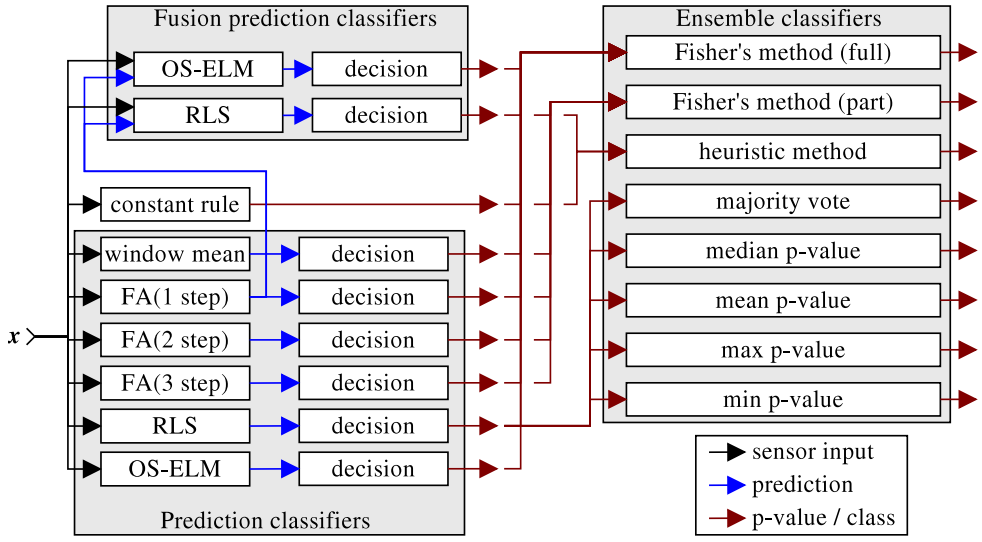

Figure 7: The combination of online embedded methods used. Each p-value/class arrow indicates a classification. The ensemble "Fisher's method (part)" refers to the combination used in [35].

analyzed, similarly to the methods above, by computing the p-value. However, since the prediction errors are known over the whole time series, the true mean μ and variance σ of the prediction error can be computed, instead of using EWMA. Moreover, because all the data are available offline, a possible benefit can be obtained by including delayed versions of the measurement time-series in the model. Since the real-world data includes day/night patterns, a logical choice for such a delay is that of one day, or one period, earlier.

Both the normal and the delay-included LLSE and ELM models, together with the offline prediction error analysis, are used as baselines. Furthermore, the results of the rule-based, the LLSE and the ELM baselines are combined into a minimum p-value ensemble to give a baseline for the ensemble methods.

4. Experimental setup

In this section, we develop the setup used to evaluate the methods introduced in Section 3. First, we define the categories of anomalies we focus on, followed by a description of how such anomalies are labeled. Then, we introduce the six datasets (both from synthetic and real-world application domains) we use for evaluating our methods. Finally, we describe the evaluation metrics used to compare our online methods to the offline baselines introduced in the previous section.

4.1. Categories of anomalies

As noted in Section 1, the kind of anomalies to be detected (and their causes) can be very different across various applications. Nevertheless, generic anomalous patterns can be observed regardless of the underlying application. We consider here a representative set of four well-known application-independent anomalies [13, 72]: *spike*, *constant*, *noise* and *drift* (see Figure 8). These anomalies are prevalent in several real-world domains [73, 74], since they can be caused by numerous environmental factors and system faults.

Spikes are short-duration (often single samples) positive or negative peaks in data. They may be caused, for example, by malfunctioning hardware connections or by bit flips. A *noise* anomaly manifests itself as a sudden, unexpected, increase in measurement variance. This is often caused by a nearly depleted battery, but can also be caused, for instance, by the sudden appearance of an extra factor in the environment. When the measured data reflect no changes present in the original signal or in the noise associated with the measurement process, we classify this as a *constant* anomaly. Such anomalies may occur, for example, when a sensor is blocked, either physically or by software, or when the connection to the sensor is broken. Lastly, *drift* expresses an offset between the measured values and the original signal. This might be caused by a degrading sensor or by unmodeled environment dynamics. From the aforementioned anomalies, *drift* is the hardest one to detect through online learning, since the latter may not be able to capture the drift over time and the difference from the real signal.

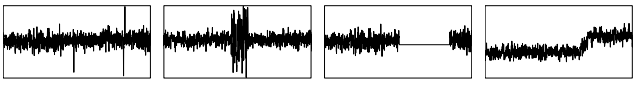

Figure 8: Injected anomalies in noisy linear data: (a) spike, (b) noise, (c) constant and (d) drift.

4.2. Labeling of data

Labeled datasets are of key importance to evaluate the results of our unsupervised anomaly detection methods. In contrast with the synthetic datasets, in real-world datasets the labeling is not always exact and largely depends on expert knowledge, which is often unavailable. Subjective or inconsistent labeling can be improved by using multiple experts, in a procedure similar to ensemble techniques. We can resort to semi-automated techniques when experts are not available for a given dataset.

To generate labels, then, we can run several automated anomaly detection techniques that are checked and corrected by hand given the limited expertise of the person correcting the labels.

By manual analysis of the real-world data, we can observe certain behavior that could be labeled by rule-based methods. Such rules can, for instance, label a value that is not changing for over 10 samples as a constant anomaly. Other behaviors we observed to be anomalous were values that signify a fault in the measurement, significant shifts in the variation of the signal, large unexpected increases in value, or abnormally large readings. For our experiments, we capture these behaviors in rules that are applied automatically to the real-world data. In a next step, we then manually check, correct or add upon these labels. This semi-automated labeling gives us a relatively efficient method to label available data, but still depends on our human judgment.

4.3. Datasets

To assess our methods' ability to detect the four aforementioned anomalies, we apply them to six datasets composed of both synthetic and real-world data. The synthetic datasets contain anomalies injected on purpose at specific moments in time. Thus, each sample is known to be either normal or anomalous (i.e., it is labeled) beforehand, which allows us to compute precise anomaly detection statistics. In the real-world datasets there is no a priori knowledge about the samples, so the detection performance can be estimated only after a preliminary offline manual analysis and labeling of the data, as described in the previous section.

We consider first the synthetic datasets. The first synthetic dataset, $Synthetic_L$, consists of 3 signals per node that follow a noisy line. That is, each signal can be described as $s = at + b + \mathcal{N}(0, 0.1)$, where t is time, a is the slope, between -1 and 1, b is a constant intercept between 0 and 1, and $\mathcal{N}(0, 0.1)$ ia a zero-mean Gaussian noise with variance 0.1.

The second synthetic dataset, $Synthetic_{LS}$, is similar to the first, but also includes a periodic signal that simulates periodic environmental changes, such as those under day and night conditions, or seasonal changes. This signal can be expressed as $s = at + b + \sin(2\pi t/d) + \mathcal{N}(0, 0.1)$, where a, b, t and $\mathcal{N}(0, 0.1)$ are as before, and where $\sin(2\pi t/d)$ is the cyclic signal with a period of d samples.

The last synthetic dataset, $Synthetic_{RW}$, is inspired by real-world temperature data that contains daily patterns and slowly varying trends. A shared daily trend $R_d(t)$ for each signal is generated from a random walk. This daily value is then interpolated using a cubic spline to generate d values per day, resulting in a smooth temperature trend. Then, for each signal, we add another random walk $R_s(t)$ to generate individual signals. Finally, we add a cyclic component $\sin(2\pi t/d)$, that is amplitude modulated by yet another random walk $R_c(t)$ in order to reflect the changes in temperature across and during the days. Furthermore, all random walks over the given length are normalized to be in the range of -0.5 to 0.5. This results in signals expressed as $s = 4R_d(t) + R_s(t) + R_c(t)\sin(2\pi t/d) + \mathcal{N}(0, 0.1)$, where all stochastic processes are re-evaluated per signal, apart from the daily trend R_d that simulates a common environment.

Each synthetic dataset is composed of 50 groups of three simulated signals, each containing 100 periods (also the non-periodic signals) made of 288 samples, for a total of 28800 samples. Each group corresponds to a wireless node containing three separate sensors. To each copy, a single type of anomaly (see Section 4.1) was added, to allow us to evaluate the performance of our methods per category. For the *spike* and *noise* categories, the amplitude is randomly chosen between two and five times the standard deviation of the incremental (1-step back) signal differences over a window surrounding the insertion of the anomaly. The offset for a drift anomaly is defined as half the amplitude of a total signal. A *spike* anomaly is a single sample long, while the length of a *noise* or a *constant* anomaly is randomly chosen between 10 samples and 1 period, i.e. 288 samples. The drift anomaly lasts 7 periods, where the first and the last two periods are sinusoidal ramps to and from the offset value, to provide a gradual transition to the anomalous offset.

Next to these synthetic datasets, we use three real-world datasets. The first contains the measurement traces collected from an indoor WSN setup consisting of 19 TelosB nodes, each equipped with three sensors: temperature, humidity and light. This setup collected data every 5 minutes for 5 months, resulting in 42112 samples per sensor. Because there is no ground truth for these measurements readily available, we labeled these using the semi-automated method described in the previous section. This dataset mainly contains *constant* anomalies, and a few instances of *spikes* and *drift*.

In addition, we consider two real-world datasets from two known deployments of WSN, namely the Sensorscope project [75] and the Intel Berkeley Lab [76]. As for Sensorscope, we use the dataset from the Grand-St-Bernard WSN deployed in 2007. This deployment consisted of 23 nodes, each one having 9 sensors, measuring environmental parameters such as temperature, humidity, solar radiation, soil moisture, rain and wind. Due to varying energy consumption, the number of samples per node ranges from 19800 to 30000 samples. We should note that since we limit our fusion algorithms up to three sensors, due of memory limitations (see Section 5.1), we split this dataset into two datasets: *temperature* (including ambient and surface temperature, and humidity sensors) and *water* (including humidity, soil moisture and watermark sensors).

The Intel Berkeley Lab deployment consisted of 54 sensor nodes deployed for over a month in 2004. During this time, over 2.3 million readings were collected. Due to connectivity issues, however, the data for individual nodes are not consistently available. Therefore, we processed the sensor data into measurements with 60 second intervals, where missing samples were imputed by the previously known measurement, and labeled as anomalous. Again due to varying energy consumption, the number of samples per node ranges from 6900 to 50800 samples. Similar to the indoor WSN dataset, both this dataset and the aforementioned Sensorscope dataset contain predominantly *constant* anomalies, and relatively few instances of *spike* and *drift* anomalies. Both these datasets were also labeled using the semi-automated method described in the previous section. The main properties (no. of samples, no. of nodes and percentage of anomalous samples) of the six datasets are summarized

Table 1: List of datasets and their properties. The three synthetic datasets are considered as single dataset, with 4 copies for each anomaly category.

Dataset	#samples	#nodes	% anomalous
Synthetic	17.2 M	3*50*4	2.2%
Indoor WSN	0.8 M	19	2.7%
Sensor Scope	0.58 M	23	5.1%
Intel Lab	2.30 M	54	19.9%

in Table 1.

4.4. Evaluation metrics

A common way to evaluate detection performance on labeled datasets is through a confusion matrix [69]. This matrix indicates the following four values: the true positives (TP), i.e., the anomalies that are detected; the false positives (FP), i.e. the normal samples that are falsely detected as anomalous; the false negatives (FN), i.e., the anomalous samples that are not detected; and, finally, the true negatives (TN), i.e. the samples that are normal and also classified as being normal.

These metrics give raw numbers of the detection performance. However, we extend these measures to take into account possibly inexact labels (as mentioned in Section 4.2) and to solve two particular issues which are due to the structure of our classifiers (as depicted in Figure 1). The first issue is a consequence of using sensor measurements $\mathbf{x} \setminus x_0$ to estimate and classify the target sensor value x_0 (see Section 3). Namely, the prediction error could be classified as anomalous not because there is an anomaly present in the data measured by the sensor x_0, but because there is an anomaly in one or more of the sensors $\mathbf{x} \setminus x_0$ (which in turn leads to a mismatch between x_0 and its predicted value $\hat{x_0}$). The second issue is the presence of delayed anomaly detections. These can occur when, for instance, a *spike* anomaly is classified as normal based on a learned model, but after the model is updated, with such anomalous data, the next prediction is so erroneous that it is classified as anomalous.

To overcome the first issue, we extend the definition of a TP in such a way that a detection is regarded TP when an anomaly occurs in any of the input sensors. In other words, we argue that a detection, caused by an anomaly in any of the sensors of a node, but that is not caused by an anomaly in the sensor being classified, is correct and should be viewed as a valid, true positive (TP), detection. To overcome the second issue, we assume that if a true anomaly lies within a few samples of the detection, this detection can still be considered valid (since it might prompt the data analyst to investigate the data around the flagged sample), therefore we extend the definition of TP to a *context* window, in the form of a window of length L_b around the detection. The effect of this mechanism is thoroughly evaluated in Section 5.4.

In addition to the raw (extended) measures, we also use more insightful statistical measures of performance (that are based upon these raw numbers), namely precision, recall, and the F-measure [69]. Precision is the ratio of correct detections over all detections, that is TP/(TP + FP), and indicates the percentage of relevant detections. Recall is the percentage of anomalies actually detected, or TP/(TP + FN). The F-measure is given by $(2 \times \text{precision} \times \text{recall})/(\text{precision} + \text{recall})$, indicating a single, weighted average, measure of performance.

Finally, because we use predictors as the basis of our classifiers, we also evaluate the prediction accuracy under normal conditions. We do this by analyzing the root-mean-square error (RMSE) of the prediction over a sequence of synthetic data without anomalies. We assume that RMSE close to the standard deviation of the noise of the data can indicate better detection performance for a classifier than when it does not resemble the standard deviation.

5. Results

We start this section by discussing the complexity of the methods that were introduced in Section 3. Section 5.2 then presents an analysis of the methods' parameter values that yield the best method performance (in terms of precision and recall). This is followed by an evaluation of the predictions made by all the methods. The next section evaluates the effect of the extension of the definition of a TP by a context window. Finally, Section 5.5 presents a detailed comparison of the performance of the online decentralized anomaly detectors with the baseline methods.

5.1. Complexity

In earlier work, we analytically established and measured the time complexity of the different anomaly detection methods [33, 34, 35]. This analysis is summarized in Table 2. Using MSPsim [77], a simulator for a WSN platform, the number of instruction cycles was empirically found to be comparable to those indicated by the aforementioned analysis. However, we do have to note that the fixed point arithmetic from `libfixmath` [56] and `libfixmatrix` [57] adds a significant overhead (a constant scale factor) which is not taken into account in the analysis of complexity.

From Table 2, we particularly note that when RLS is used in the *fusion* setup, where the predictions of other methods are added as inputs to the models, its quadratic complexity becomes significant. This is similar to the case of OS-ELM, where also the number of hidden nodes influences complexity. In terms of space complexity, we should note that both methods require several matrices of size $O(d^2)$ to be kept in memory. This entails that, due to the limited memory available on the TelosB platform we use in our tests, in the *fusion* setup we must limit the number of sensors (in our experiments, three).

Table 2: Comparison of order of complexity derived from algorithms. Here, L_h is the window length of the window mean, d the number of inputs, $\tilde{N}$ is the number of hidden nodes, deg the degree of the polynomial of the function approximation (FA), L_s the window length for FA, and c the number of classifiers.

Method	Window	RLS	OS-ELM	FA	Ensemble
CPU	$O(L_h)$	$O(d^2)$	$O(\tilde{N}d + \tilde{N}^2)$	$O(deg^2 + L_s)$	$O(c)$
Memory	$O(L_h)$	$O(d^2)$	$O(\tilde{N}d + \tilde{N}^2)$	$O((deg + 2)^2 + L_s)$	$O(c)$

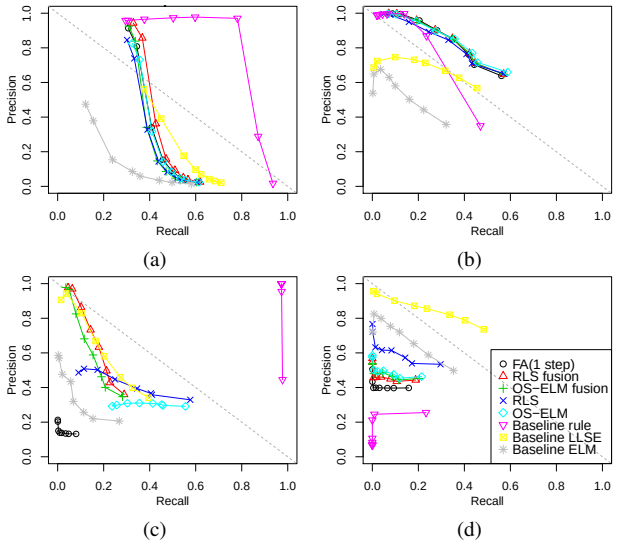

Figure 9: Precision-recall curves for several of the individual classification methods. The different points on the curves represent the confidence intervals from 68.27% (more recall) to 99.73% (more precision), for the anomaly types (a) spike, (b) noise, (c) constant and (d) drift. The legend holds for all sub figures.

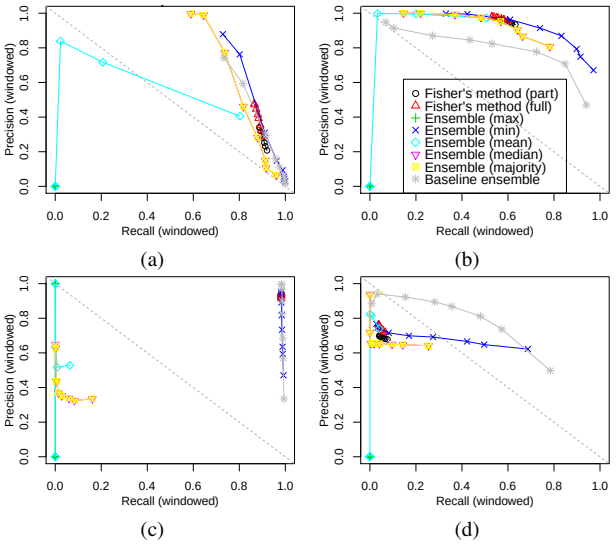

Figure 10: Precision-recall curves for the ensemble methods, and sending a window of data around the detection of length $L_b = 3$. The different points on the curves represent the confidence intervals from 68.27% (more recall) to 99.73% (more precision) for the anomaly types (a) spike, (b) noise, (c) constant and (d) drift. The legend holds for all sub figures.

5.2. *Parameter analysis*

A common parameter used in all the classifiers described in Section 3 is the confidence interval. More specifically, each classifiers outputs a p-value, that as we have seen indicates the confidence with which a sample is classified as anomalous or not. Indeed, in order to evaluate the anomaly detection performance in terms of TP, FP, FN, and TN (see Section 4.4), a binary classification is needed, which can be obtained by assigning a confidence interval to the data considered to be "normal". To assess what is a "reasonably good" confidence interval, we conduct a preliminary analysis on the synthetic datasets, where we know the ground truth classification. We generate precision-recall curves for the different anomaly detection methods and confidence intervals of 68.27, 75.00, 80.00, 86.64, 90.00, 95.00, 99.00, and 99.73%.

Figure 9 shows the results divided by anomaly type. We can immediately observe that the performance varies greatly with the type of anomaly (see Section 5.5 for details). However, the effect of selecting a specific confidence interval as threshold behaves as expected from the definition of the p-value, being a measure for how likely a sample belongs to the normal class. That is, the higher the confidence interval, the more precise the detection is, but the fewer anomalies are recalled. From these results we conclude that a confidence interval of 95% (i.e., p-value < 0.05), gives a reasonable trade-off between precision and recall.

To show the effect of the confidence level on the ensemble classifiers (both online and offline baselines), we repeat this analysis also on them, including in this case a *context* window around the anomalies of length $L_b = 3$, i.e., a context of data that is close to the detection in terms of time: this means that a detection on one sample prior or one sample after the actual label of the anomaly is regarded TP (recalling the discussion in Section 4.4). We further discuss the effect of the context window in Section 5.4 and the benefit of ensembles in Section 5.5. The resulting precision-recall curves can be seen in Figure 10. Once again we observe that the detection performance per anomaly type varies, but using ensembles and accounting for context has a positive effect on both recall and precision compared to the individual methods presented in Figure 9. Most importantly, however, also here a confidence interval of 95% results in a good balance between precision and recall for further evaluation of the detection performance. That is, over all the different types of anomaly categories, this confidence interval results in a reasonable recall percentage, with reasonable precision. Nevertheless, depending on the application, an end user might accept a different trade-off where, for instance, precision is much preferred over recall.

Besides the confidence interval and the context length, there are other classifier-specific parameters that influence the performance of the methods presented here. In earlier work we established the parameter setting for both RLS and ELM that gives a good trade-off between performance and complexity [33, 34]. RLS has a single free parameter, the forgetting factor, α, with should be set to $\alpha = 10$. ELM, on the other hand, has several free parameters, including the learning rate, the number of hidden nodes and the activation function. The method shows good performance with a learning rate of 0.999985, $\tilde{N} = 4$ hidden nodes, and a sigmoid activation function. In addition to these parameters, we can vary the window sizes of the function approximation (L_s) and the sliding window mean (L_h). To show their effect, we have chosen a number of possible settings for L_h and L_s that are applicable to the limited resources available in WSN. Because we know that the labeling in the synthetic data are correct, we aggregate the precision and recall of these datasets. The resulting precision and recall (corresponding to a

95% confidence interval) are plotted in Figure 11. For the function approximation window, we see an average recall of around 20%, that slowly decreases as L_s increases. The effect on the precision, however, is more significant. As L_s increases, the precision increases too. This increase stabilizes after $L_s = 20$, and given the marginal effect on the recall, this is a good choice for L_s. A similar effect can be observed for the sliding window length L_h in Figure 11d. The effect on recall is marginal, with an average recall around 15%, but the increase of precision is strongest up to $L_h = 48$.

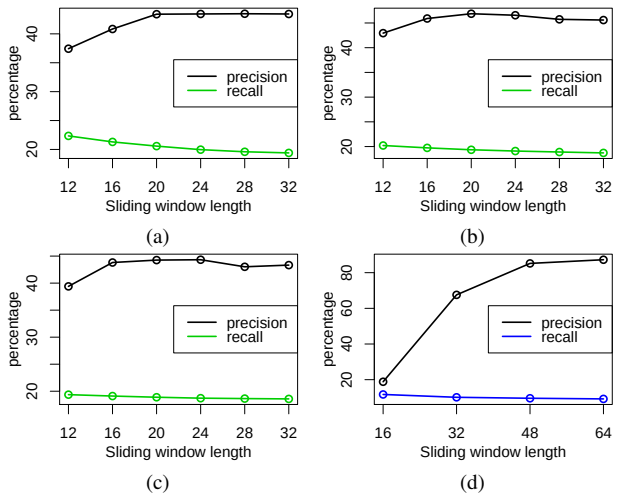

Figure 11: Effect of window length on anomaly detection performance of single-dimensional time-series classifiers, evaluated on the synthetic data. The effect of L_s is shown for the (a) 1-step-ahead, (b) 2-step-ahead and (c) 3-step-ahead classifier. The effect of L_h on the windowed statistics classifier is shown in (d).

With these parameters established, we can analyze the accuracy of the predictions made by the methods, and the anomaly detection performance based on these predictions, as shown in the following sections.

5.3. Prediction errors

The prediction error analysis method, outlined in Section 3.6, assumes errors to be normally distributed, although the parameters of this distribution might change over time. Furthermore, it also assumes that if the distribution has similar properties to the Gaussian distribution (i.e., it is bell-shaped), the method should still perform reasonably. If, however, the prediction error distribution shows a mixture of distribution models or very distinct peaks, which are not due to anomalies, the method might give unreliable results.

We have performed an offline analysis of the prediction error to get a general idea of its distribution. For each combination of method, dataset, node and sensor within that node, we have uniformly sampled the prediction error for that combination and aggregated these prediction error samples per dataset. We then analyzed the distribution of the resulting aggregation. Due to the large number of combinations, we only show a few results in Figure 12. These show that the error distribution does not closely follow a normal distribution, but seems to be very much centered within the fitted normal distribution and, therefore, the

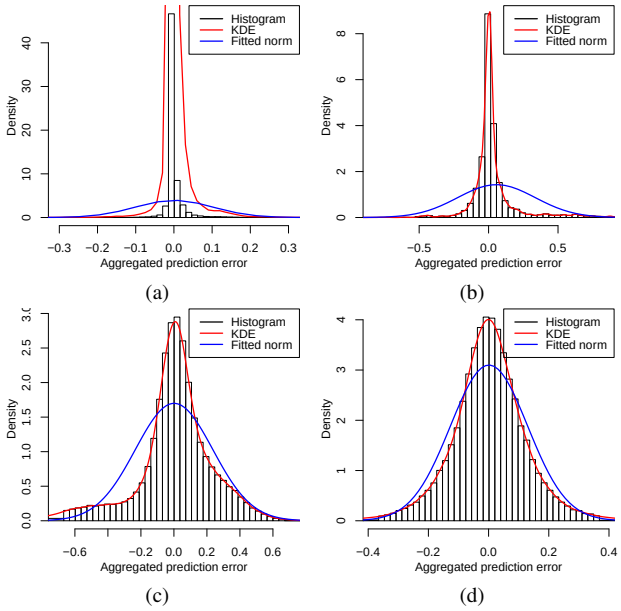

Figure 12: Comparison of different prediction error distributions with fitted normal distributions and Kernel Density Estimates (KDE) of the distributions. The prediction error distribution depends on dataset and on the prediction method used. (a) Intel Lab data, sliding window mean, (b) Intel Lab data, RLS-fusion. (c) Synthetic Drift data, sliding window mean, and (d) Synthetic Drift data, RLS-fusion.

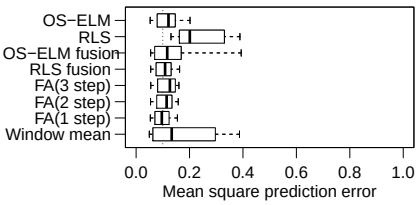

Figure 13: Comparison of the mean square prediction error of the different methods on the synthetic data, who has zero mean Gaussian noise with a variance of 0.1.

p-values are expected to reflect a reasonable outlier score. Furthermore, the distribution parameters vary greatly with dataset and type of sensor, but in all cases the prediction error distribution does display a single peak with a bell-shaped curve.

As Figure 12 shows, the prediction errors generated by applying the detection methods to the synthetic data can exhibit a close-to-normal distribution, which can be expected from the synthesis method of this data, which uses Gaussian noise. Furthermore, because in all synthetic datasets we have added a Gaussian component with zero mean and 0.1 variance, we expect the average mean square error to be no better than 0.1 units. This is shown in Figure 13 where, for each method, we have summarized the mean square prediction error per node in a box plot. Moreover, we can see that the performance of the sliding window mean predictor varies most, which can be expected from the short window over which the mean is estimated. The larger difference in performance of OS-ELM fusion can be attributed to the random initialization of input node weights, combined with fixed-point precision.

5.4. Effect of a detection context window

We now turn our attention to the effect of context around the detected anomalous measurement (i.e., if a labeled anomaly lies within a *context* window around a classified anomaly, the latter can be regarded a TP). As discussed in Section and Section 5.2, here we assume the context window to include the data samples right before and after a sample classified as anomalous ($L_b = 3$).

We have estimated the performance of the anomaly detection methods when applied to the test datasets and compared the results with and without using a context window. These results are shown in Figures 14 to 17. Figures 14 and 15 show the results based on the synthetic datasets. When we take a context window into account, we obtain an almost doubling of the TP ratio for all methods, and for all types of anomalies other than the constant anomaly. The effect of the context window is less noticeable on the real-world data shown in Figures 16 and 17. Conceivably, this can mainly be attributed to the fact that most of the anomalies in the real-world data are constant anomalies. The constant classifier therefore has a high ratio of TP, which is reflected in the performance of the ensemble methods. On the other hand, the FP ratio of the constant classifier is much higher for the real-world data than for the synthetic data, which can be attributed to constant natural signals, such as low light intensity signals at night.

The impact of using a context window over the anomaly detection performance is, on the whole, significant. This is consistent with our hypothesis that an anomaly does not only show when it differs from a prediction, but it can also have an effect on the model's prediction after it is updated with the anomalous data. Transmitting the context window has the added benefit of allowing one to determine the system's behavior before and after an anomaly is detected. In addition, one could also apply a post-processing step to identify the actual anomalous sample. Overall, many anomalies are detected within the context window using this online decentralized approach.

5.5. Anomaly detection performance

Lastly, we evaluate the anomaly detection performance of our methods by analyzing the results according to the metrics described in Section 4.4. We measure those metrics on both the online and offline (baseline) methods with application of the context window, as described in the previous section. These results are presented in Figures 15 and 17 and Table 3 (it is worth noting that the TP ratio shown in the figures is equal to the recall rate presented in the table, whereas the TP divided by the sum of TP and FP ratio can be related to the precision). We first evaluate the performance of the different methods for the synthetic datasets. Then, the methods are evaluated on three real-world datasets.

Synthetic datasets

As remarked in Section 5.2, there is a performance difference in detecting anomalies of different types, as clearly shown in Figure 15. We observe, for instance, that the constant classifier, as expected from its nature, has very high performance for *constant* anomalies in the synthetic data, with over 98% recall and zero FP, which is equal to 100% precision. This performance is also reflected in some ensemble methods, mainly the heuristic ensemble, those based on Fisher's method and the minimum p-value ensemble. These ensembles, however, do have 1.6% to 12.0% FP due to the inclusion of other individual classifiers. Among the latter, the RLS and ELM classifiers show over 20% TP ratio, but also a high amount of FP. Both effects can be attributed to online learning, which adapts slowly to constant anomalies. The FA classifier shows very few TP. This, we suspect, is due to the window length ($L_s = 20$) over which the function is approximated, which is almost as large as the duration of the average constant anomaly. The RLS and ELM fusion classifiers fuse the FA predictions together with window mean predictions, as shown in the TP and FP ratios. Compared to the baselines, the constant rules perform equally. On the other hand, centralized offline analysis is advantageous for the LLSE and ELM classifiers which, compared to RLS and OS-ELM, have better precision, albeit with lower recall.

The *spike* detection results in Figure 15 show that function approximation, RLS and ELM perform similarly, with around 77-83% recall. Increasing the length of the prediction horizon in FA has a small negative effect on the recall ratio, because the further ahead in the future a prediction is, the less relevant it becomes, thus increasing the average prediction error and the detection threshold. All methods have similar precision a little over 50%. Here, again, heuristic, Fisher's and minimum p-value ensemble methods outperform the individual classifiers in terms of true detection, gaining up to 7%. But, in the case of Fisher's method (part), which combines a partial set of classifiers, and the minimum p-value ensemble, the precision is cut by half. A similar performance can be observed for the baselines (other than the baseline rule), that also have low precision but high recall. The majority voting and median ensemble behave equally (as expected from the median voter theorem [70]), but have relatively low TP ratio, similar to the offline rule baseline. The positive effect of voting, however, results in the relatively high precision. Overall, if we calculate the F-measure, the median/majority voting ensemble has the best score for *spike* anomalies, outperforming even the baselines.

The detection performance for *noise* anomalies is similar to that of spike anomalies for most methods. However, because the number of samples affected by the noise anomaly is larger, the resulting TP ratio (around 40%) is less than for *spikes*. The FP ratio, on the other hand, is very low for all methods, implying a high precision (over 97%). This could indicate that the methods do signal the anomalies, although not all anomalous samples are detected. Similar to the performance on *spike* anomalies, the heuristic, Fisher's and minimum p-value ensembles also display a doubling in FP ratio, with respect to the individual classifiers, from 1.4% to 3.4% FP. But, in this case, the increase in TP ratio from 44% to around 54 to 60% for ensemble methods is more than the increase in detection for *spike* anomalies. The maximum F-measure shows that the online minimum p-value ensemble has the best average results, although Fisher's method shows similar performance. For the *noise* anomaly, again, the online methods seem to outperform

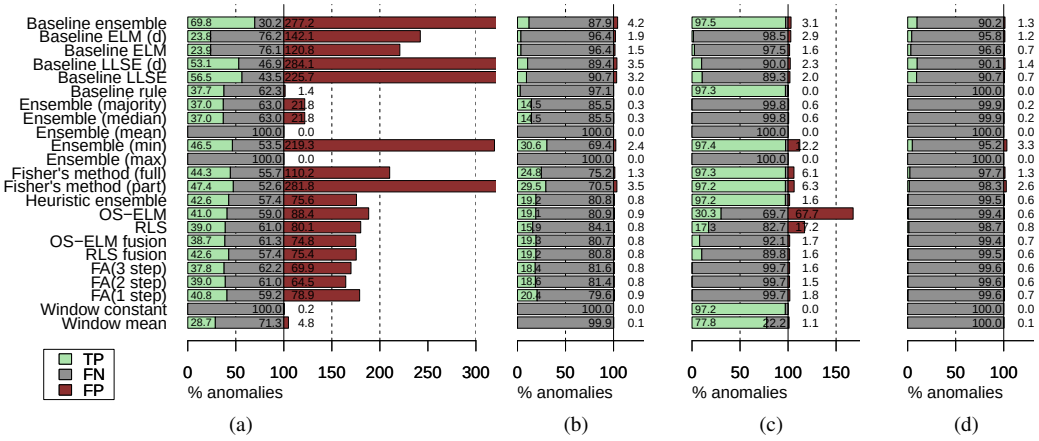

Figure 14: True Positives (TP), False Negatives (FN) and False Positives (FP) as the ratio of number of anomalies in synthetic datasets. We can distinguish the anomaly types (a) spike, (b) noise, (c) constant and (d) drift.

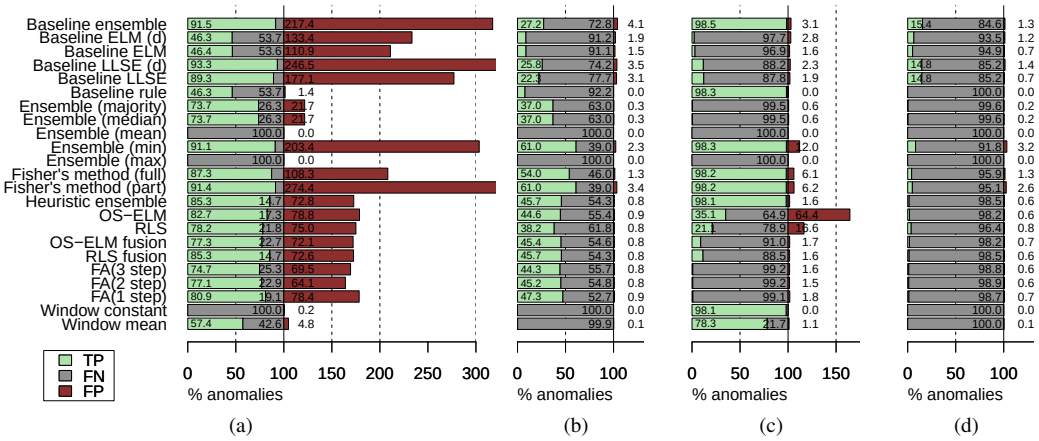

Figure 15: True Positives (TP), False Negatives (FN) and False Positives (FP) as the ratio of number of anomalies in synthetic datasets. The TP and FP are improved due to a context window of length $L_b = 3$ around a detection, as discussed in Section 4.4. We can distinguish the anomaly types (a) spike, (b) noise, (c) constant and (d) drift.

the offline baselines: while for the baseline the precision is high (99% for the rule baseline), the recall is low compared to the online methods.

The *drift* anomaly is the most difficult to detect, as we can see from the performance results shown in Figure 15. Overall the online detection methods seem to adapt to drift, but, if they detect it, they do it only in the beginning of the anomaly. The individual classifiers have a TP ratio of around 1% and a FP ratio that is only a little lower, with a precision around 65%. The ensemble methods do, also in this case, display an increased TP, but also an increased FP ratio, giving them a small benefit in recall but with equal precision. It is here that the offline baseline methods have a clear advantage. Where the online learning methods slowly adapt to the drift, the static model of LLSE and ELM generate larger prediction errors, showing in a much higher TP ratio, and a higher precision. This is reflected in the maximum F-measure for drift, seen in Table 3, which is established by the baseline ensemble with 92% precision and 15% recall. If, however, we only evaluate the online methods, the minimum p-value ensemble gives the best F-measure with 81% precision and 8% recall.

Overall, the ensembles in general do have a positive benefit on the performance. However, the max and mean p-value ensemble do not perform well at all, as they are both affected too strongly by the "least confident" classification. Another observation is that the majority voting and median ensemble schemes perform equally, as expected from the median voter theorem [70]. While their TP ratio (recall) is less than any other classifier, the number of FP produced by these ensembles is much less than other classifiers, thus resulting in relatively high precision. If the target were, however, to detect (recall) the largest number of anomalies, then the heuristic, Fisher's or minimum p-value methods are a better choice. Furthermore, these ensembles come the closest to matching the performance of the baseline (offline) ensemble.

Real-world datasets

We have performed the same analysis for the three real-world datasets, where as said we have split up the Sensorscope data in temperature-related and water-related sensors due to memory limitations. The results for these datasets are shown in Figure 17 and Table 3. From observations made after manually check-

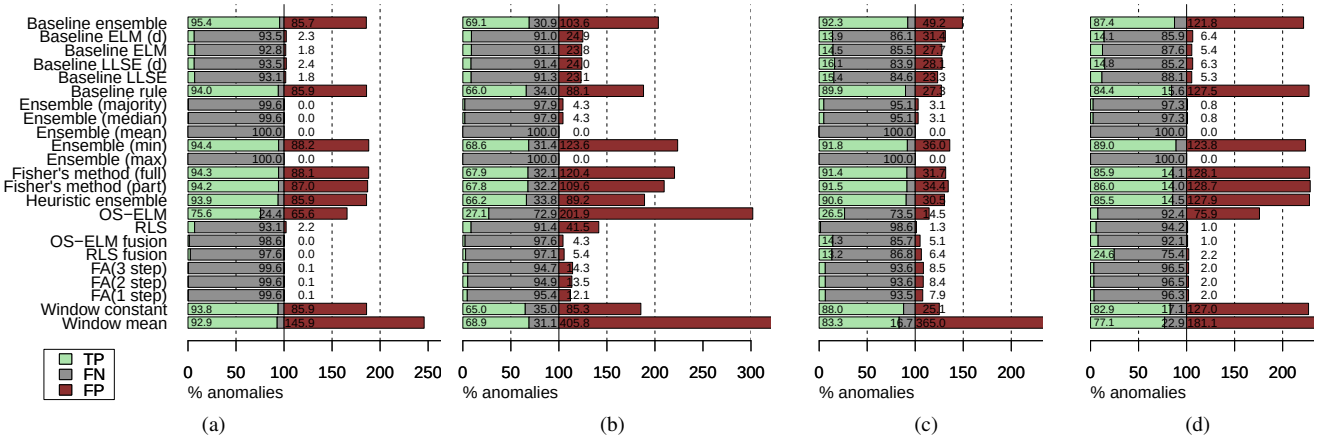

Figure 16: True Positives (TP), False Negatives (FN) and False Positives (FP) as the ratio of number of anomalies in real-world datasets. The *constant* anomaly prevails, resulting in the good scores of the constant rule and Baseline rule (offline). We can distinguish the datasets (a) Intel Lab, (b) Indoor WSN, (c) Sensorscope temperature and (d) Sensorscope water.

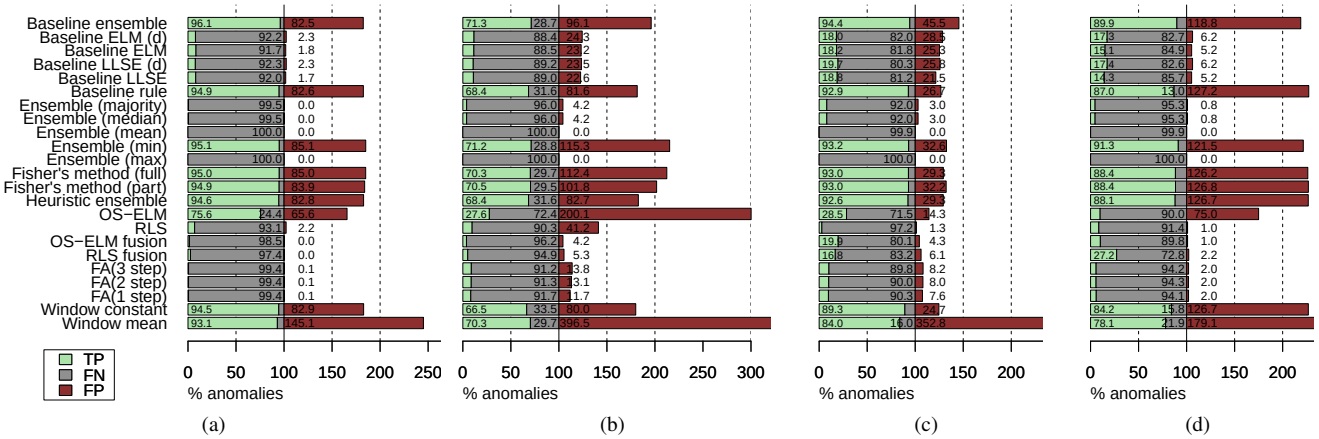

Figure 17: True Positives (TP), False Negatives (FN) and False Positives (FP) as the ratio of number of anomalies in real-world datasets. The TP and FP are improved due to a context window of length $L_b = 3$ around a detection, as discussed in Section 4.4. The *constant* anomaly prevails, resulting in the good scores of the constant rule and Baseline rule (offline). We can distinguish the datasets (a) Intel Lab, (b) Indoor WSN, (c) Sensorscope temperature and (d) Sensorscope water.

ing the labels (according to the procedure outlined in Section 4.2), we know that the predominant anomalies are constants, often resulting from missing values. This predominance is the reason for the performance of the constant classifier, which has very high TP ratios of 70% and higher. However, the FP ratio of the constant classifier is much higher for the real-world data than for the synthetic data, resulting in a precision from 40% to 78%. This is caused by stable natural signals, such as the low light intensity readings at night.

The OS-ELM-based classifier performs well across most real-world datasets, except for the Sensorscope water-related data. Its performance matches or exceeds that of the baseline ELM method in terms of recall. However, its precision is lower (from 11% to 53%), resulting in a high number of FP. The latter could be caused by the random initialization of the hidden node weights and biases. The RLS classifier, on the other hand, has slightly lower recall than its baseline counterpart, LLSE. On the Sensorscope temperature data, for example, the RLS recall is only 2.8%, while LLSE recalls around 19% of the data. However, for this dataset in particular its 68% precision is higher than the 38% of LLSE. On the other datasets, the precision is similar to LLSE. The recall performance of the FA-based classifiers seems to be low compared to the other individual classifiers, particularly for the Intel Lab data recalling less than 1%. However, their precision is on par with the other methods. Moreover, for the Sensorscope temperature data, the FA-based classifiers have better recall than RLS.

Combining the predictions of these classifiers, using the RLS or OS-ELM fusion, shows varying results. For all the real-world datasets, the precision of these fusion methods is the highest of all classifiers (see the bold face numbers of Table 3). However, for the Intel Lab and Indoor WSN datasets, the recall (with 1 to 5%) is much lower than the individual RLS or OS-ELM methods (having 7 to 27% recall). On the other hand, the fusion classifiers perform on par with the LLSE and ELM baseline classifiers for the Sensorscope datasets in terms of recall, and outperform them in terms of precision.

There are similarities in the datasets for the Intel Lab and Indoor WSN data, in terms of type of sensors and environment. This also holds, to a lesser extent, for the Sensorscope

temperature and Sensorscope water sensors. These similarities are evident from the results of the different methods where, for instance, the baseline methods for the Sensorscope datasets show higher recall than for the other real-world datasets (around 18% vs around 10%). Similarly, the individual classifiers, and mainly the fusion classifiers, show better performance for the Sensorscope data. This might be the result of the indoor (Intel Lab, Indoor WSN) versus outdoor (Sensorscope) nature of the data: changes in the signal for outdoor data may occur more slowly than changes indoor, that are affected by human activity.

Overall, a proper choice of the ensemble method is beneficial to the detection performance. However, similar to the results from the synthetic datasets, the max and mean p-value ensembles do not perform well on the real-world data. The majority voting and median p-value ensembles perform equally, and outperform the baseline ensemble in terms of precision, as clearly shown by the FP ratios. Their drawback, however, is the very low TP ratio, or recall, which is the lowest across all methods. A much better performance is achieved by the Fisher's method, the heuristic and the minimum p-value ensembles. Their results are dominated by the constant classifier, but the other methods do contribute extra TP, resulting in higher recall. The performance of the heuristic ensemble is, according to the F-measure shown by the dashed underlines in Table 3, often the best of the online ensembles, closely followed by Fisher's method and the minimum p-value ensembles. On the whole, the performance of the decentralized online learning methods in Fisher's method or minimum p-value ensembles is very close to the baseline ensemble, showing that ensembles of classifiers are a viable approach to anomaly detection in resource-limited systems.

6. Discussion and conclusions

In this paper we introduced a lightweight, application-independent, framework for online anomaly detection in IoT applications, such as wireless sensor networks, based on incremental learning. The framework addresses two key challenges: 1) the realization of decentralized, automated, online learning methods to detect anomalies, individually or in an ensemble, within extremely limited hardware resources; and 2) the performance evaluation of these decentralized online learning methods.

Anomaly detection was realized in two steps. First, we developed a heterogeneous set of local online learning classifiers, each one composed of a predictor and a decision making component. All our local classifiers perform an incremental online (unsupervised) learning, which allows them to automatically recognize anomalies in (acquired or sensed) data without any a priori knowledge, assumptions or pre-defined rules. Second, we combined multiple and diverse individual classifiers in an *ensemble*, which we hypothesized (and confirmed through experimentation) would further improve the anomaly detection accuracy, overcoming the limitations of the individual classifiers. In the ensemble, the individual classifiers act in parallel on the same data, while their classifications can be aggregated either by using simple heuristic rules (which order the independent classifications according to the most likely kind of anomaly), by applying algebraic combiners, such as the median, or by applying the Fisher's method.

In order to assess the performance impact of decentralizing anomaly detection, we extensively evaluated the proposed decentralized ensemble, as well as the individual online learning methods, and compared to their centralized offline counterparts, taken as baseline. The evaluation was performed using various large synthetic and real-world datasets and was based on prediction accuracy and confusion matrix metrics. In the latter we have accounted for false positives caused by anomalies in correlated sensors, and for false positives caused by a delayed detection. Our experiments verified the general viability of the local classifiers and the ensemble. They also showed that the performance of both the offline centralized and online decentralized methods largely depends on the datasets and the kinds of anomaly to be detected. A general trend is that individual online learning methods are characterized by a reduced recall, while their precision is often better than that of their offline counterparts. Considering known anomaly types (spike, noise, constant and drift), offline centralized methods seem to be more suitable for detecting slow long-term effects such as drift, while noise anomalies are best detected with online decentralized methods. Moreover, depending on the application goal (e.g., detecting the most anomalies, detecting with few false positives, etc.), different combinations of online classifiers might be appropriate. The rule-based constant classifier is computationally cheap and performs well. If the most frequent anomalies for an application relate to *spike*, *noise* or, to a lesser extent, *drift*, then an efficient choice might be the RLS-based classifier.

The main benefit of our proposed framework, however, derives from the combination of classifiers as prediction fusion or ensembles. By combining the classifications of the different learning techniques, the average performance of the online combinations is increased for all datasets, approaching that of the offline ensembles. For instance, the bold-face numbers in Table 3 show that the RLS and OS-ELM-based prediction fusion classifiers show the best precision across the real-world data. And, notably, among the different tested combination mechanisms, the Fisher's and the minimum p-value methods make the online ensembles match the performance of their offline counterparts. However, in terms of recall, the simpler heuristic or minimum p-value classifiers gives the best F-measure, shown by the dashed underlines in Table 3. For the most accurate detection precision, a majority voting or median p-value ensemble scheme delivers the best precision (at the cost of low recall), whereas the most reliable performance across all datasets seems to be given by the ensemble based on Fisher's method. Overall, the online methods can provide a valid alternative to the offline baseline methods, to detect anomalies in an online, decentralized manner.

These results show that online learning anomaly detection methods can be implemented in a decentralized framework, and achieve similar performance to offline centralized baselines. While the latter often make use of large resources in terms of storage, computation and (human) expert knowledge to optimally configure the detection system, our proposed framework constitutes a computationally cheap, completely autonomous

Table 3: The precision/recall results. Confidence level > 95%, $L_h = 48$, $L_s = 20$, $L_b = 3$. The baselines followed with (d) include a delayed version of the signal. The bold-face numbers indicate the maximum precision or recall. The underlined numbers are the best combination of precision and recall, the dashed underlined numbers are the best online embedded combinations (both calculated according to the F-Measure).

Classifier \ Dataset	Intel Lab		Indoor WSN		Sensorscope temperature		Sensorscope water		Constant		Drift		Noise		Spike	
	pr.	re.	pr.	re.	pr.	re.	pr.	re.	pr.	re.	pr.	re.	pr.	re.	pr.	re.
Window mean	39.07	93.05	15.07	70.33	19.23	84.01	30.37	78.11	98.66	78.31	21.63	0.02	62.16	0.15	92.26	57.42
Window constant	53.29	94.54	45.39	66.50	78.35	89.26	39.93	84.23	**100.00**	98.09	48.28	0.00	23.08	0.00	0.00	0.00
FA(1 step)	88.37	0.58	41.40	8.27	56.15	9.67	74.72	5.91	33.61	0.92	65.95	1.29	98.15	47.25	50.78	80.91
FA(2 step)	88.00	0.57	39.98	8.74	55.38	9.98	74.35	5.73	34.41	0.81	66.66	1.11	98.34	45.19	54.59	77.09
FA(3 step)	85.36	0.60	38.98	8.83	55.43	10.20	73.97	5.78	34.45	0.84	65.88	1.15	98.20	44.27	51.80	74.69
RLS fusion	**99.39**	2.58	**48.93**	5.11	73.29	16.78	**92.53**	27.17	87.73	11.45	71.02	1.52	98.24	45.69	54.05	85.35
OS-ELM fusion	97.80	1.47	47.08	3.78	**82.15**	19.86	91.02	10.20	84.37	9.03	72.80	1.79	98.20	45.36	51.72	77.26
RLS	76.08	6.91	19.03	9.67	68.59	2.83	89.56	8.61	55.92	21.09	82.77	3.64	97.88	38.18	51.02	78.18
OS-ELM	53.55	75.63	12.12	27.60	66.58	28.51	11.74	9.97	35.27	35.09	74.24	1.84	98.04	44.64	51.21	82.67
Heuristic ensemble	53.32	94.61	45.27	68.43	75.99	92.58	41.01	88.07	98.39	98.14	70.97	1.52	98.23	45.69	53.98	85.35
Fisher's method (part)	53.07	94.92	40.91	70.46	74.27	93.00	41.10	88.44	94.05	98.20	65.07	4.87	94.68	60.96	24.98	91.37
Fisher's method (full)	52.78	95.02	38.49	70.31	76.04	92.97	41.18	88.35	94.17	98.20	75.51	4.11	97.65	53.97	44.64	87.32
Ensemble (max)	100.00	0.00	0.00	0.00	0.00	0.00	100.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Ensemble (min)	52.78	95.06	38.17	71.19	74.09	93.18	42.92	**91.34**	89.09	98.28	71.55	8.17	96.35	**60.98**	30.93	91.08
Ensemble (mean)	100.00	0.00	0.00	0.00	100.00	0.06	100.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Ensemble (median)	97.23	0.53	48.52	3.96	72.69	7.96	85.37	4.71	43.59	0.46	65.46	0.39	99.27	36.99	77.24	73.69
Ensemble (majority)	97.23	0.53	48.52	3.96	72.69	7.96	85.37	4.71	43.59	0.46	65.46	0.39	99.27	36.99	77.24	73.69
Baseline rule	53.47	94.89	45.62	68.42	77.68	92.90	40.62	86.98	99.97	98.32	37.20	0.01	**99.69**	7.80	**97.15**	46.34
Baseline LLSE	82.19	8.05	32.73	11.00	46.69	18.80	73.27	14.30	86.27	12.24	**95.31**	14.78	87.79	22.34	33.51	89.26
Baseline LLSE (d)	76.73	7.65	31.59	10.83	43.22	19.65	73.73	17.35	83.96	11.80	91.62	14.81	88.09	25.78	27.46	**93.30**
Baseline ELM	82.48	8.33	33.15	11.49	41.79	18.15	74.26	15.13	65.96	3.11	87.69	5.14	85.66	8.86	29.51	46.44
Baseline ELM (d)	77.25	7.78	32.40	11.63	38.72	18.00	73.64	17.27	45.07	2.33	84.58	6.45	81.99	8.80	25.75	46.26
Baseline ensemble	53.82	**96.11**	42.58	**71.26**	67.48	**94.39**	43.06	89.88	96.98	**98.49**	92.25	**15.44**	87.03	27.24	29.62	91.49

decentralized anomaly detection system, capable of obtaining acceptable performance without much a priori knowledge of the application or context.

Due to the limited-resource environment that is targeted by our work, there are, of course, some limitations in the use of these methods. These limitations mainly concern memory usage: as the memory complexity of the proposed methods grows with the number of inputs (i.e., sensors), the number of inputs that can be processed is inherently bounded by the amount of memory that is available on the sensor platform. However, this limitation is largely compensated by the reduced cost of communicating data over a WSN for central analysis.

Our work may be further developed in various directions. The most obvious evolution would be to consider the decentralization of other machine learning techniques, to pursue further gains in terms of memory requirement and detection accuracy. Even more promising would be the incorporation of local neighborhood data, in such a way that spatially-correlated information can be combined to further improve the detection performance. Collaborative learning techniques, such as transfer learning among neighboring nodes, have considerable potential since they will enhance the horizon of individual nodes and, in turn, accuracy and prediction capability. We believe that, in the context of IoT, such a distributed approach will eventually enable the deployment of large-scale smart networks that will be able to sense their environment and report to users only the relevant information.

Acknowledgment

INCAS[3] is co-funded by the Province of Drenthe, the Municipality of Assen, the European Fund for Regional Development and the Ministry of Economic Affairs, Peaks in the Delta.

References

[1] D. Miorandi, S. Sicari, F. D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Networks 10 (7) (2012) 1497 – 1516.

[2] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645–1660.

[3] L. Bencini, F. Chiti, G. Collodi, D. Di Palma, R. Fantacci, A. Manes, G. Manes, Agricultural monitoring based on wireless sensor network technology: real long life deployments for physiology and pathogens control, in: Sensor Technologies and Applications, 2009. SENSORCOMM'09. Third International Conference on, IEEE, 2009, pp. 372–377.

[4] K. Martinez, J. K. Hart, R. Ong, Deploying a wireless sensor network in iceland, in: GeoSensor Networks, Springer, 2009, pp. 131–137.

[5] M. Reyer, S. Hurlebaus, J. Mander, O. Ozbulut, Design of a wireless sensor network for structural health monitoring of bridges, in: S. C. Mukhopadhyay, J.-A. Jiang (Eds.), Wireless Sensor Networks and Ecological Monitoring, Vol. 3 of Smart Sensors, Measurement and Instrumentation, Springer Berlin Heidelberg, 2013, pp. 195–216.

[6] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G. P. Jesi, R. L. Cigno, L. Mottola, A. L. Murphy, et al., Is there light at the ends of the tunnel? wireless sensor networks for adaptive lighting in road tunnels, in: Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on, IEEE, 2011, pp. 187–198.

[7] M. Lin, Y. Wu, I. Wassell, Wireless sensor network: Water distribution monitoring system, in: Radio and Wireless Symposium, 2008 IEEE, 2008, pp. 775–778.

[8] H. Alemdar, C. Ersoy, Wireless sensor networks for healthcare: A survey, Computer Networks 54 (15) (2010) 2688–2710.

[9] J. Åkerberg, M. Gidlund, T. Lennvall, J. Neander, M. Björkman, Efficient integration of secure and safety critical industrial wireless sensor networks, EURASIP Journal on Wireless Communications and Networking 2011 (1) (2011) 1–13.

[10] R. Jurdak, X. Wang, O. Obst, P. Valencia, Wireless sensor network anomalies: Diagnosis and detection strategies, Intelligence-Based Systems Engineering (2011) 309–325.

[11] C. C. Aggarwal, N. Ashish, A. Sheth, The internet of things: A survey from the data-centric perspective, in: Managing and mining sensor data, Springer, 2013, pp. 383–428.

[12] S. Rajasegarar, C. Leckie, M. Palaniswami, Anomaly detection in wireless sensor networks, Wireless Communications, IEEE 15 (4) (2008) 34–40.

[13] Y. Yao, A. Sharma, L. Golubchik, R. Govindan, Online anomaly detection for sensor systems: A simple and efficient approach, Performance Evaluation.

[14] J. Cabrera, C. Gutiérrez, R. Mehra, Ensemble methods for anomaly detection and distributed intrusion detection in mobile ad-hoc networks, Information Fusion 9 (1) (2008) 96–119.

[15] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys (CSUR) 41 (3) (2007/2009) 1–58.

[16] R. V. Kulkarni, A. Forster, G. K. Venayagamoorthy, Computational intelligence in wireless sensor networks: A survey, Communications Surveys & Tutorials, IEEE 13 (1) (2011) 68–96.

[17] A. Föerster, A. L. Murphy, Machine learning across the wsn layers (2010).

[18] M. A. Alsheikh, S. Lin, D. Niyato, H.-P. Tan, Machine learning in wireless sensor networks: Algorithms, strategies, and applications, Communications Surveys & Tutorials, IEEE 16 (4) (2014) 1996–2018.

[19] J. Predd, S. Kulkarni, H. Poor, Distributed learning in wireless sensor networks, Signal Processing Magazine, IEEE 23 (4) (2006) 56–69.

[20] D. Talia, P. Trunfio, How distributed data mining tasks can thrive as knowledge services, Commun. ACM 53 (7) (2010) 132–137.

[21] G. Iacca, Distributed optimization in wireless sensor networks: an island-model framework, Soft Computing 17 (12) (2013) 2257–2277.

[22] A. Liotta, The cognitive net is coming, IEEE Spectrum 50 (8) (2013) 26–31.

[23] G. P. Joshi, S. Y. Nam, S. W. Kim, Cognitive radio wireless sensor networks: Applications, challenges and research trends, Sensors 13 (9) (2013) 11196–11228.

[24] M. Paskin, C. Guestrin, J. McFadden, A robust architecture for distributed inference in sensor networks, in: Proceedings of the 4th international symposium on Information processing in sensor networks, IEEE Press, 2005, p. 8.

[25] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, S. Madden, Distributed regression: an efficient framework for modeling sensor network data, in: Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on, IEEE, 2004, pp. 1–10.

[26] J. B. Predd, S. R. Kulkarni, H. V. Poor, Regression in sensor networks: Training distributively with alternating projections, in: Optics & Photonics 2005, International Society for Optics and Photonics, 2005, pp. 591006–591006.

[27] L. Pirmez, F. Delicato, P. Pires, A. Mostardinha, N. de Rezende, Applying fuzzy logic for decision-making on wireless sensor networks, in: Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International, 2007, pp. 1–6.

[28] T. A. Nguyen, M. Aiello, K. Tei, A decentralized scheme for fault detection and classification in wsns, in: The 1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA 2013, Work in Progress session), 2013, pp. 1–4.

[29] S. Rajasegarar, C. Leckie, M. Palaniswami, J. Bezdek, Distributed anomaly detection in wireless sensor networks, in: Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on, IEEE, 2006, pp. 1–5.

[30] H. Kumarage, I. Khalil, Z. Tari, A. Zomaya, Distributed anomaly detection for industrial wireless sensor networks based on fuzzy data modelling, Journal of Parallel and Distributed Computing 73 (6) (2013) 790–806.

[31] C. Lo, J. Lynch, M. Liu, Reference-free detection of spike faults in wireless sensor networks, in: Resilient Control Systems (ISRCS), 2011 4th International Symposium on, IEEE, 2011, pp. 148–153.

[32] R. Rajagopal, X. Nguyen, S. C. Ergen, P. Varaiya, Distributed online simultaneous fault detection for multiple sensors, in: Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on, IEEE, 2008, pp. 133–144.

[33] H. H. W. J. Bosman, A. Liotta, G. Iacca, H. J. Wortche, Anomaly detection in sensor systems using lightweight machine learning, in: Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on, IEEE, 2013, pp. 7–13.

[34] H. H. W. J. Bosman, A. Liotta, G. Iacca, H. J. Wortche, Online extreme learning on fixed-point sensor networks, in: Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on, IEEE, 2013, pp. 319–326.

[35] H. H. W. J. Bosman, G. Iacca, H. J. Wörtche, A. Liotta, Online fusion of incremental learning for wireless sensor networks, in: Data Mining Workshop (ICDMW), 2014 IEEE International Conference on, 2014, pp. 525–532.

[36] R. Biuk-Aghai, Y.-W. Si, S. Fong, P.-F. Yan, Individual movement behaviour in secure physical environments: Modeling and detection of suspicious activity, in: L. Cao, P. S. Yu (Eds.), Behavior Computing, Springer London, 2012, pp. 241–253.

[37] C. Phua, V. Lee, K. Smith, R. Gayler, A comprehensive survey of data mining-based fraud detection research, Arxiv preprint arXiv:1009.6119.

[38] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, in: Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on, IEEE, 2011, pp. 385–392.

[39] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, M. Rajarajan, A survey of intrusion detection techniques in cloud, Journal of Network and Computer Applications 36 (1) (2013) 42–57.

[40] K. Tiwari, M. Arora, D. Singh, An assessment of independent component analysis for detection of military targets from hyperspectral images, International Journal of Applied Earth Observation and Geoinformation 13 (5) (2011) 730–740.

[41] S. Kao, A. Ganguly, K. Steinhaeuser, Motivating complex dependence structures in data mining: A case study with anomaly detection in climate, in: 2009 IEEE International Conference on Data Mining Workshops, IEEE, 2009, pp. 223–230.

[42] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[43] L. Zeng, L. Li, L. Duan, K. Lu, Z. Shi, M. Wang, W. Wu, P. Luo, Distributed data mining: a survey, Information Technology and Management 13 (4) (2012) 403–409.

[44] A. Bordes, S. Ertekin, J. Weston, L. Bottou, Fast kernel classifiers with online and active learning, The Journal of Machine Learning Research 6 (2005) 1579–1619.

[45] Y. Zhang, N. Meratnia, P. Havinga, Outlier detection techniques for wireless sensor networks: A survey, Communications Surveys & Tutorials, IEEE 12 (2) (2010) 159–170.

[46] M. Xie, S. Han, B. Tian, S. Parvin, Anomaly detection in wireless sensor networks: A survey, Journal of Network and Computer Applications.

[47] D.-I. Curiac, C. Volosencu, Ensemble based sensing anomaly detection in wireless sensor networks, Expert Systems with Applications 39 (10) (2012) 9087–9096.

[48] M. Chang, A. Terzis, P. Bonnet, Mote-based online anomaly detection using echo state networks, in: Distributed Computing in Sensor Systems, Springer, 2009, pp. 72–86.

[49] T. A. Nguyen, D. Bucur, M. Aiello, K. Tei, Applying time series analysis and neighbourhood voting in a decentralised approach for fault detection and classification in wsns, in: Proceedings of The 4th International Symposium on Information and Communication Technology, ACM, 2013, pp. 234–241.

[50] H. Sorenson, Least-squares estimation: from Gauss to Kalman, Spectrum, IEEE 7 (7) (1970) 63–68.

[51] W. Barbakh, Y. Wu, C. Fyfe, Online clustering algorithms and reinforcement learning, in: Non-Standard Parameter Adaptation for Exploratory Data Analysis, Vol. 249 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2009, pp. 85–108.

[52] M. Mihaylov, K. Tuyls, A. Nowé, Decentralized Learning in Wireless Sensor Networks, Lecture Notes in Computer Science (Springer

Berlin/Heidelberg) 5924 (2010) 60–73.
[53] P. Wang, T. Wang, Adaptive routing for sensor networks using reinforcement learning, in: Computer and Information Technology, 2006. CIT'06. The Sixth IEEE International Conference on, IEEE, 2006, pp. 219–219.
[54] G. D. Fatta, F. Blasa, S. Cafiero, G. Fortino, Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks , Journal of Parallel and Distributed Computing 73 (3) (2013) 317 – 329, models and Algorithms for High-Performance Distributed Data Mining.
[55] P. Sasikumar, S. Khara, K-means clustering in wireless sensor networks, in: Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on, 2012, pp. 140–144.
[56] B. Brewer, libfixmath - cross platform fixed point maths library (2012). URL `http://code.google.com/p/libfixmath/`
[57] P. Aimonen, libfixmatrix - c library for fixed point matrix, quaternion and vector calculations (2012). URL `https://github.com/PetteriAimonen/libfixmatrix`
[58] G. E. Bottomley, A novel approach for stabilizing recursive least squares filters, IEEE Transactions on Signal Processing 39.
[59] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems 2 (4) (1989) 303–314.
[60] R. Hecht-Nielsen, Theory of the backpropagation neural network, in: Neural Networks, 1989. IJCNN., International Joint Conference on, IEEE, 1989, pp. 593–605.
[61] Y.-H. Pao, G.-H. Park, D. J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.
[62] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, Vol. 2, IEEE, 2004, pp. 985–990.
[63] S. Ding, H. Zhao, Y. Zhang, X. Xu, R. Nie, Extreme learning machine: algorithm, theory and applications, Artificial Intelligence Review (2013) 1–13.
[64] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, Neural Networks, IEEE Transactions on 17 (6) (2006) 1411–1423.
[65] E. Fuchs, T. Gruber, J. Nitschke, B. Sick, Online segmentation of time series based on polynomial least-squares approximations, Pattern Analysis and Machine Intelligence, IEEE Transactions on 32 (12) (2010) 2232–2245.
[66] S. Roberts, Control chart tests based on geometric moving averages, Technometrics 1 (3) (1959) 239–250.
[67] L. K. Hansen, P. Salamon, Neural network ensembles, IEEE transactions on pattern analysis and machine intelligence 12 (1990) 993–1001.
[68] D. Hall, J. Llinas, An introduction to multisensor data fusion, Proceedings of the IEEE 85 (1) (1997) 6–23.
[69] A. Zimek, R. J. Campello, J. Sander, Ensembles for unsupervised outlier detection: challenges and research questions a position paper, ACM SIGKDD Explorations Newsletter 15 (1) (2014) 11–22.
[70] A. Downs, An economic theory of political action in a democracy, The Journal of Political Economy 65 (1957) 135–150. doi:10.1086/257897.
[71] R. A. Fisher, Statistical methods for research workers, Genesis Publishing Pvt Ltd, 1925.
[72] K. Ni, N. Ramanathan, M. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, M. Srivastava, Sensor network data fault types, ACM Transactions on Sensor Networks 5 (3) (2009) 25.
[73] A. Sharma, L. Golubchik, R. Govindan, On the prevalence of sensor faults in real-world deployments, in: Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on, IEEE, 2007, pp. 213–222.
[74] A. B. Sharma, L. Golubchik, R. Govindan, Sensor faults: Detection methods and prevalence in real-world datasets, ACM Transactions on Sensor Networks 6 (3).
[75] Sensorscope project (2014). URL `http://lcav.epfl.ch/op/edit/sensorscope-en`
[76] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, R. Thibaux, J. Polastre, R. Szewczyk, The intel lab, berkely dataset (2004). URL `http://db.csail.mit.edu/labdata/labdata.html`
[77] J. Eriksson, A. Dunkels, N. Finne, F. sterlind, T. Voigt, Mspsim – an extensible simulator for msp430-equipped sensor boards, in: Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, 2007, p. 27.