NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

PEDRO MIGUEL PEREIRA MATIAS

Master in Electrical and Computer Engineering

OUTLIER AND ATTACKER RESILIENT METHODS BASED ON RATING AND REPUTATION SYSTEM

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

OUTLIER AND ATTACKER RESILIENT METHODS BASED ON RATING AND REPUTATION SYSTEM

PEDRO MIGUEL PEREIRA MATIAS

Master in Electrical and Computer Engineering

Adviser: Daniel de Matos Silvestre
Assistant Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2023

Outlier and Attacker Resilient Methods based on Rating and Reputation System

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.7.0) [1].

To you, who watch over my heart.

Acknowledgements

I begin by thanking my advisor, Professor Daniel Silvestre, for the opportunity he has given me as well as his patience and support throughout this project. To all faculty from FCT and in particular to the members of the Department of Electrical Engineering. I am thankful to our great university and all of its people, who work every day to ensure the future of our younger generations.

I would also like to leave a word of appreciation toward all the people I have met while attending this institution. To all my friends, both old and new, to all my classmates and fellow students who have been there not just for me but for us all since the beginning. To my family, who have supported my dreams and cared for me with all their love and patience; who've pushed me forward and helped me at all moments.

Finally, I would like to leave a massive thank you to my girlfriend, without whom I would not be the person I am today. She has inspired me like no other and helped me in every manner humanly possible, and I am deeply thankful for all she's done in that regard. Thank you, my love.

Thank you, all.

This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UIDB/50009/2020, through project PCIF/MPG/0156/2019 FirePuma and through COPELABS, University Lusófona project UIDB/04111/2020.

"I have no country to fight for; my country is the earth, and I am a citizen of the world." (Eugene V. Debs)

Abstract

A key component in an automatic surveillance system that can receive crowd-sourced data, such as an early forest fire detection system, must consider the possibility of corrupted data and also attacks on the processors in the network running the estimation task. In both cases, there is the need to introduce some process to decide when to remove a specific value from the computations. In this thesis, we study using reputation and rating metrics to construct an algorithm that is resilient to erroneous data and attacks in linear dynamical systems and compare it against traditional methods to remove outliers. It is shown in simulation that the presented methods have performance comparing or surpassing the traditional methods, which is an interesting outcome that reinforces the importance of the literature on rating and reputation use for resilient consensus and distributed optimization.

Keywords: Outlier Detection, Isolation Forest, Local Factor Outlier, One Class Support Vector Machines, Minimum Covariance Determinant, Rating and Reputation, Multi-Agent Systems, Fault-tolerant, Estimation, Fault accommodation, unknown dynamics, Attack resilient.

Resumo

Um componente-chave num sistema de vigilância automática que pode receber dados de crowdsourcing, como um sistema de detecção preventiva de incêndios florestais, deve considerar a possibilidade de existirem dados corrompidos e também ataques aos processadores na rede que executam a tarefa de estimação. Em ambos os casos, há a necessidade de introduzir algum processo para decidir quando remover um valor específico aos cálculos. Nesta tese, estudamos o uso de métricas de reputação e classificação para construir um algoritmo resiliente a dados erróneos e ataques em sistemas dinâmicos lineares e comparamos com métodos tradicionais de remoção de valores atípicos. É mostrado em simulação, que os métodos apresentados têm desempenho semelhante ou superior ao dos métodos tradicionais, o que é um resultado interessante que reforça a importância da literatura sobre uso de rating e reputação para consenso resiliente e otimização distribuída.

Palavras-chave: Detecção de Erros, Isolation Forest, Local Factor Outlier, One Class Support Vector Machines, Minimum Covariance Determinant, Avaliação e Reputação, Sistemas Multi-agente, tolerância a falhas, estimação, acomodação de falhas, dinâmica desconhecida, resistente a ataques.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

In the field of fire surveillance, crowd-sourced data and citizen science can be valuable sources of information to complement traditional fire detection methods [2–5]. Crowd-sensing systems [6] may include the use of a dedicated application in the hands of citizens [7, 8], social media monitoring [9], or reports from citizens through a dedicated phone line. Further, these methods can be equated to a sensor network where each node moves dynamically within the surveyed space.

However, the quality of crowd-sourced data is often questionable due to a lack of standardization and control over the data collection process [10, 11], drawing a similarity to a dynamic system with unknown dynamics and subject to outlier data, which can lead to inaccurate results and faulty predictions. Here, outlier detection and removal techniques can be used to aid in the identification and elimination of unreliable data points, improving the accuracy and effectiveness of the fire detection system.

Looking towards the sensor network as a whole, it can be seen as a multi-agent system [12, 13] with a distributed network, where each agent can be a user of a mobile app, or a set of sensors working in a closed network and sending information to a larger network that also encompassed the mobile users. Within the same context there is no assurance that all agents in the network will be doing their best to help in the detection of a fire. The sensors employed may suffer a malfunction or suffer noise due to unknown factors, or mobile users can act as rogue agents and intentionally inject false data into the network, compromising the integrity of the data and undermining the performance of the system, thus methods that allow their identification and elimination is a must, ensuring the accuracy and reliability of the estimation performed by the network.

In this thesis, we first look into the data received by a single sensor through the scope of several outlier detection methods as well as scoring methods to understand which are more fit to increase the accuracy of the data collected. Then, using the methods with the better performance, we take a step further and verify their reliability within a multi-agent system, with the goal of increasing the resilience of the network and elimination of rogue agents.

2

Outlier Removal in Measurement Data

2.1 Motivation

Methods to detect the presence of outlier values are well-known and disseminated throughout the literature on the subject of automated classification. These algorithms are useful to identify values that do not conform with the model in some sense, depending on the given proposal. However, recent developments have focused on viewing an equivalent process in the realm of dynamical systems as a way to enforce resilience against both faults and bad sensor data.

In the industry, dynamical models have also played an important role in various applications such as consensus [14], optimization [15], motion coordination tasks such as flocking or leader following [16], rendezvous problems, computer network resource allocation, algorithms to classify the relative importance of webpages (i.e., PageRank [17]), clock synchronization, desynchronization at the Medium Access Control layer [18], maintaining formations, among others.

There are several known methods across literature [19–26] to deal with this problem when the dynamic model of the system is known. However, this hardly applies when the model is unknown and past data is not available. In such scenario, it might be preferable to employ distance metrics between the received values to discard bad data. With Machine Learning, that can be accomplished via K-Nearest Neighbour (KNN) [27] algorithms. Another method could be to discard the μ most extreme values [28], or by defining scoring mechanisms [29].

2.2 Problem Statement

The problem of selecting the more accurate measurements provided by sensors or human contributors with respect to the underlying state of an unknown dynamical system is difficult to classify for the lack of available information.

In this chapter, we consider a dynamical system with state $x \in \mathbb{R}^n$ and external input $u \in \mathbb{R}^p$ that can, for instance, used to model a wildfire. In such case, x accounts for the state variables such as position and velocity whereas u accounts for external signals such as wind, humidity, available fuel, etc, while the dynamics function g describes the behaviour of the fire. Formally, this system is assumed to be described by the continuous-time differential state equation:

$$\dot{x} = g(x, u). \tag{2.1}$$

Depending on the approach and application, g may not be possible to describe mathematically, be it due to the complex dynamics of the system or other associated complications. Incidentally, a collection of sensors or human actors providing measurements of a portion of the system state can be modelled by the output equation in discrete time:

$$y(k) = C(k) \cdot x(k) + \eta(k) \tag{2.2}$$

where, for simplicity of notation, the matrix $C(k) \in \mathbb{R}^{md \times n}$ models how all possible measurements m of size d can be taken regarding the state x and the noise signal $\eta(k) \in \mathbb{R}^{md}$, which is unknown and acts on all measurements for which no prior stochastic information or bounds are known.

Since not all measurements are known for all time instants k, some rows in $C(k)$ and $\eta(k)$ can be set to a null value. It should also be noted that even though the state equation in (2.1) is written in continuous time, measurements are received in discrete-time slots.

To further simplify, after removing null measurements, $\mathcal{M}(k)$ is defined as a set of all received data at time k such that $\mathcal{M}(k) = \{y(k) : y(k) \neq 0\}$.

2.3 Automatic Outlier Detection Methods

Outliers are observations that deviate so much from the remaining set suspected of being generated by a different method [30]. Within the field of data analysis, methods to correctly identify outliers are quite relevant to improve whatever values are being computed with a given data sample.

In this paper, the purpose is to compare the performance of four Automatic Outlier Detection Methods (AODM) against those inspired by Rating and Reputation system often seen on movie or bookstore websites. Four AODMs were chosen for this task due to their popularity and widespread use: Isolation Forest (iForest), Local Outlier Factor (LOF), One Class Support Vector Machines (OCSVM) and Minimum Covariance Determinant (MCD).

2.3.1 Isolation Forest

The Isolation Forest (iForest) algorithm functions by averaging the path length to all external nodes of a collection of Isolation Trees (iTrees) which recursively isolate observations in T nodes through a random feature q and a split value p, such that $q < p$,

creating branches dividing objects into T_l and T_r [31]. The number of branches created is the path length $h(y)$ from the root to its external node.

One of the great advantages of this algorithm is its ability to score outlier points with only a partial model since a large part of each tree is not needed for anomaly detection (only up to the desired path length). It can also exploit sub-sampling techniques to avoid problems such as *swamping* and *masking*.

Definition 2.3.1 (Swamping)**.** When normal instances are too close to anomalies, the number of partitions required to isolate them increases, which makes it harder to distinguish anomalies from normal instances. [31]

Definition 2.3.2 (Masking)**.** When an anomaly cluster is large and dense, it increases the number of partitions to isolate an anomaly, leading to the classification of anomalies as normal data. [31]

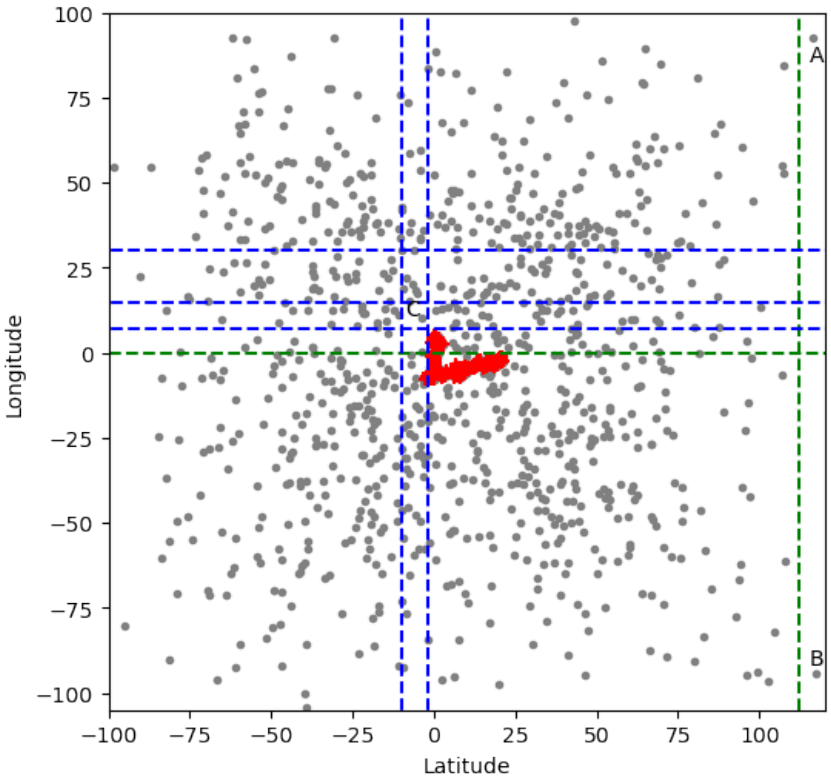


Figure 2.1: In dashed green lines, are shown the branches that isolate points A and B. In this case, since they are on the outermost edges of the distribution, only 2 iterations of the algorithm are needed. In dashed blue lines, are shown the successive branches for 5 iterations to isolate point C.

To discover anomalous data, a method for scoring all data is required. Deriving such a score is difficult as $h(y)$ cannot be normalized due to the unbounded nature of each iTree's height, which grows linearly with the number of instances n in a sample, or its average height, which grows by $\log(n)$ [31].

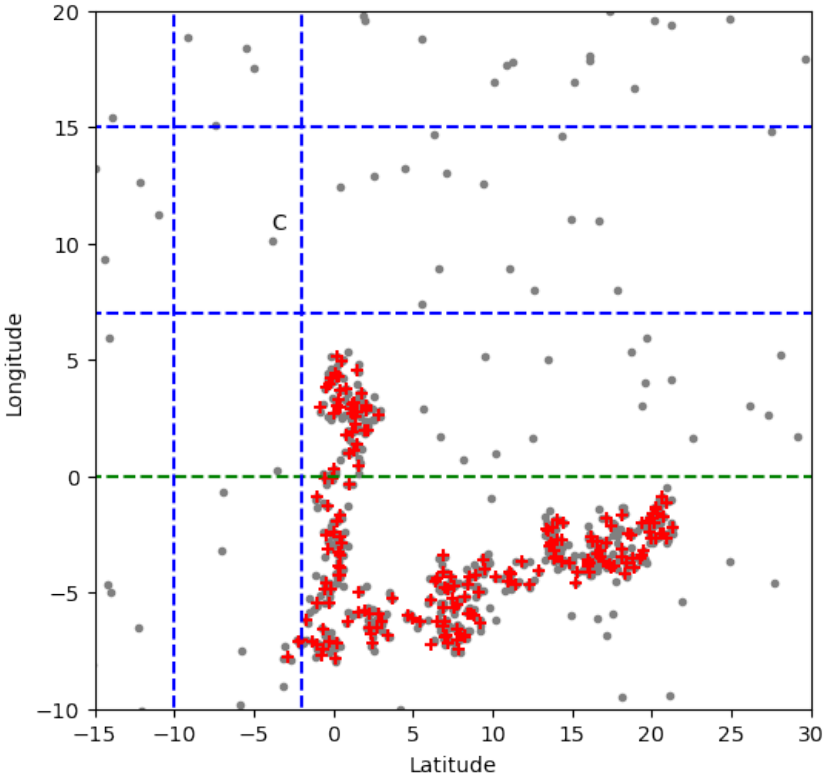


Figure 2.2: A close-up of the branching process to isolate C. Because this point is much closer to the targets (red), it takes more iterations to isolate it.

The path taken to isolate the feature creates the decision function. The length of a branch conveys the degree of belonging of an attribute. As such, it is possible to discern that, when the trees consistently produce shorter branches for an attribute, it is more likely that the attribute is an anomaly.

Figures 2.1 and 2.2 represent randomly generated points simulating the contributors' use of an app that uses GPS locations to pinpoint their proximity (grey) to a developing fire (red); with dashed lines denoting the branching required to isolate objects A, B and C.

2.3.2 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm was introduced by Breunig, Kriegel, Ng and Sander [32] as a non-binary density-based classification method to detect outliers in a data sample. Unlike other methods, LOF first verifies the density of an object's neighbourhood (N) through a reachability distance function (2.3) of an object a with respect to an object b:

$$\text{reach-dist}(a,b) = \max(\text{k-dist(b)}, \text{dist(a,b)}). \tag{2.3}$$

Here, k is the number of nearest neighbours being taken into account, k-dist(b) is the distance between point b and its k^{th}-nearest neighbour; and dist(a,b) is the distance

between points a and b, in a straight line. The reachability distance will then be the largest of either distance. In Figure 2.3, distances D1 and D2 are depicted: i) for objects a and b, reach-dist(a,b) = dist$(a,b) = D_2$; in case of $y_i \in \{c,d,e,f\}$ their respective reach-dist(y_i,b) = k-dist$(b) = D_1$.

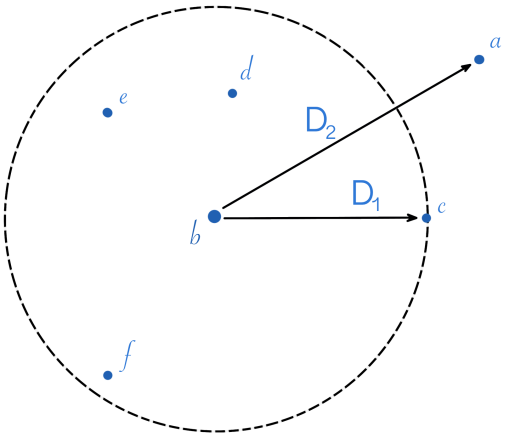


Figure 2.3: D1 represents the k-dist(b) for k = 5 and D2 represents dist(a,b). [33]

In density-based algorithms, two parameters are required to determine a cluster's density threshold [32]: (i) k_{min}, minimizing the number of objects required to declare a cluster; and (ii) a parameter specifying the volume of the cluster. Objects or regions whose neighbourhoods respect these constraints are connected, forming clusters.

Another requirement [32] to the detection of outliers in these conditions is a method to dynamically compare the density of different clusters of objects. For that, k_{min} is fixed as a parameter, and reach-dist(a, b) $\forall\, b \in N_{k_{min}}(a)$ as a measure of the volume to determine the density of an object a's neighbourhood. This method is denoted the Local Reachability Density (lrd) of the object a and is given by:

$$\text{lrd}(a) = \frac{|N_{k_{min}}(a)|}{\sum_{b \in N_{k_{min}(a)}} \text{reach-dist}_{k_{min}}(a,b)} \tag{2.4}$$

Formally, lrd(a) is denoted as the inverse of the average reachability distance of a based on its k_{min}-nearest neighbours. It is also important to note that, by (2.4), it is possible for lrd$(a) \to \infty$ if the sum of its reachability distances for all $b \in N_{k_{min}}(a)$ tends to 0. However, this case can be guarded against by forcing $b \neq a \;\forall\, b \in N_{k_{min}}(a)$.

The object's LOF, also referred to as the degree of outlyingness, is thus denoted as the average of the ratio of lrd(a) with lrd(b) $\forall b \in N_{k_{min}}(a)$:

$$\text{LOF}_{\text{k}_{\text{min}}}(a) = \frac{\sum_b \frac{lrd_{k_{min}}(a)}{lrd_{k_{min}}(b)}}{|N_{k_{min}}(a)|} \tag{2.5}$$

From (2.5) it is intuitive [32] how the algorithm will rank an object a with respect to its neighbours: lower $lrd(a)$ and large neighbours' local reachability distances mean high

LOF(a). This means that the LOF of isolated objects will be much higher than 1, while data points within a cluster will have values close to 1.

This method has a great advantage when classifying samples with many clusters of different densities because it can verify that, even if an object does not belong to one cluster, it may still belong to another without it becoming an outlier. Taking the initial example of a forest fire having been located by several different sources, this could be useful not only in cleaning the data received about their locations but to gauge if there is enough data to suspect more than one focus point for the fire.

2.3.3 Support Vector Machines

The Support Vector Machine (SVM) algorithm is one focused on problems where linear functions are not enough to separate between classes. SVM takes an approach to elevate objects to a higher dimension, a feature space $\mathbb{H}$, where points can be separated by a linear function or a planar surface.

Using the measurement set $\mathcal{M}(k)$ defined in Section 2.2, let $\mathcal{M}_{SVM} = \{(y_i, s_i)\} \forall\ y_i \in \mathcal{M}(k) \wedge s_i \in \{-1, 1\}$ be a data sample where y_i are input values and s_i are their respective output class. Figure 2.4 shows a representation of this with $y_i \in \mathbb{R}^2$ and the Class represented as Red for $s_i = 1$ and Blue for $s_i = -1$.

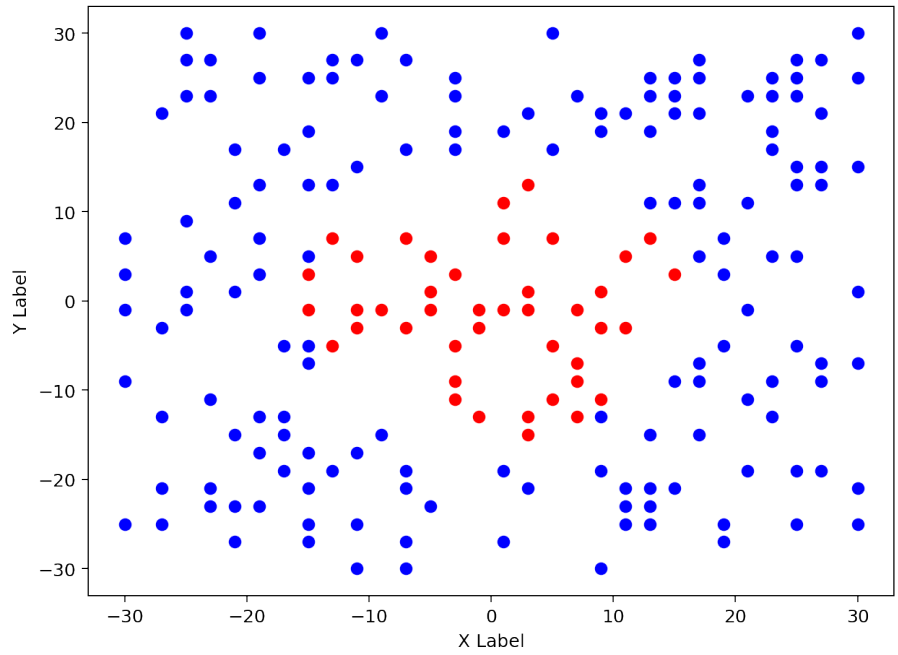


Figure 2.4: Distribution of points from $\mathcal{M}_{SVM}$ in a Cartesian graphic. Class 1 ($s_i = 1$) can be visualized in Red, while Class 2 ($s_i = -1$) can be visualized in Blue.

It is not possible to pass a linear hyperplane through the points that would separate the 2 Classes. For this example in particular, SVM adds a 3^{rd} dimension to the plot and elevates all points along a feature space, $\mathbb{H}$, as shown in Figure 2.5, on the right.

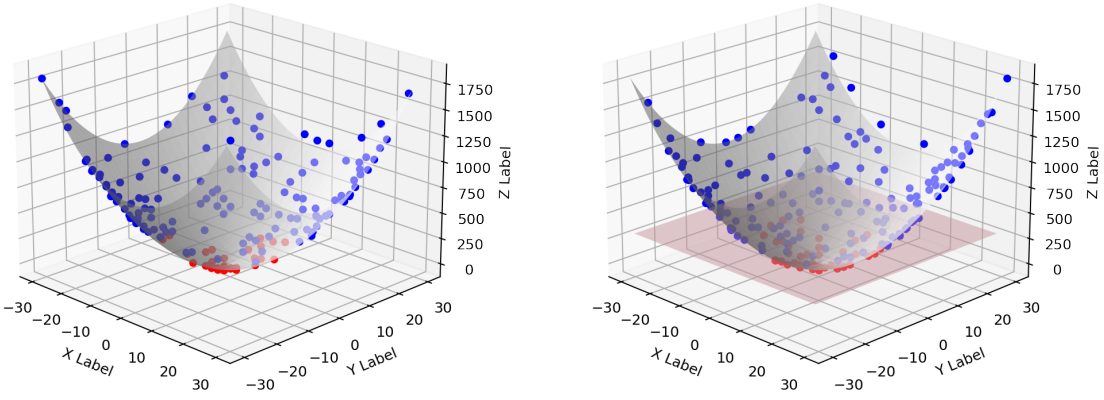


Figure 2.5: On the left: The feature space $\mathbb{H}$ (grey) is created and the points are elevated into its surface. On the right: The intersecting plane Φ separates points from class 1 and class 2, according to the minimization function in (2.7).

The separation margin is then calculated by minimizing the distance between the closest points of each class, generating a plane Φ intersecting the feature space, shown in Figure 2.5 on the left, which can be described by (2.6), below:

$$\omega^{\mathsf{T}} \cdot y + b = 0 \tag{2.6}$$

Where $\omega \in \mathbb{H}$ and $b \in \mathbb{R}$. The plane intersection is then projected back to the original space, resulting in a non-linear margin surrounding the elements of Class 1, as shown in Figure 2.6.

To prevent inadequacies with objects over Φ's separation margins, a set of slack variables ζ_i is introduced. This creates a soft margin, which allows for an admissible training error. So that this error does not create a bias in the model, the number of objects within the soft margin has to be bounded, thus a control variable $C > 0$ is set. Its purpose is to limit the number of accepted objects within the soft margin.

The minimization problem is described as follows:

$$\min_{\omega,b,\zeta_i} \frac{\|\omega\|^2}{2} + C \sum_{i=1}^{n} \zeta_i \tag{2.7}$$

subject to the constraints:

- $y_i(\Phi(y_i) + b) \geq 1 - \zeta_i \; \forall i = 1, 2, 3, ..., n$
- $\zeta_i \geq 0 \; \forall i = 1, 2, 3, ..., n$

Resorting to quadratic programming and Lagrange Multipliers, the decision function $f(y)$ is then obtained as:

$$f(y) = \operatorname{sgn}\left(\sum_{i=1}^{n} \lambda_i s_i K(y, y_i) + b \right) \tag{2.8}$$

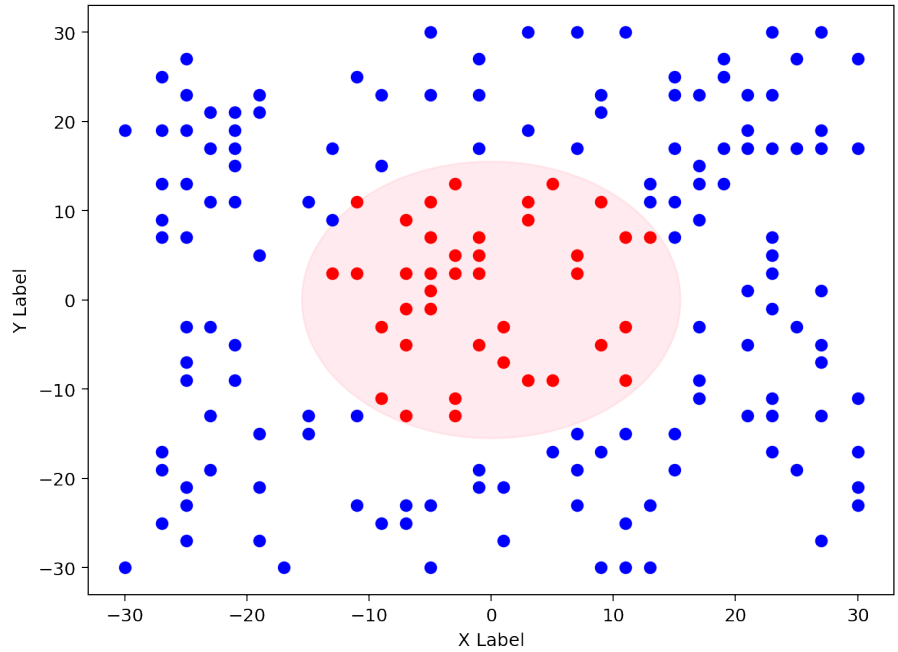


Figure 2.6: The intersection of Φ with $\mathbb{H}$, in pink, separates Class 1 (red) from Class 2 (blue).

where, λ_i are the Lagrange multipliers. Since support vectors will be sparse, there will be relatively few Lagrange multipliers with non-zero values. The term $K(y, y_i)$ is known as the *Kernel Function*, and is given by

$$K(y, y_i) = \Phi(y)^{\intercal}\Phi(y_i) \tag{2.9}$$

This function replaces the need to perform the explicit projection to the feature space $\mathbb{H}$, as shown in Figure 2.5. This kernel trick allows a faster solution of non-linear separable objects.

There are a few variants of this algorithm in the literature. One such is Scholköpf's [33] approach, which turns the SVM into a one-class classifier. Meaning that, through this approach, information can be inferred about a model while knowing only one classification state. This method is commonly referred to as the One-Class Support Vector Machine (OCSVM). This approach excels in the detection of outlier points, which is the focus of this chapter.

With OCSVM, objects are separated from the origin through a feature space akin to the previous approach, maximizing the distance from the hyperplane to the origin. The result is a binary function (2.11) which captures regions in the input space where the probability density of the data lives and returns, as output, +1 in a «small» region and -1 elsewhere. The quadratic programming minimization function differs from (2.7) as follows:

$$\min_{\vec{\omega},\zeta_i,\rho}\left(\frac{\|\omega\|^2}{2} + \frac{1}{n\nu}\sum_{i=1}^{n}(\zeta_i - \rho)\right) \tag{2.10}$$

OCSVM's decision function differs from normal SVM with the introduction of ν that replaces C and is referred to as the smoothness constant. Here, ν sets an upper bound on the fraction of outliers and serves as a lower boundary on the number of training examples required for Support Vectors. Using Lagrange multipliers and a kernel function or the dot-product calculations, the decision function then becomes:

$$\begin{aligned} f(y) =& \text{sgn}((\omega\Phi(y_i)) - \rho) \Leftrightarrow \\ f(y) =& \text{sgn}\left(\sum_{i=1}^{n}(\lambda_i K(y, y_i) - \rho)\right) \end{aligned} \tag{2.11}$$

The method creates a hyper-plane, described by ω and ρ with maximized distance from the origin in the feature space $\mathbb{H}$, separating all objects from the origin.

2.3.4 Minimum Covariance Determinant

The Minimum Covariance Determinant's (MCD) objective is to find a number o of observations, in the universe of $\sigma = |\mathscr{M}(k)|$, whose classical covariance matrix minimizes its determinant [34]. Toward that goal, a collection of subsets $\mathscr{M}_i$ are randomly generated with equal size and without repetition. Meaning that, in a data sample with $\mathscr{M}(k) = \{y_1, ..., y_\sigma\}$ as inputs and $y_i \in \mathbb{R}^d$, each subset $\mathscr{M}_{1:\sigma}$ contains elements such that $\mathscr{M}_1 \cup \mathscr{M}_2 \cup ... \cup \mathscr{M}_n = \mathscr{M}(k)$. Subset $\mathscr{M}_1$ is characterized by its arithmetic average (2.12) and its classical covariance matrix (2.13).

$$T_1 = \frac{1}{\sigma}\sum_{y_i \in \mathscr{M}_1} y_i \tag{2.12}$$

$$S_1 = \frac{1}{\sigma}\sum_{y_i \in \mathscr{M}_1} (y_i - T_1)(y_i - T_1)^\intercal \tag{2.13}$$

Then, each point's relative distance can be yielded through Mahalanobis Distances $d_1(\text{i})$ (2.14) and $\mathscr{M}'_1$ is formed by sorting d_1 through a π-permutation such that $d_1(\pi(1)) \leq d_1(\pi(2)) \leq ... \leq d_1(\pi(n))$ and $\mathscr{M}'_1 = \{\pi(1), ..., \pi(n)\}$.

$$\begin{aligned} d_1(i) =& \sqrt{(y_i - T_1)^\intercal S_1^{-1}(x_i - T_1)} \\ & \forall i = 1, ..., n \end{aligned} \tag{2.14}$$

If $\det(S_1) > 0$, applying equations 2.12, 2.13 and 2.14 in a sequence yields (T'_1, S'_1) with $\det(S'_1) \leq \det(S_1)$. This sequence is denominated a Concentration Step (C-step), as it concentrates on the o observations with lowest distances and S'_1 is more concentrated than S_1. This process is repeated until a ceiling, usually 10 iterations[34], is achieved or $\det(S_1^{(n)}) = 0$ or $\det(S_1^{(n)}) = \det(S_1^{(n-1)})$.

Because the sequence of $\det(S_1) \geq ... \geq \det(S_1^{(n)})$ is a monotonic non-negative sequence [34], it becomes intuitive that the algorithm will converge. This is a necessary condition, albeit not sufficient to determine $\det(S_1^{(n)})$ as a global minimum for the MCD.

After convergence has been achieved throughout all subsets $\mathcal{M}$, the arithmetic average T_{MCD} and the classical covariance matrix S_{MCD} are obtained through (2.15) and (2.16), respectively, where θ_i is constrained by (2.17).

$$T_{MCD} = \frac{\sum_{i=1}^{n} \theta_i y_i}{\sum_{i=1}^{n} \theta} \tag{2.15}$$

$$S_{MCD} = \frac{\sum_{i=1}^{n} (y_i - T_{\text{MCD}})(y_i - T_{\text{MCD}})^{\intercal}}{\sum_{i=1}^{n} \theta_i - 1} \tag{2.16}$$

$$\theta_i = \begin{cases} 1 & \forall\, d_{(T_{\text{MCD}}, S_{\text{MCD}})}(i) \leq \sqrt{\chi^2_{p,.975}} \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

Computing (2.15) and (2.16) through the Mahalanobis Distance (2.14), an elliptical separation margin is obtained, with normal objects enclosed inside it and outlying objects left on the outside.

2.4 Rating and Reputation

Rating algorithms [10] are most prevalent in situations where there is little information to start classifying data or its validity may be dependent on the most recently available data.

The rating function uses the euclidean distance (2.18). Data is provided through a set Λ containing the previously best-rated objects or, during a first iteration, all known objects. This subset can also be called a sliding window when dealing with streamed data and has a maximum size, defined by the user and is the minimum known information required for a rating to be performed.

$$\text{rating}(p, \Lambda) = \sum_{v_i \in \Lambda} \|p - v_i\|_2^2 \tag{2.18}$$

The input p is the object being rated and, since the rating function is a sum of distances, higher ratings will point to less reliable data, thus the better objects will be close to their minimum values.

Reputation (2.19) serves the purpose of providing extra information in the shape of weight to the rating system. This means that a sensor is considered reliable when the rating score of its objects is consistently low.

$$r_j = 1 - \frac{d_j}{s_j} \tag{2.19}$$

Here d_j is the number of discarded messages by sensor j and s_j is the total number of messages sent by sensor j.

$$\text{rating}(p, \Lambda, r) = \sum_{v_{ij} \in \Lambda} r_j \|p - v_{ij}\|_2^2 \tag{2.20}$$

The Rating and Reputation score is thus given by (2.20). Note that v_{ij} refers to object v with id i provided by sensor j and r is the collection of reputation scores for all sensors in the system.

2.5 Method comparisons

The objective of this section is to outline the steps performed to produce the comparison among all five methods surveyed in Section 2.3. For that purpose, a situation was simulated where ten sensors, each with its own associated error $\eta_j \in \mathrm{R}$ such that $\eta_j = \{0.01, 0.173, 0.366, 0.5, 45, 56, 67, 78, 89, 100\}$. Each sensor sends a measurement related to the state $x(k) \in \mathrm{R}^2$ in time instant k. The objective is to understand the capabilities of each ODM in a situation where information arrives in a live stream. The received data in discrete-time slots are stored in matrix $M(k)$ that denotes the collection of objects received from all sensors at time-slot k. A history matrix Ψ was used to keep a record of all good objects received such that $\Psi(k) = \Psi(k-1) \cup M'(k)$, where $M'(k)$ are the objects considered normal data for time-slot k. Algorithm 1 describes a first attempt at accomplishing this.

Algorithm 1 Object Estimation for Simulation 1

```
Initialize Ψ(0) = ∅                                   ▷ History matrix
for k > 0 do
    Receive M(k)                                      ▷ Sensor Data
    if Ψ(k − 1) ≠ ∅ then
        M'(k) = Classify(Ψ(k − 1), M(k))
        Ψ(k) = Ψ(k − 1) ∪ M'(k)
    else
        Ψ(k) = Ψ(k − 1) ∪ M(k)
    end if
    x̃(k) = mean(x_i ∈ Ψ(k))                           ▷ Compute estimate
end for
```

The Classify function implements the classification model and prediction for the ODM being evaluated. Using the l^2-norm - $\|x_i - \tilde{x}(k)\|_2 \; \forall \, y_i \in \Psi(k)$ - as a distance metric, a comparative analysis of results was performed. The four ODM perform similarly as seen in Figure 2.7, but cannot reach the performance of the Rating and Reputation algorithms. This is due to the influence that old objects in Ψ have in the classification and prediction calculations.

It should be noted that no guards were imposed for a minimum number of objects required for classification to start or to control their arrival. As such, if in a given moment k, no new objects received $M(k)$ are accepted, only old objects are being counted in the current estimation. This may have been the main reason for the weak performance, with the OCSVM and LOF models having the worst mean error which was 4.402 and 4.204,

respectively. In comparison, the Reputation model achieved an error of 0.979 in the same interval.

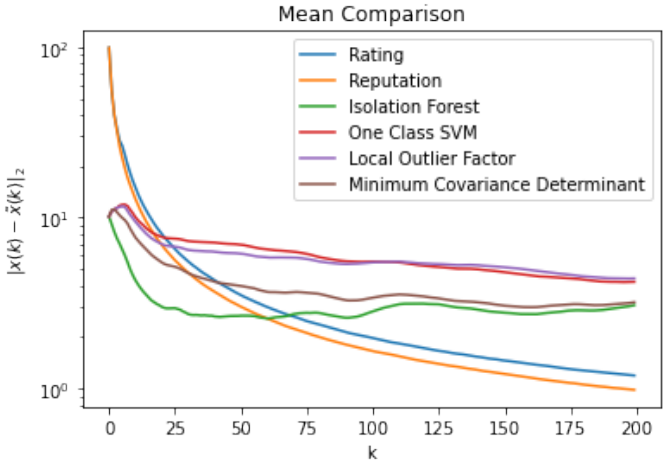


Figure 2.7: Mean estimation error comparison with logarithmic y-scale.

In order to account for the time-varying behaviour, Algorithm 1 was changed to take into account only accepted points at the current time instant for the estimation. Additionally, if no points were deemed normal, the previous state estimation is used to avoid large errors from the absence of points. These changes are documented on Algorithm 2.

Algorithm 2 Object Estimation for Simulation 2

Initialize $\Psi(0) = \emptyset$ ▷ History matrix
for $k > 0$ **do**
  Receive $M(k)$ ▷ Sensor Data
  if $\Psi(k-1) \neq \emptyset$ **then**
    $M'(k) = Classify(\Psi(k-1), M(k))$
    $\Psi(k) = \Psi(k-1) \cup M'(k)$
  else
    $\Psi(k) = \Psi(k-1) \cup M(k)$
  end if
  if $M'(k) = \emptyset$ **then**:
    $\tilde{x}(k) = \tilde{x}(k-1)$
  else
    $\tilde{x}(k) = mean(y_i \in M'(k))$ ▷ Compute estimate
  end if
end for

The second simulation results are presented in Figure 2.8, showing a much more competitive error between the ODM models and the Rating and Reputation algorithms.

The highest mean error for this simulation was 24.171, achieved by the LOF; while the lowest was 0.740, achieved by the Reputation model (Table 2.1). Figure 2.8 shows interesting results in this regard, as it is possible to remark that the OCSVM and iForest methods showed better results than the Rating and Reputation models in earlier iterations. The

reasons for the loss of accuracy across all ODM except the MCD, which performs coherently across time, was found to be due to the guard restrictions imposed and overfitting. This is especially prominent with OCSVM and iForest.

OCSVM decreases rapidly in early instants with the mean error then increasing quickly because the value for $\tilde{x}(k)$ used stagnated without new information from $M'(k)$, thus the estimation becomes outdated.

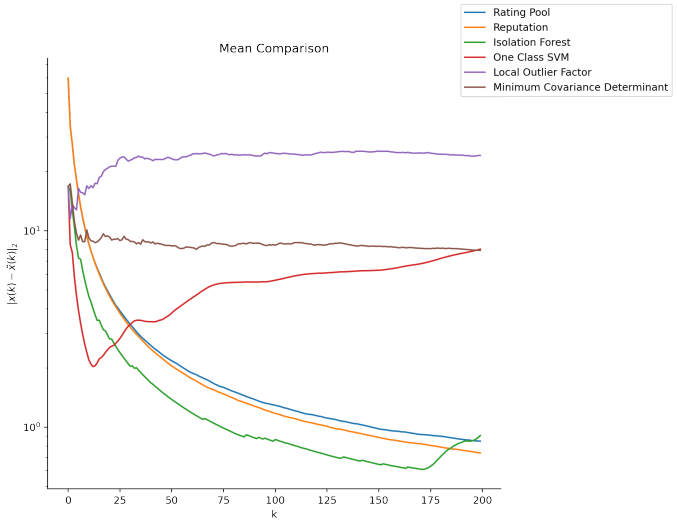


Figure 2.8: Mean estimation error comparison with logarithmic y-scaling for the 2^{nd} simulation

iForest suffers of the same problem, although its manifestation occurs much later. Prior to the stagnation, iForest had been outperforming every other model but, as soon as its estimation begins to lose relevance, its estimation error increases.

Method	Edge Mean Error Value
Reputation	**0.740402**
Rating Pool	0.849171
iForest	0.907189
MCD	7.948775
OC-SVM	8.063033
LOF	24.171177

Table 2.1: Comparison of the mean estimated error for the methods, taken during the 2^{nd} simulation.

The mean error behaviour for the ODM methods in the second simulation occurs because points in initial time steps become irrelevant as time progresses. The Rating and Reputation methods do not hold old information as a metric to inform the present as there is no need for a training set and instead use a Sliding Window to keep its estimation timely. This simulation also pinpoints a bias that happens as, at some point, the ODM methods stop accepting new points due to overfitting. As a consequence, the mean estimation error keeps increasing.

To counter this, the ODM estimators' *score_sample* function, described in the scikit-learn python package [35], was used. This function scores each object evaluated by the classifier and returns an array of scores. Outlier objects always score negative values, while normal objects have positive scores.

It is important to remark this leaves ODMs a step behind the Rating and Reputation algorithms because, in order to always receive new objects with each iteration, they are forced to accept some classified as outliers whenever there are no new normal objects in the model. This solution showed results similar to the initial simulation.

Algorithm 3 Object Estimation for Simulation 3

$\Lambda(0) = \emptyset$ ▷ Initialize Sliding window
for $k > 0$ **do**
  Receive $M(k)$ ▷ Sensor Data
  $\Lambda(k) = \text{update}(\Lambda(k-1), M(k))$
  if $|\Lambda(k)| > \psi$ **then**
    $M'(k) = Classify(\Lambda(k))$
    $\Lambda(k).discard(M(k) \cap \Lambda(k) \cap \overline{M'(k)})$ ▷ Discard objects with high ratings
    $\tilde{x}(k) = mean(y_i \in \Lambda(k) \cap M(k)$ ▷ Compute estimate
  end if
end for

A third simulation was performed imposing a Sliding Window Λ to replace the global history Ψ described in Algorithms 1 and 2. The size was chosen to allow for a significant number of objects. Effectively, this means each object has a lifespan before it is discarded and replaced by newer, more relevant data points. Thus $|\Lambda| \leq \psi = \frac{\text{ceil}(k)}{10} = 20$.

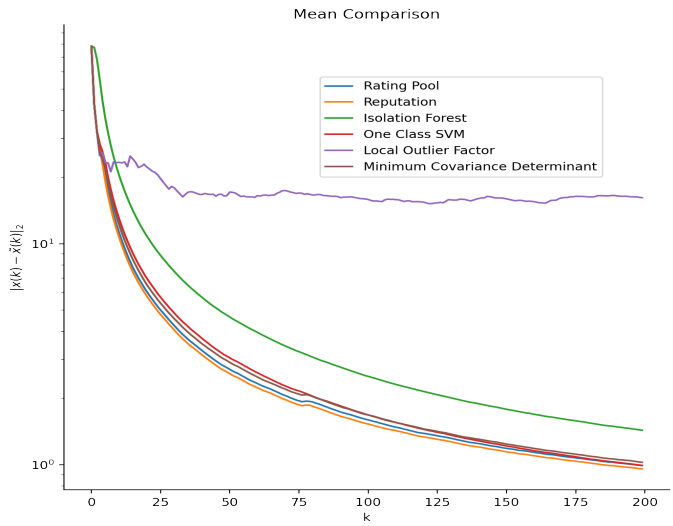


Figure 2.9: A comparison of the ODM with the rating and reputation models, regarding the simulation using a sliding window under a log y-scale.

With Algorithm 3, results become competitive as depicted in Figure 2.9, where the evolution of the mean error is presented. The OCSVM now shows to be approximately

as effective in classifying the objects as the rating method, with the MCD not far behind. The LOF method, however, cannot accompany the other methods. At this point, it can be remarked that the LOF is not suited for this type of classification.

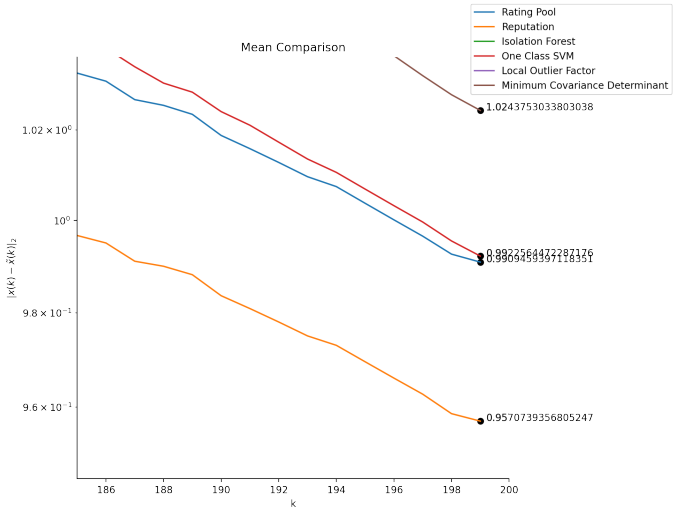


Figure 2.10: The MCD, Rating, Reputation and OCSVM methods produce similar results. A logarithmic scale is used to improve visibility and show that these methods have not yet converged.

A closer look at the last 15 iterations of k (Figure 2.10), shows that the methods produce very similar results. It is intuitive that the Reputation method will outperform the Rating method since it uses the same metric but adds a weight that rewards objects sent by reliable contributors while punishing unreliable sources.

On Table 2.2, a comparison of the best values achieved after 200 iterations of the simulations for the six compared methods is shown. At first inspection, these values could lead to some misconceptions about the results.

Method	A1	A2	A3
iForest	3.067	0.907	1.432
LOF	4.402	24.171	16.157
MCD	3.190	7.949	1.024
OCSVM	4.204	8.063	0.992
Rating	1.188	0.849	0.991
Reputation	**0.979**	**0.740**	**0.957**

Table 2.2: Mean Error Value of the Estimation task by each algorithm developed.

A first conclusion that can be drawn is that the LOF has consistently had the worst performance. This can also be seen in Table 2.3, where its performance is evaluated over 10 simulations using Algorithm 3. This is due to its dependency on density functions and the time relevance of the dynamic system. An object neighbourhood may be densely populated, but if all its neighbours are outdated, then the scoring function will incur in a bias. A possible solution to this could be the inclusion of a time variable in the object

	Method	iForest	LOF	MCD	OCSVM	Rating	Reputation
Simulations	1	1.775	18.049	1.151	**1.047**	1.079	1.074
	2	2.135	16.203	1.199	1.168	1.149	**1.145**
	3	1.028	17.774	0.746	**0.607**	0.758	0.757
	4	1.483	15.609	1.109	**1.023**	1.064	1.062
	5	1.419	13.786	1.243	1.093	**1.082**	1.085
	6	1.592	15.745	1.325	**1.123**	1.242	1.237
	7	1.532	14.063	0.936	1.042	0.843	**0.817**
	8	1.683	16.379	1.197	1.188	1.139	**1.125**
	9	1.551	12.689	**1.029**	1.070	1.072	1.075
	10	1.767	15.839	1.116	1.113	**1.103**	1.110
	Average	1.597	15.614	1.105	**1.048**	1.127	1.049

Table 2.3: Mean Error Value of the Estimation task over 10 simulations of Algorithm 3, with the lowest value for each simulation and the final average in bold text.Values were rounded for visualization purposes. Real values stand within a ±0.5 margin of shown values.

attributes.

Another conclusion is that the Reputation method is constantly outperforming all others (Table 2.2). Its relation to the Rating method has already been stated and, compared to the ODMs, it shows great potential within the scope of this problem. To better study this, Table 2.3 was developed by comparing 10 different simulations using Algorithm 3.

From here, it is possible to verify that, over the 10 simulations, the Reputation method accrued an average mean error of 1.049 ± 0.5. A value superseded only by the OCSVM's own average. The error margin given above is advanced as a measure of the rounding process used. From all simulations, the lowest mean error of the estimations achieved were 0.607 and 0.757, both in simulation 3 and by the OCSVM and Reputation methods, respectively.

3

Multi-Agent Systems

3.1 Motivation

As technologies become increasingly connected through the Internet of Things (IoT), multi-agent systems have become an integral part of the quotidian. These systems rely on cooperation, accuracy and reliability of many devices to achieve their objectives. However, their reliability and accuracy, crucial characteristics for the application of multi-agent systems, can be compromised by the presence of uncooperative agents, which can inject false data and harm the integrity of the system.

In wildfire surveillance, multi-agent systems can help improve systems of early detection, providing early warnings to firefighters and helping in the prevention of loss of life and property, through relying on a network of sensors, citizen science and crowdsensing mechanisms. To prevent the compromise of the data collected through these systems, we explore how methods such as MCD, OCSVM, and rating and reputation systems can be used to improve the reliability and accuracy of the system.

3.2 Problem Formulation

In this chapter, the results obtained in Chapter 2 are transposed to a distributed systems topology, where the goal is to understand whether the methods studied can be used in an environment of multi-agents to create resilient consensus.

To that goal, equation 3.1 is proposed as the node update function, where $x^{(k+1)}$ is the system's next state vector, W is a row stochastic weighted matrix derived from an adjacency matrix A and altered through classification methods studied previously, $x^{(k)}$ is the system's current state vector and $u^{(k)}$ is an external input.

$$x^{(k+1)} = Wx^{(k)} + u^{(k)} \tag{3.1}$$

In this analysis, a set of agents will be denoted as a *network*, where each agent is represented as a *node* in the adjacency matrix $A \in \mathrm{R}^{n \times n}$ where $A_{i,j} = 1$ if agent i communicates

to agent j, and $A_{i,j} = 0$, otherwise. Intuitively, $A_{i,i} = 1 \forall i$, meaning the node is aware of its own value.

Furthermore, since the goal is to reach a consensus, it must be ensured that A is strongly connected: there must be at least one path that links any node to any other node through a chain of communicating nodes. Thus, A can be described in the form of a directed graph $G = (V, E)$ such that $V = \{v1, ..., v_N\}$ is a nonempty finite set of nodes and $E \subseteq V \times V$ is a set of edges, in which an edge is represented by an ordered pair of distinct nodes [36]. Formally, a directed graph $G = (V, E)$ is strongly connected if and only if for any pair of distinct nodes $i, j \in V$, there exists a sequence of nodes such that each pair $(V_j, V_i) \in E$ creates a chained path that passes through all nodes in the network. The agents that can send information to agent i are defined as the set of in-neighbours of agent i, denoted as N_i^{in}. Similarly, the agents that can receive information from agent i are defined as the set of out-neighbours of i, N_i^{out}. Note that, in a directed graph, when $(i, j) \in E$, it is not necessary that $(j, i) \in E$, thus generally $N_i^{in} \neq N_i^{out}$.

The state of each agent i at the k^{th} iteration is represented as $x_i^{(k)}$, where k is the discrete-time index. The initial state of each agent in the network is given by $x_i^{(0)}$. The general state of the network at the k^{th} iteration will be represented as $x^{(k)}$.

The practical value of the consensus value will float due to changes in A performed by the classification algorithm. To perform an accurate comparison of the fidelity of the algorithm, a theoretical consensus value is calculated through equation 3.2.

$$g = q^{\intercal} x^{(0)} \tag{3.2}$$

where g represents the theoretical consensus value, q is the left eigenvector of A associated with the unity eigenvalue and $x^{(0)}$ is the initial state of the agents' set.

In the previous chapters, the results obtained came from using a single agent system that received a set of values and acted upon the information it had and reached decisions on its own. As was observed, of the four ODM explored, only the MCD and OCSVM reached comparable results to the proposed algorithms of rating and rating with reputation (Table 2.3). If the Rating and Reputation algorithms were to be removed from the comparison, the OCSVM algorithm would have achieved the lowest average error in 8 out of the 10 simulations, while the MCD algorithm gathered the remaining 2 lowest average errors.

In the following sections, a new comparison survey is performed, using multi-agent systems [12, 13], or MAS. These systems are composed of several agents able to interact with each other to achieve a given collective goal. In the scope of this project, each agent can classify a series of values corresponding to their position at a given time, compare to the positions of other communicating agents and decide whether they should abide their fellows and move toward them or stay their course. In order of performing this comparison, two functions are used: Weighted Consensus [37, 38] and DEXTRA[39, 40].

3.2.1 Weighted Consensus Method

A simple and fast way to reach consensus in a fully connected network would be to perform an average of all node values. In a strongly connected network, it is best to use a weighted average instead. In this case, A must be transformed to provide a neat description of the influence each node has on its neighbours' values, resulting in a weighted averages matrix W. Take the following adjacency matrix as an example:

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.3}$$

In order to achieve consensus between the values of $x^{(k)}$, W must be row stochastic. Thus it is transformed such that $W_{i,j} > 0$ if $j \in N_i^{in}$ and $W_{i,j} = 0$ otherwise, with the additional constraint that $\sum_{j=1}^{n} W_{i,j} = 1, \forall i$.

The weighted matrix of A, as provided in 3.3 becomes:

$$W_A = \begin{bmatrix} 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0 & 0.25 \\ 0 & 0.333 & 0.333 & 0 & 0.333 \\ 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix} \tag{3.4}$$

And the next state of the nodes is given by:

$$x^{(1)}(i) = W_{i,j} x^{(0)}(j) \tag{3.5}$$

The general case can be written similar to 3.1, with $u^{(k)} = 0, \forall k > 0$:

$$x^{(k+1)} = W^{(k)} x^{(k)} \tag{3.6}$$

where we will have to make W dependent on time to account for the changes in the weights caused by the classification algorithms.

As an example, with the following initial state $x^{(0)} = \begin{bmatrix} 5.99 & 4.49 & 7.61 & 6.49 & 6.19 \end{bmatrix}^T$ and applying equation 3.6 iteratively, the consensus is achieved as shown in Figure 3.1, where each agent is represented as a node and a progression of the sate values across iterations can be verified.

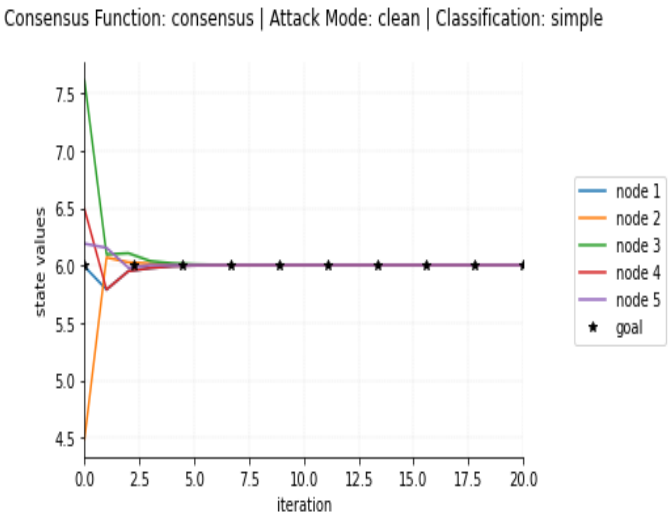


Figure 3.1: Weighted Consensus progression per node.

Using equation 3.2, the theoretical value for the consensus in this example is 6,00179. After 20 iterations of equation 3.6, the value found for each node is

$$x^{(20)} = \begin{bmatrix} 6.00408804 & 6.00408805 & 6.00408806 & 6.00408804 & 6.00408805 \end{bmatrix}^{\mathsf{T}} \tag{3.7}$$

which results in an average error of 2.298×10^{-3}. It can be concluded that, by extending the limit of equation 3.6 to infinity, an accurate prediction of the consensus is produced. A deeper analysis on weighted consensus algorithms can be found on [37, 38].

3.2.2 DEXTRA

DEXTRA stands for Directed Exact first-order Algorithm [40]. It is an improvement over the EXTRA algorithm [41], introduced with the intent of pushing agents to achieve consensus and reach the optimal solution for a composition of private cost functions for the case of directed graphs.

As with the Weighted Consensus, each agent $j \in V$ in DEXTRA keeps the state vector $x_j^{(k)} \in \mathbb{R}^p$. Additionally, a second vector $z_j^{(k)} \in \mathbb{R}^p$ and a scalar value $y_j^{(k)} \in \mathbb{R}$, where k is the discrete-time index, are introduced.

At the k^{th} iteration, agent j weighs its states $W_{i,j}x_j^{(k)}$ and $W_{i,j}y_j^{(k)}$, as well as $\tilde{W}_{i,j}x_j^{(k-1)}$, and shares these to each of its out-neighbours $i \in N_i^{out}$, where the weights $W_{i,j}$ are a weighted adjacency matrix and $\tilde{W}_{i,j}$ are defined as:

$$\tilde{W}_{i,j} = \begin{cases} \theta + (1-\theta)W_{i,j}, i = j, \\ (1-\theta)W_{i,j}, i \neq j, \end{cases} \tag{3.8}$$

where $\theta \in [0, \frac{1}{2}]$. The step update of $x^{(k)}$ is then performed using the following set of

equations:

$$x_i^{(k+1)} = \begin{cases} \sum_{i \in N_j^{in}} W_{i,j} x_i^{(k)} - \alpha \nabla f_i(z_i^{(k)}), & k = 0 \\ x_i^{(k)} + \sum_{i \in N_j^{in}} W_{i,j} x_i^{(k)} - \sum_{i \in N_j^{in}} \tilde{W}_{i,j} x_i^{(k-1)} - \alpha\Big(\nabla f_i(z_i^{(k)}) - \nabla f_i(z_i^{(k-1)})\Big), & k > 0 \end{cases} \tag{3.9a}$$

$$z_i^{(k)} = D(k) x_i^{(k)} \tag{3.9b}$$

$$D(k) = diag(W^k) 1_n \tag{3.9c}$$

where f_i is a convex and differentiable function only known to agent i, such that f_i: $\mathbb{R}^p \longrightarrow \mathbb{R}$, and $\nabla f_i(z)$ is the gradient of f_i at $z = z_i^{(k)}$ and α represents learning rate of the step function. A deeper analysis on the DEXTRA algorithm can be found on [40].

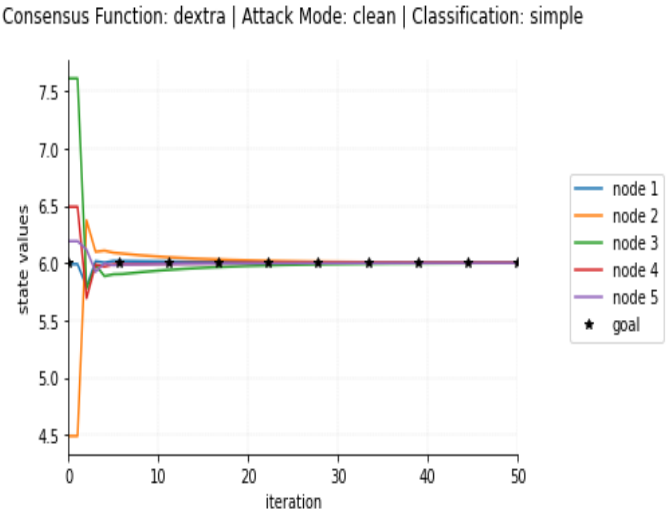


Figure 3.2: DEXTRA progression per node.

After performing a similar test with DEXTRA as the one that preceded Figure 3.1, Figure 3.2 is obtained. It is immediate that the DEXTRA algorithm is slower in achieving consensus than the Weighted Consensus. After 50 iterations, the average error is 9.487×10^{-5}. An error much lower than the calculated for the Weighted Consensus. Again, extending the limit to infinity demonstrates a tendency toward achieving a value close to the theoretical goal.

3.3 Changes to the Algorithm

In Section 2.5, Algorithm 3 was used to estimate the mean value of x through a mean of values considered by a classifier function. The method of progression was performed

through two methods, keeping an active sliding window, which was updated as to always maintain a collection of the best rated values received. This method can be useful in single agent systems, but not in a multi-agent setting, thus a new method to use the classifiers had to be developed, using the Consensus functions described previously to unify them. As both the Weighted Consensus and the DEXTRA functions use the weight matrix W, then it can be used by the classifiers to tell the agent which values of its in-neighbours are more valuable to the consensus.

For the MCD and OCSVM classifiers, the sliding window was kept to train the methods, and values accepted are used to change the adjacency matrix A, indicating that an agent whose value is accepted should be used in the consensus and setting the corresponding entry to zero if that agent was deemed an attacker. As it will be shown in later sections, this method can also reduce the effects of noise or possible rogue agents. Figure 3.3 shows how the classification is performed, where v is any value contributed by the in-neighbours of agent j, Q is a constant value introduced to cap the sliding window, and $model$ refers to the trained model for the ODM classifier.

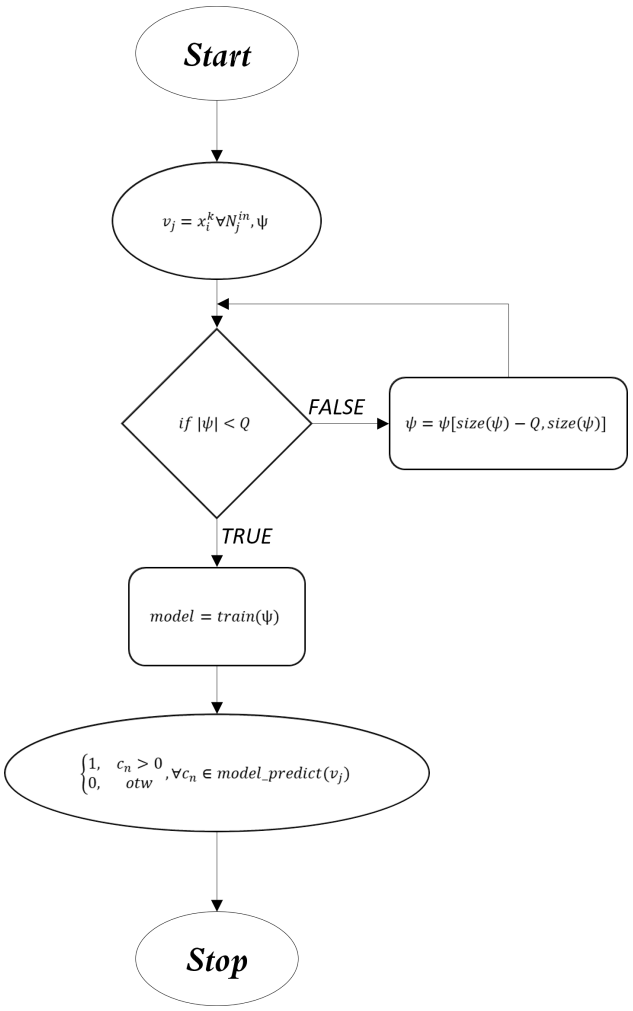


Figure 3.3: Outlier Detection Method Classification

In the case of the Rating and Reputation algorithms, instead of severing the link for the iteration, it is strengthened or weakened, depending on the score each of the agent's in-Neighbours' achieves. In order to further reduce the effects of noise in the Rating and Reputation algorithms, the score (S) is normalized through equation 3.10.

$$S_{norm} = \frac{r_i^{(k)} - min(r^{(k)})}{max(r^{(k)})} \tag{3.10}$$

where $r^{(k)}$ represents the set of rating scores calculated through equation 2.18 at the k^{th} iteration and $S_{norm} \in]0,1]$. Figure 3.4 shows how the classification is performed when using the Rating or Reputation methods, where S is the score produced by equations 2.18 or 2.20, respectively. Note that, in Figure 3.4, E_j represents the reputation score of the in-neighbours of j.

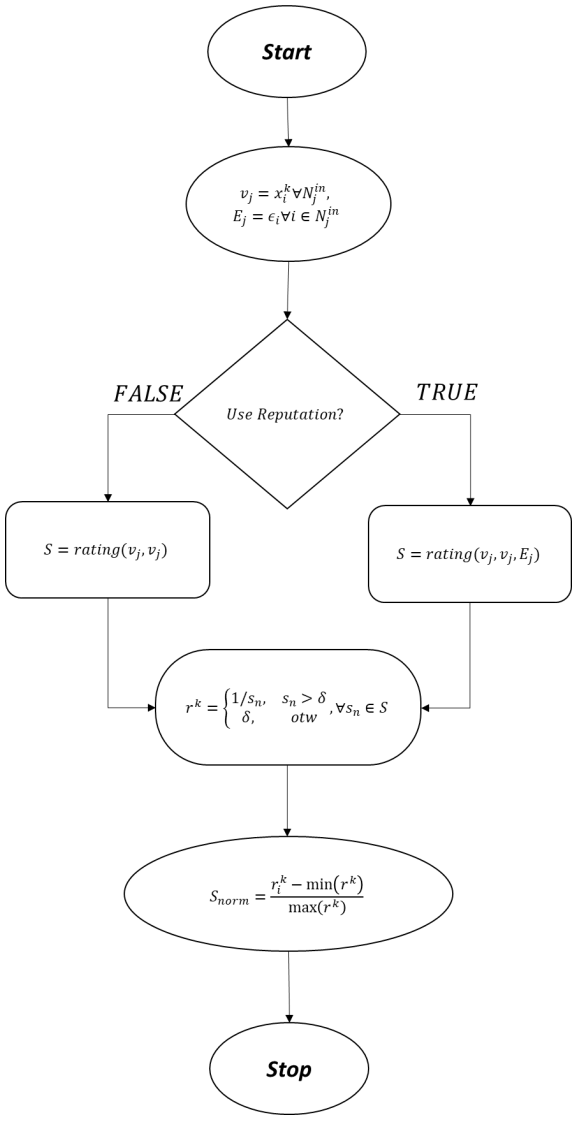


Figure 3.4: Rating and Reputation Classification

The ceiling δ imposed on the Rating and Reputation algorithms is used to prevent the appearance of $\frac{1}{0}$ undefined values, with the consequence of limiting the possible values of r^k to the interval $]0, 100]$. While this may result in a smaller distinction between scores when $k \longrightarrow \infty$ and the algorithm is closer to achieving consensus, it is not expected to produce an increase in the classifier's average error as agents with in-neighbours that allow them to produce scores lower than 0.01 must be at most within $\frac{1}{n}$ of the agent's own value, where $n = |N_j^{in}|$.

With the changes performed to the classification algorithms, the comparison can be performed using a multi-agent system.

3.4 Results

In this section the classification algorithms described are put into practice. It is split into three different scenarios where a comparison of results between the methods discussed in Chapter 3 are employed along with the Weighted Consensus or the DEXTRA equations to achieve consensus in the algorithm. Each case is characterized by how noise influences the results.

- Case 1: Algorithm performance without noisy agents in the network,
- Case 2: Algorithm performance against single noise moment,
- Case 3: Algorithm performance against persistent noise.

Before the consensus process is started, the initial state is generated randomly through the python module *numpy.random* and the adjacency matrix A is generated through function *strongly_connected_matrix*[1], such that first it generates a diagonal matrix with ones along the main diagonal, and then iteratively adds random edges until the resulting graph is strongly connected. Random edges between two nodes that are not yet connected have a probability 1/3 of being created. For the tests performed in this section, the adjacency matrix A used is

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.11}$$

and the initial state generated is $x^{(0)} = \begin{bmatrix} 5.99 & 4.49 & 7.61 & 6.49 & 6.19 \end{bmatrix}$. The theoretical consensus calculated through equation 3.2 is 6.00179.

In Case 1, the goal is to create a baseline for how the classification algorithms perform without noisy agents in the network. The expectation is that, without any noise, the agents will reach a consensus close to what is seen in Figures 3.1 and 3.2.

[1] Can be found in module "essentials_v2"of https://github.com/pmmatias6/Tese_Pedro_Matias

In Case 2, a single noise moment will be introduced into the system at a given time, representing a period where an agent is temporarily under the influence of exterior events. The expectation here is that although the system may suffer a fluctuation along the path to achieve consensus, it should still be able to recover and achieve a good performance.

Lastly, in Case 3 a persistent noise signal is introduced, overwriting one agent's information throughout the simulation. The expectation is that, through the classification methods, it will be possible to create a consensus between the remaining agents while the noisy agent becomes isolated and its influence in the system is reduced.

3.4.1 Algorithm performance without noisy agents in the network

Once x_0 and A were set and the theoretical consensus is obtained through equation 3.2, the simulation is conducted with the results being presented in the following figures.

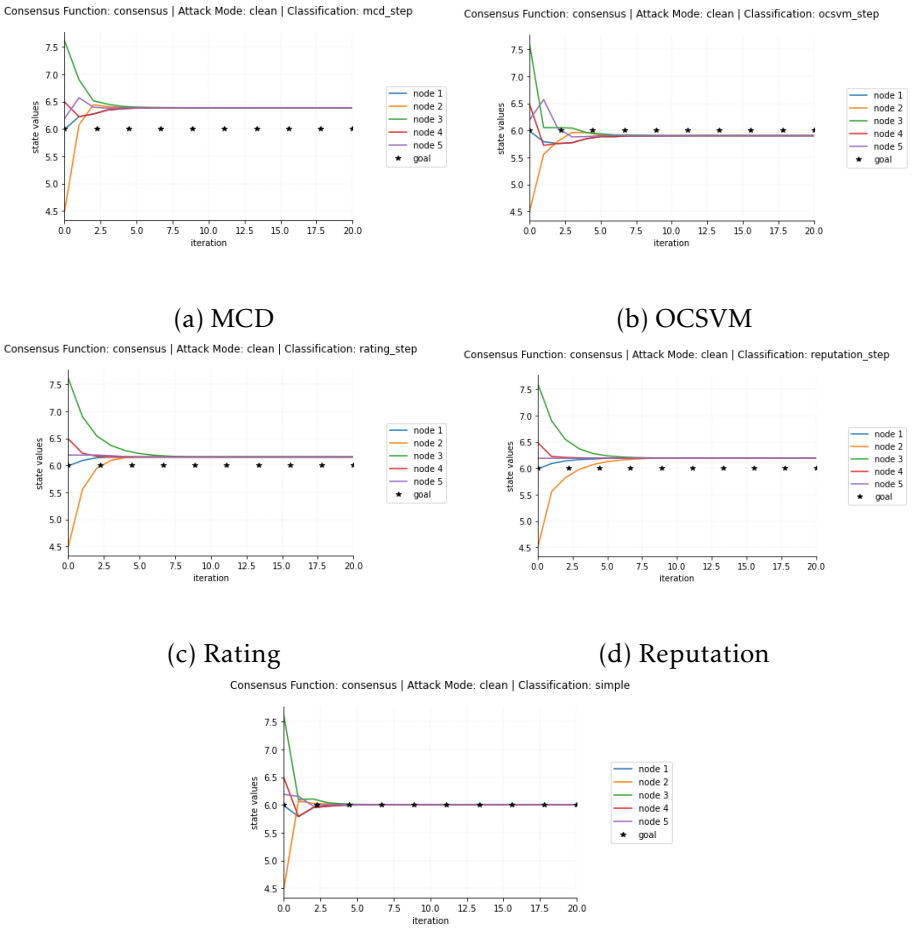
Figure 3.5 shows the evolution of the nodes' states per iteration, and their proximity to the goal when using the Weighted Consensus method. The results obtained show that the ODM classification (Figures 3.5a and 3.5b) obtained dissimilar results. While the OCSVM method approaches closer to the theoretical value, hereafter referred to as **goal**, the MCD method obtained the furthest distance from the goal. At the same time the scoring mechanisms (Figures 3.5c and 3.5d) obtained competitive values when compared with their ODM counterparts. This distance to the goal can be better demonstrated in Table 3.1, where the steady state of each mechanism's consensus and respective distance to the goal is shown.

Repeating the same process using DEXTRA revealed similar conclusions as Figure 3.6 shows. Again, there is disparity in the results obtained by each ODM classification mechanism studied, while the scoring mechanisms maintain close proximity between their results. With DEXTRA, and using Table 3.2 as a comparison of the steady state for each method, it is evident that in cases where there is no concern for attacks or the sensors are not subject to significant noise fluctuations there isn't need to make use of classification. At the same time, we get a glimpse at MCDs future inadequacy for the problem at hand.

This may be due to the fact that, as outlier detection methods, the MCD and OCSVM will tend to classify values too different from their peers as outliers, thus ignoring their influence. As this is performed through changing the connections in the adjacency matrix, the practical value found will be different than the goal, as the weight matrix W will also change throughout the consensus process.

Regarding the scoring methods studied, because equation 3.10 is used to alter the weights of each node in W, there will always be at least one node whose communication link is cut entirely, while the other nodes' weights are changed to reflect their proximity to each other's values through the normalization of equations 2.18 and 2.20, using equation 3.10.

3.4.1.1 Weighted Consensus



(a) MCD

(b) OCSVM

(c) Rating

(d) Reputation

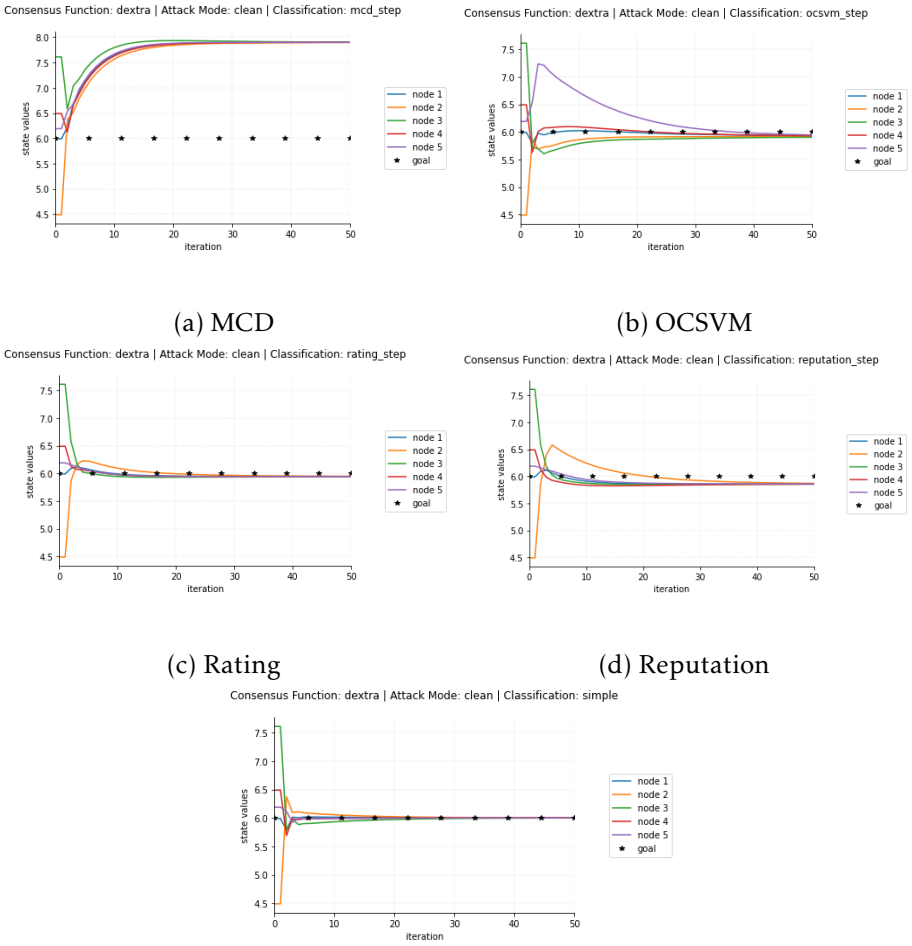
(e) w/o classification

Figure 3.5: Weighted Consensus progression per node, using different classification mechanisms.

Method	Steady State $\left(x^{(\infty)}\right)$	$\|\frac{\sum x_i^{(\infty)}}{N} - g\|$
w/o Classification	6.00408805	0.00229805
MCD	6.3868	0.3850
OCSVM	5.9013	0.1004
Rating	6.1548	0.1530
Reputation	6.1932	0.1914

Table 3.1: Weighted Consensus Steady State comparison in a system without noise.

3.4.1.2 DEXTRA



(a) MCD

(b) OCSVM

(c) Rating

(d) Reputation

(e) w/o classification

Figure 3.6: DEXTRA progression per node, using different classification mechanisms.

Method	Steady State $\left(x^{(\infty)}\right)$	$\|\frac{\sum x_i^{(\infty)}}{N} - g\|$
Simple	6.00179	9.96181×10^{-12}
MCD	7.8983	1.8965
OCSVM	5.916535349	0.08525
Rating	5.944988274	0.0568
Reputation	5.8578	0.1439

Table 3.2: DEXTRA Steady State comparison in a system without compromised nodes.

3.4.2 Algorithm performance against single noise moment

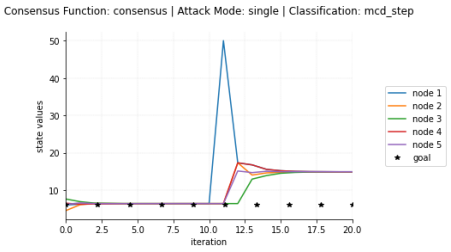
In this section, the network is assailed once by a noise peak where *node 1*, representing agent 1 in the network, has its value set to 50 at a point of the classification process. In Figures 3.7 and 3.8, the moment of the peak was iteration 10. No changes were performed on the classification mechanisms employed.

Figure 3.7 shows the evolution of the nodes' states using the Weighted Consensus, while Figure 3.8 shows the evolution of the nodes' states using the DEXTRA method.

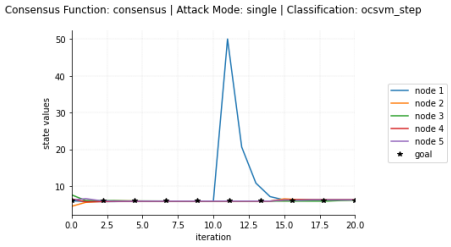
It is visible that the presence of noise affects the behaviour of the consensus. Where previously, without classification the deviation was minor, it is now much larger. At the same time, the use of classification mechanisms granted the system with resilience against the noise's effects, as shown in Table 3.3. The fact that the results obtained by the OCSVM, Rating and Reputation mechanisms are comparable to the previous simulations is a good indicator of their effectiveness in this scenario. Again, the MCD has fallen behind its competitors, showing a reduction in accuracy similar to the event without classification. This may be due to a few factors such as overfitting, but it is more likely that the method is not adequate for the problem at hand.

Table 3.4 shows the steady state of the nodes' states using the DEXTRA method, as well as the distance to the goal. Here, it is possible to see a reduction on all models' accuracy. This is likely due to the sudden momentum generated by the gradients present in equations 3.9a and 3.9b. The sudden noise introduction may have revealed a possible fragility with the DEXTRA algorithm.

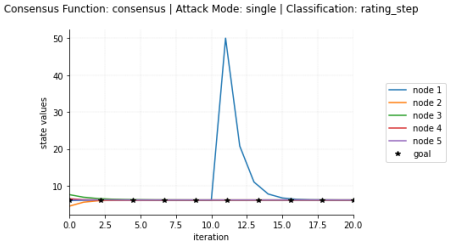
3.4.2.1 Weighted Consensus



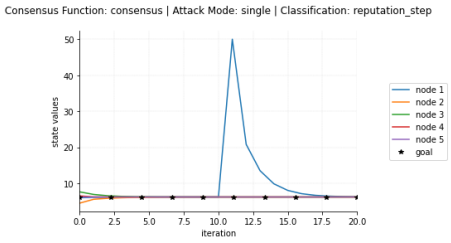
(a) MCD



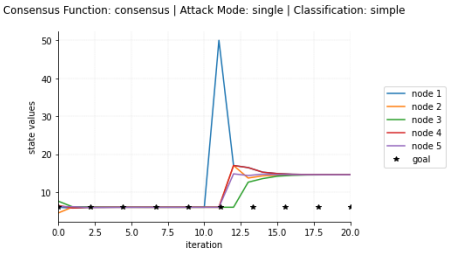
(b) OCSVM



(c) Rating



(d) Reputation



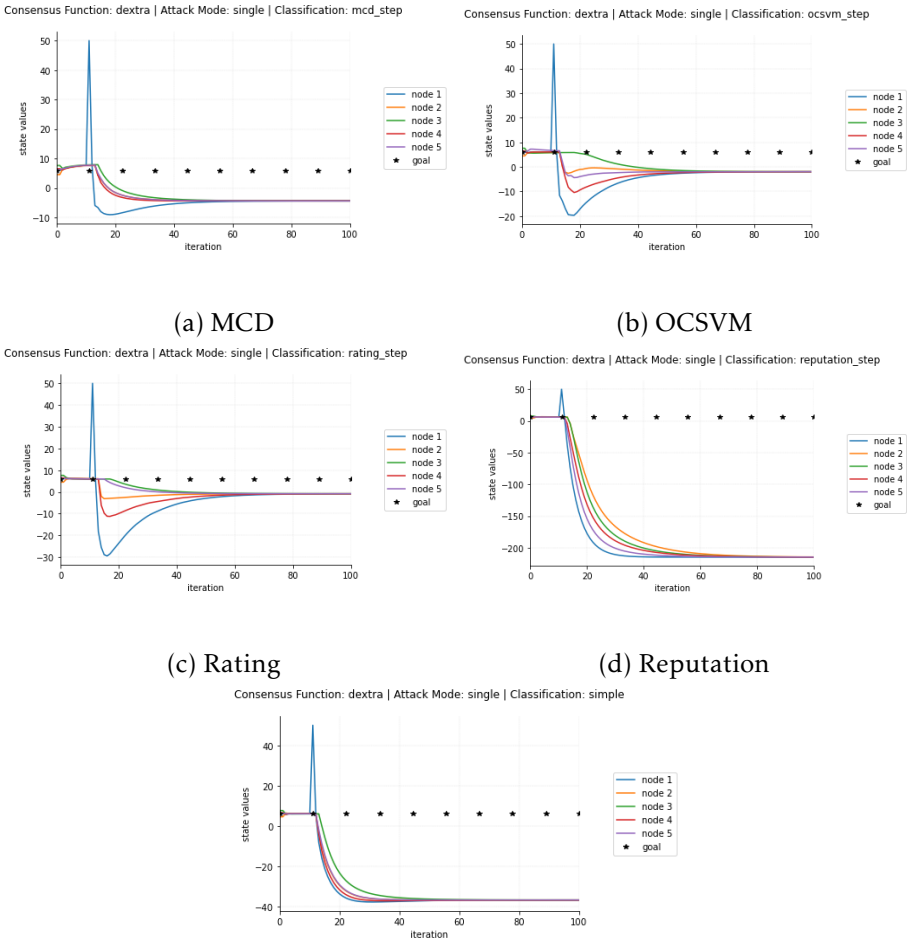
(e) w/o classification

Figure 3.7: Weighted Consensus progression per node with single noise moment, using different classification mechanisms.

Method	Steady State $\left(x^{(\infty)}\right)$	$\|\frac{\sum x_i^{(\infty)}}{N} - g\|$
Simple	14.5819	8.58012
MCD	14.8900	8.8882
OCSVM	6.2414	0.2396
Rating	6.1551	0.1533
Reputation	6.1936	0.1918

Table 3.3: Weighted Consensus Steady State comparison of the nodes with an isolated noise moment.

3.4.2.2 DEXTRA



(a) MCD

(b) OCSVM

(c) Rating

(d) Reputation

(e) w/o classification

Figure 3.8: DEXTRA progression per node with single noise moment, using different classification mechanisms.

Method	Steady State $\left(x^{(\infty)}\right)$	$\|\frac{\sum x_i^{(\infty)}}{N} - g\|$
Simple	-36.8792	42.8810
MCD	-4.3970	10.3988
OCSVM	-1.9480	7.9498
Rating	-0.8391	6.8409
Reputation	-214.6443	220.6461

Table 3.4: DEXTRA Steady State comparison of the nodes with an isolated noise moment.

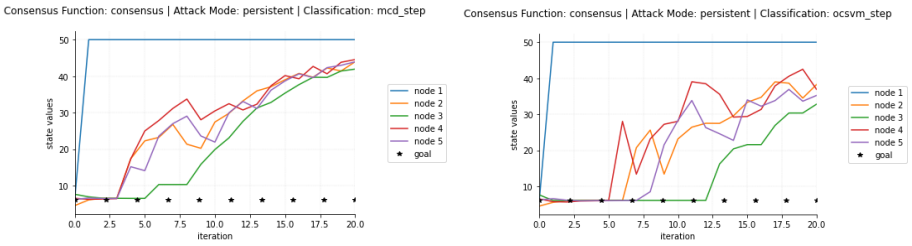
3.4.3 Algorithm performance against persistent noise

In this section, a persistent noise signal infiltrates the network, overwriting *node 1*'s values independently of its own value. Here, the objective is to assess the true resilience of each mechanism. There no longer is a possibility of consensus between all nodes, but the question to be answered is whether the remaining nodes can overcome the influence of the noise and reach a consensus among each other without straying further from the goal set beforehand.

In this case, Figures 3.9 and 3.10 show that all but the scoring classification mechanisms fell prey to the noisy node and ended with a consensus diverted from the goal. Particularly, in DEXTRA's case, the momentum generated by the persistent noise's presence provoked an exponential and out of bounds behaviour for the trials without classification, OCSVM and MCD. In regards to the Rating and Reputation scoring mechanisms, their nature allowed for a greater resilience against the noise, even isolating the faulty node while achieving consensus.

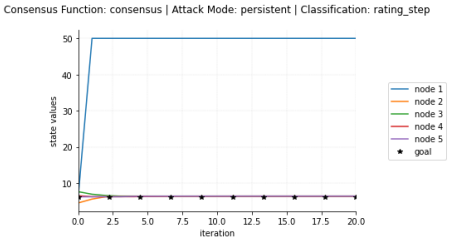
In Tables 3.5 and 3.6 this achievement can be noted such that, in both cases, the Reputation mechanism achieved the consensus closest to the goal. Despite this, it failed to reach consensus using the DEXTRA algorithm, as shown in Figure 3.10d, where *node 4* (corresponding with agent no. 4 of the network) suffered a slight deviation from the consensus value and instead achieved its steady state at the value of 4.6685 (resulting in a distance to the goal of 1.3333 instead). The reason behind this deviation might be due to the communication post classification shifting in such a way that this node became isolated of the network after being categorised as deviant at some point during the simulation.

3.4.3.1 Weighted Consensus

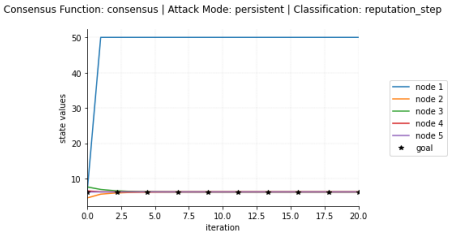


(a) MCD

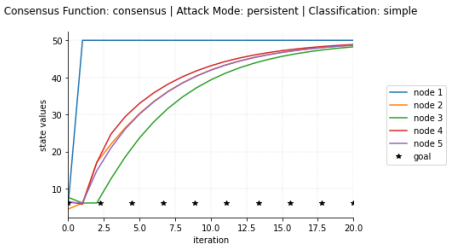
(b) OCSVM



(c) Rating



(d) Reputation



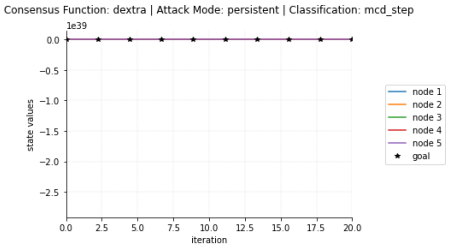
(e) w/o classification

Figure 3.9: Weighted Consensus progression per node with persistent noise moment, using different classification mechanisms.

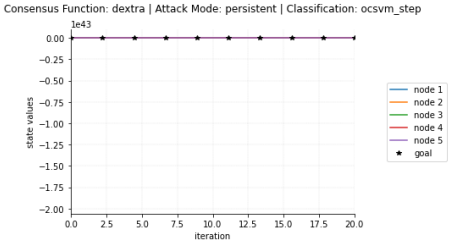
Method	Steady State $\left(x^{(\infty)}\right)$	$\lVert\frac{\sum x_i^{(\infty)}}{N} - g\rVert$
Simple	50	43.99821
MCD	50	43.99821
OCSVM	49.9999	43.9981
Rating	6.3797	0.3779
Reputation	6.1954	0.1937

Table 3.5: Weighted Consensus Steady State comparison of uncompromised nodes with persistent noise moment.

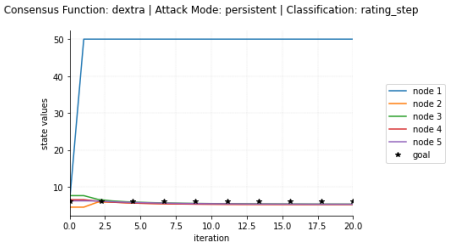
3.4.3.2 DEXTRA



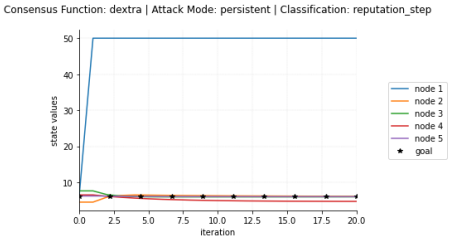
(a) MCD



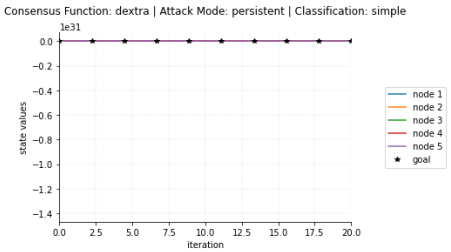
(b) OCSVM



(c) Rating



(d) Reputation



(e) w/o classification

Figure 3.10: DEXTRA progression per node with persistent noise moment, using different classification mechanisms.

Method	Steady State $\left(x^{(\infty)}\right)$	$\|\frac{\sum x_i^{(\infty)}}{N} - g\|$
Simple	-1.15471×10^{31}	1.15471×10^{31}
MCD	-1.80015×10^{39}	1.80015×10^{39}
OCSVM	-1.30414×10^{43}	1.30414×10^{43}
Rating	5.2601	0.7416
Reputation	5.9999	0.00187

Table 3.6: DEXTRA Steady State comparison of uncompromised nodes with persistent noise moment.

4

Conclusion

The studied problem of achieving a consensus with high accuracy in dynamic and distributed systems is a complex issue in academia, especially when the system dynamics are unknown and it is susceptible to faulty data, whether it be through attacks by outside entities or noisy nodes. The proposed methods and mechanisms have their own place and function within the scope of the issue, but static methods tend to fail when the scenario becomes more complicated while the scoring nature of reputation and rating mechanisms tend to be more stable overall, thus achieving resilience.

In simulation, it is shown that mechanisms which involve rating and reputation, be it in dynamic systems or multi-agent distributed systems, perform better overall. Further, in situations where there is a a danger of the network being compromised by rogue agents (Figures 3.7, 3.8, 3.9, 3.10) these mechanisms maintain a greater resilience while achieving consensus (Tables 3.3, 3.4, 3.5, 3.6), and have the advantage of being able to identify the faulty node and isolating it.

In the scenario of pinpointing the location of a forest fire, the deviations obtained are still too large. If the values are taken as the radius of a circumference (in kilometres) encompassing an area where the fire is supposed to be developing, there isn't a consensus on which method has achieved better results. In Case 1 (section 3.4.1) the ideal situation of a system that does not suffer from any sort of large noise fluctuations or attacks from evil agents obtained weak results in regards to accuracy, with the Rating and the OCSVM methods obtain the better results overall, yet still obtaining error margins between 56 and 153 metres - distances that still accrue too large of an error to be reliable without employing other scouting methods. When noise was first introduced in Case 2 (section 3.4.2), the DEXTRA algorithm would have shown itself unable to provide an error margin below 7-8 kilometres in such a scenario. In the same situation, Weighted Consensus allowed for a minimum error margin close to 150 metres, when using the Rating mechanism. With the existence of persistent noise in Case 3 (section 3.4.3) all the ODM classification methods can be declared as failures. In regards to scoring mechanisms, Reputation obtained the highest accuracy when joined with the DEXTRA algorithm of all classification mechanisms employed in the three studied cases, with an error margin of less than 2 metres.

This may be the only situation where success can be confirmed, yet must be taken with a grain of salt as consensus was not achieved for all non-compromised nodes, as shown in Figure 3.10d.

With this, we conclude that, although the scoring mechanisms show promising features in this field, further validation of the results obtained should be performed in the future, through the exposure of the algorithms developed to real data and real-world situations.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] J. Lloret et al. "A wireless sensor network deployment for rural and forest fire detection and verification". In: *sensors* 9.11 (2009), pp. 8722–8747 (cit. on p. 1).

[3] C. V. A et al. *Science for Disaster Risk Management 2020*. Scientific analysis or review KJ-NA-30183-EN-N (online),KJ-NA-30183-EN-C (print). Luxembourg (Luxembourg), 2021. DOI: 10.2760/438998(online),10.2760/571085(print) (cit. on p. 1).

[4] R. Surette. "The thinking eye: Pros and cons of second generation CCTV surveillance systems". In: *Policing: An International Journal of Police Strategies & Management* (2005) (cit. on p. 1).

[5] F. X. Catry et al. "Modeling and mapping wildfire ignition risk in Portugal". In: *International Journal of Wildland Fire* 18.8 (2009), pp. 921–931 (cit. on p. 1).

[6] A. Capponi et al. "A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities". In: *IEEE communications surveys & tutorials* 21.3 (2019), pp. 2419–2465 (cit. on p. 1).

[7] Z. Xu et al. "Mobile crowd sensing of human-like intelligence using social sensors: A survey". In: *Neurocomputing* 279 (2018), pp. 3–10 (cit. on p. 1).

[8] M. Hefeeda and M. Bagheri. "Wireless sensor networks for early detection of forest fires". In: *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*. IEEE. 2007, pp. 1–6 (cit. on p. 1).

[9] H. Zheng et al. "Monitoring surface water quality using social media in the context of citizen science". In: *Hydrology and Earth System Sciences* 21.2 (2017), pp. 949–961 (cit. on p. 1).

[10] D. Silvestre. "Reputation-based Method to Deal with Bad Sensor Data". In: *IEEE Control Systems Letters* (2020) (cit. on pp. 1, 11).

[11] C. R. Perez-Toro, R. K. Panta, and S. Bagchi. "RDAS: reputation-based resilient data aggregation in sensor network". In: *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. IEEE. 2010, pp. 1–9 (cit. on p. 1).

[12] W. Van der Hoek and M. Wooldridge. "Multi-agent systems". In: *Foundations of Artificial Intelligence* 3 (2008), pp. 887–928 (cit. on pp. 1, 19).

[13] L. Busoniu, R. Babuska, and B. De Schutter. "Multi-agent reinforcement learning: A survey". In: *2006 9th International Conference on Control, Automation, Robotics and Vision*. IEEE. 2006, pp. 1–6 (cit. on pp. 1, 19).

[14] D. Silvestre, J. P. Hespanha, and C. Silvestre. "Broadcast and gossip stochastic average consensus algorithms in directed topologies". In: *IEEE Transactions on Control of Network Systems* 6.2 (2018), pp. 474–486 (cit. on p. 2).

[15] D. Silvestre. "Optool—an optimization toolbox for iterative algorithms". In: *SoftwareX* 11 (2020), p. 100371 (cit. on p. 2).

[16] R. Ribeiro, D. Silvestre, and C. Silvestre. "Decentralized control for multi-agent missions based on flocking rules". In: *Portuguese Conference on Automatic Control*. Springer. 2020, pp. 445–454 (cit. on p. 2).

[17] D. F. Gleich. "PageRank beyond the Web". In: *siam REVIEW* 57.3 (2015), pp. 321–363 (cit. on p. 2).

[18] D. Silvestre, J. Hespanha, and C. Silvestre. "Desynchronization for decentralized medium access control based on gauss-seidel iterations". In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 4049–4054 (cit. on p. 2).

[19] A. Cristofaro and T. A. Johansen. "Fault tolerant control allocation using unknown input observers". In: *Automatica* 50.7 (2014), pp. 1891–1897 (cit. on p. 2).

[20] K. Manandhar et al. "Detection of faults and attacks including false data injection attack in smart grid using Kalman filter". In: *IEEE transactions on control of network systems* 1.4 (2014), pp. 370–379 (cit. on p. 2).

[21] P. P. Menon and C. Edwards. "Robust fault estimation using relative information in linear multi-agent networks". In: *IEEE Transactions on Automatic Control* 59.2 (2013), pp. 477–482 (cit. on p. 2).

[22] J. K. Scott et al. "Constrained zonotopes: A new tool for set-based estimation and fault detection". In: *Automatica* 69 (2016), pp. 126–136 (cit. on p. 2).

[23] J. K. Scott et al. "Input design for guaranteed fault diagnosis using zonotopes". In: *Automatica* 50.6 (2014), pp. 1580–1589 (cit. on p. 2).

[24] D. Silvestre, J. P. Hespanha, and C. Silvestre. "Resilient desynchronization for decentralized medium access control". In: *IEEE Control Systems Letters* 5.3 (2020), pp. 803–808 (cit. on p. 2).

[25] D. Silvestre et al. "Fault detection for LPV systems using set-valued observers: A coprime factorization approach". In: *Systems & Control Letters* 106 (2017), pp. 32–39 (cit. on p. 2).

[26] D. Silvestre et al. "Stochastic and deterministic fault detection for randomized gossip algorithms". In: *Automatica* 78 (2017), pp. 46–60 (cit. on p. 2).

[27] G. Gan and M. K.-P. Ng. "K-means clustering with outlier removal". In: *Pattern Recognition Letters* 90 (2017), pp. 8–14 (cit. on p. 2).

[28] S. M. Dibaji and H. Ishii. "Consensus of second-order multi-agent systems in the presence of locally bounded faults". In: *Systems & Control Letters* 79 (2015), pp. 23–29 (cit. on p. 2).

[29] G. Ramos, D. Silvestre, and C. Silvestre. "A general discrete-time method to achieve resilience in consensus algorithms". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 2702–2707 (cit. on p. 2).

[30] A. Atkinson and D. Hawkins. "Identification of Outliers." In: *Biometrics* 37 (Dec. 1981), p. 860. DOI: 10.2307/2530182 (cit. on p. 3).

[31] K. M. Liu Fei Tony and Z.-H. Zhou. "Isolation Forest". In: *2008 eighth ieee international conference on data mining* (2008), pp. 413–422 (cit. on p. 4).

[32] M. M. Breunig et al. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104 (cit. on pp. 5, 6).

[33] B. Schölkopf et al. "Support vector method for novelty detection." In: *NIPS*. Vol. 12. Citeseer. 1999, pp. 582–588 (cit. on pp. 6, 9).

[34] P. J. Rousseeuw and K. V. Driessen. "A Fast Algorithm for the Minimum Covariance Determinant Estimator". In: *Technometrics* 41.3 (1999), pp. 212–223. DOI: 10.1080/00401706.1999.10485670. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.1999.10485670. URL: https://www.tandfonline.com/doi/abs/10.1080/00401706.1999.10485670 (cit. on p. 10).

[35] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 15).

[36] Z. Li et al. "Designing Fully Distributed Consensus Protocols for Linear Multi-Agent Systems With Directed Graphs". In: *IEEE Transactions on Automatic Control* 60.4 (2015), pp. 1152–1157. DOI: 10.1109/TAC.2014.2350391 (cit. on p. 19).

[37] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. "Dynamic consensus on mobile networks". In: *IFAC world congress*. 2005, pp. 1–6 (cit. on pp. 19, 21).

[38] R. Olfati-Saber and R. M. Murray. "Consensus problems in networks of agents with switching topology and time-delays". In: *IEEE Transactions on automatic control* 49.9 (2004), pp. 1520–1533 (cit. on pp. 19, 21).

[39] C. Xi and U. A. Khan. "On the linear convergence of distributed optimization over directed graphs". In: *arXiv preprint arXiv:1510.02149* (2015) (cit. on p. 19).

[40] C. Xi and U. A. Khan. "DEXTRA: A fast algorithm for optimization over directed graphs". In: *IEEE Transactions on Automatic Control* 62.10 (2017), pp. 4980–4993 (cit. on pp. 19, 21, 22).

[41] X. Sun and J. Zhang. "An exact first-order algorithm for decentralized consensus optimization". In: (2014) (cit. on p. 21).