Internet of Things 9 (2020) 100167

Contents lists available at ScienceDirect

Internet of Things

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

Research article

Scalable and Robust Outlier Detector using Hierarchical Clustering and Long Short-Term Memory (LSTM) Neural Network for the Internet of Things

Raj Mani Shukla*, Shamik Sengupta

Department of Computer Science and Engineering University of Nevada, Reno, USA

ARTICLE INFO

Article history:
Received 17 June 2019
Revised 19 January 2020
Accepted 19 January 2020
Available online 30 January 2020

Keywords:
Outlier detector
Hierarchical clustering
LSTM neural network
M-estimator

ABSTRACT

The emerging centralized entities, like cloud, edge, or Software-Defined Network (SDN), make automated decisions for the Internet of Things (IoT) applications based on the measured data from several sensors. However, the malicious injection of the anomalies or outliers in measured sensor data may disrupt the automated decision making capabilities of the applications running at the centralized location. Therefore, the detection of such outliers is an essential problem for IoT that needs to be researched out. This paper presents a scalable outlier detector that uses Hierarchical clustering in conjunction with Long Short-Term Memory (LSTM) neural network. Hierarchical clustering provides scalability to the outlier detector by finding correlated sensors. The LSTM neural network is coupled with the robust statistics, M-estimator, to accurately detect outliers in time-series data. The simulation results on different data-sets show that the proposed method has an accuracy of more than 90% for different attack strength. Also, the model parameter can be tuned according to the application requirement so that the outlier detector can be tailored to either precision or recall sensitive.

1. Introduction

Internet of Things (IoT) envisions the presence of ubiquitous devices and sensors for continuous measuring of environmental parameters [1–3]. The measured data is transferred to the centralized entities, like cloud, edge, or Software-Defined Network (SDN) controller, where the data is analyzed and processed to extract useful information and assist in IoT-enabled automated applications [1,4]. There are many IoT applications that use measured data in time-series format [5,6]. However, it is possible that the time-series data is manipulated by an adversary, inducing outliers in it [7,8]. Exposure to the Internet may allow adversary to remotely control sensor and alter its measured value. Multiple recent studies have emphasized the analysis and detection of such attacks [9,10]. The main motivation for such attacks is to disrupt the reliability of IoT service providers [11,12].

Frequent occurrence of outliers due to compromised sensors reduces the effectiveness of intelligent and automated decision making capabilities of the applications. The outliers may result in system breakdown, thus reducing the user's trust regarding a service. For an enterprise, this may result in a long-term revenue loss due to bad customer experience or brand perception. For example, as reported in [11], if a particular mobile application fails, it is likely that 48% of users will use

* Corresponding author.
E-mail addresses: rshukla@unr.edu (R.M. Shukla), ssengupta@unr.edu (S. Sengupta).

https://doi.org/10.1016/j.iot.2020.100167
2542-6605/

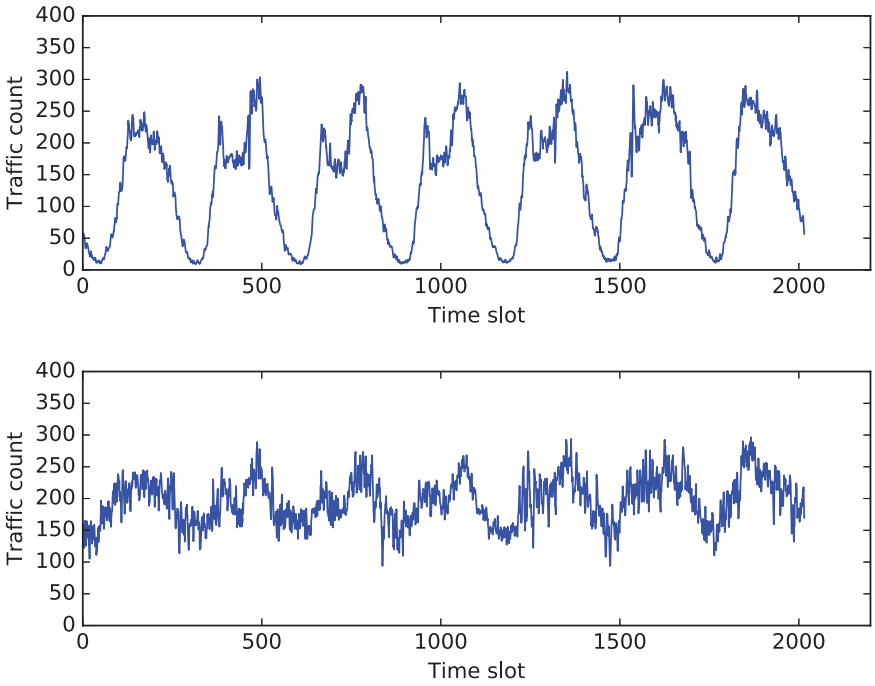

Fig. 1. Traffic count at different locations.

it again, 34% users may switch to some other competitive application, and 31% of them will tell to friends about their bad experience [11,12]. Therefore, outliers need to be effectively detected and filtered out.

In the context of IoT, detecting outliers is difficult due to the volume and variability of the data. Also, a large number of sensors are present in IoT, making problem further difficult. The IoT devices generate data having diverse patterns and characteristics. Therefore, developing anomaly or outlier detector satisfying the need of every sensor is difficult. Fig. 1 shows the vehicular traffic near two different locations. It can be observed that the two time-series differ significantly. Therefore, an outlier detector developed for one of them cannot be effectively used for the other. Thus, the traditional anomaly detectors, that does not consider the diversity in the data and presence of a large number of sensors, cannot be effectively used in the IoT domain. If conventional methods are applied in the IoT domain, then a separate model is required for each sensor. Such a system suffers from scalability issue due to the presence of a large number of sensors in IoT. Therefore, for IoT, a scalable outlier detector needs to be researched out.

Further, several state-of-the-art anomaly detectors relies on different assumptions about the sample data-set. For example, the conventional Grubbâs test assumes that the sample size should be large enough for high detection accuracy [13]. The assumption about the sample size of data cannot always hold, and therefore, it is not an effective method for every condition. Outlier detectors relying on statistical properties of sample data assume that the data comes from a particular distribution [14]. The assumption cannot always be true. Therefore, although the statistical characteristics of a sample data-set are useful for detecting outliers, relying solely on them reduces their effectiveness. Many outlier detectors use supervised learning on training data having labeled (or known) anomaly patterns [15]. However, due to diversity and volume of data, the labeled anomaly patterns may not always be available. Therefore, supervised learning on existing anomaly patterns is not an efficient technique.

In this paper, we spell out an outlier detector that is scalable and does not depend on the underlying assumptions about the sample data or labeled anomalies. We employ Hierarchical clustering to impart scalability to the outlier detector. Hierarchical clustering finds the correlated sensors and forms clusters. The sensors in a cluster have similar measurement patterns and an outlier detector developed for one of them can be used for every sensor in a cluster. For detecting outliers, we partition time-series into segments so that the effect of data distribution is minimized. We use robust statistics M-estimators coupled with the LSTM neural network to detect outliers in a time series segment. The proposed method that combines the M-estimators and LSTM neural network can effectively detect up to 50% outliers in every time-series segment. The main contributions of this paper are as follows:

- Proposal of a Hierarchical clustering method for effectively finding sensors with correlated measurements. The Hierarchical clustering is used for providing scalability to the outlier detector.
- Investigation of an outlier detector that uses the cluster information and employs M-estimator coupled with the LSTM neural network.

- Evaluation of the proposed method using two disparate data-sets to validate its effectiveness and comparison with the other state-of-the-art techniques.

The rest of this paper is organized as follows. Section 2 provides a brief literature survey on this topic. Section 3 present the problem statements and the description of the methods used in this paper. Section 4 describes an overview of the proposed outlier detector. In Section 5, we investigate the proposed outlier detector. In Section 6, simulation experiments and results are presented and Section 7 concludes this paper.

2. Literature Survey

There have been several works done on anomaly or outlier detection for IoT-based applications. Thanigaivelan et al. have presented a distributed approach to detect anomalies for IoT in [16]. In the given work, the anomaly detection task is distributed between nodes and router. The nodes in the system record data rate or packet size for detecting anomalies. The router located at the edge collects information about anomalies, and based on the obtained correlated values, verifies a point as anomalous. Li et al. have proposed a distributed anomaly detection and trust management services for the IoT sensors in [17]. In the given method, an IoT node verifies for any anomaly in the other IoT nodes that are present within its transmission range. Based on the information, each IoT node informs to the sensors in its neighborhood about the possible outliers. Then, the data is fused according to the Dempster-Shafer Theory (DST) [18]. To verify the trustworthiness of the collected data, the proposed method checks if the measured sensor data is consistent with its neighborhood sensors. The approaches mentioned by Thanigaivelan et al. and Li et al. are distributed in nature. Therefore, they pose a problem due to the low computing power of individual nodes. The rise of the Big data in IoT requires a massive amount of data to be processed, and thus, the above approaches are not suitable for the IoT domain. Furthermore, the per sensor model is not scalable for IoT due to the presence of a large number of sensors.

Bhattacharjee et al. [9] have proposed the use of statistical characteristic of sample data, Harmonic-Arithmetic mean (HM-AM) ratio, for anomaly detection. The proposed method is useful for the point as well as group anomalies. However, the work has used the data-sets where $HM \approxeq AM$ such that the data points are close to each other. For the data-sets having significant variations in measured values, the method cannot be effective as in such a case $HM - AM$ ratio is not close to one. Furthermore, the given method assumes that the data sample is normally distributed that cannot be correct every time. The importance of finding local outliers has been described by Markus et al. in [19]. The local outliers are the points that are not close to their neighborhood points. For this purpose, the paper has described a Local Outlier Factor (LOF) for every data point. The LOF determines the amount by which an object is separated from its neighborhood points. If a point is well close to its neighbors, its LOF value is high. For every other point, the method assigns an upper and lower bound on LOF value. The LOF is a useful method for finding local outliers as local factors are sometimes more meaningful and provide better insights into the data than the global factors. Su et al. [20] have proposed an outlier detector based on the density of the scattered data. In this work, the local outlier factor is redefined by taking the advantage of the distribution of the data. The work pre-processes the data using Rough Clustering based on Multi-Level Queries (RCMLQ) method, and then Efficient Density-based Local Outlier detection for Scattered data (E2DLOS) is proposed for anomaly detection. Bhattacharjee et al., Markus et al., and Su et al. have proposed methods that rely on the data having a certain distribution. Often the sample size is small enough, and thus, the sample data cannot be categorized into specific distribution, or distribution advantages cannot be made. Thus, the above approaches for the anomaly detection are not valid in such types of cases.

In [21], Pandeeswari et al. have described using the neural network for anomaly detection in the cloud data. The given method employs a combination of Fuzzy C-Means and Artificial Neural Network (FCM-ANN) methods that improve the detection accuracy as compared to the techniques like Naïve Bayes classifier and simple ANN algorithm. The given methodology is found to be effective in the presence of low-frequency outliers and performs with a low false alarm rate. Vallis et al. [22] have presented Extreme Studentized Deviate (ESD) and Seasonal Extreme Studentized Deviate (S-ESD) methods for the anomaly detection in Twitter data. The ESD and S-ESD methods use Median and Median Absolute Deviation (MAD) to determine the outliers. However, the methods proposed by Pandeeswari et al. and Vallis et al. detect anomalies on the aggregate data collected in the cloud from a large number of devices. Thus, they are suitable for the cloud computing data analysis. The anomalies in the measured data in a particular IoT node has not been described in the given works.

In contrast to the above works, this paper develops an outlier detector that does not relies on computing capabilities of the individual IoT nodes. It does not depend upon data samples having specific distribution. Furthermore, the proposed outlier detector is both scalable and accurate. It is not tailored to a specific sensor. Also, the given outlier detector does not depend upon the training data containing labeled anomaly patterns.

3. Preliminaries

Before proceeding to the proposed methodology, we explain below the problem statements and the motivations for using Hierarchical clustering and Long Short-Term Memory (LSTM) neural networks, in the context of the proposed scalable outlier detector for Internet of Things (IoT) sensors.

3.1. Problem statements

The outliers in time-series data are the points in the sample that does not have an expected value. They have value greater or less than the actual value because a malicious adversary may have altered the sensor data. The problem to find outliers in IoT is divided into two parts.

The different IoT sensors provide a diverse range of time-series patterns. There are also a large number of sensors in IoT. Therefore, the first problem is to determine correlated sensors in IoT. We propose to use Hierarchical clustering to find the sensors that have correlated measurements. Once correlated sensors are obtained, the second step is to develop an outlier detector for a correlated cluster of sensors. Therefore, the second problem is to accurately find anomalous points in the sample. We use Long Short-Term Memory (LSTM) neural network coupled with the robust statistical analysis to find the anomalous points or outliers in IoT.

3.2. Hierarchical clustering

Hierarchical clustering is one of the many clustering techniques that is used for the data analysis purpose [23]. We use Hierarchical clustering since it is flexible and has fewer assumptions about the underlying data patterns. The methods like K-means algorithm assumes that the number of clusters is known in advance. The density-based methods like Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Ordering Points to Identify the Clustering Structure (OPTICS) are based on the different density between clusters [24]. The distribution-based algorithm like expectation maximization method assumes that the data comes from a particular distribution like normal or gaussian [25]. Support Vector Machines (SVM) is also a popular mechanism, but it is computationally expensive [26].

In the given context, we have a large number of sensors providing disparate time-series patterns. The problem is to determine sensors that provide similar measurement patterns. Thus, it is not known in advance the number of correlated clusters of sensors. Additionally, we consider that sample distribution is not known in advance. The clustering method should also be less computation-intensive. Since, the Hierarchical clustering is a light-weight and non-parametric technique, it aptly fits in our problem.

In the Hierarchical clustering method, initially, all the data points are considered as an individual cluster. Subsequently, it repeatedly performs the two steps. First, it identifies the two clusters that are close to each other. Then, it merges those clusters to form one big cluster. The process is repeated until the entire data-set is collected in a single cluster. Depending upon the variation in the distance between two clusters that are being merged, the user can determine the number of clusters for an application.

3.3. LSTM neural network

There are different methods for time-series analysis. For example, the Auto Regressive Integrated Moving Average (ARIMA) is often used for time-series analysis. However, LSTM network is found to be efficient in capturing long-term relations between temporally separated data points in sequential data. LSTM is suitable for time-series data analysis as it has memory units. Thus, for non-linear and time-series data-sets the LSTM is often suitable as compared to other methods [27]. Therefore, this paper has used LSTM for the outlier detector problem.

LSTM neural network contain cells for processing data where each cell has input, output, and forget gate. The gates keep track of amount of information transferred from input to output and next cell. The input gate controls the extent up to which input value is passed to the cell, forget gate controls the information to be remembered and passed to the next cell, and output gate controls the amount of information from input and previous gate to be used as the output [27]. Thus, the cell in the LSTM neural network acts as its memory and helps in extracting long-term dependencies in the data.

4. Outlier Detector Architecture

Fig. 2 shows the overview of the proposed model. The proposed model is deployed at the centralized cloud unit. The measured data from spatially distributed sensors is transfered to the centralized unit for use in different applications [28]. In cloud, before using data for a specific application, it is analyzed using the given model to detect the presence of outliers.

The proposed model consists of the two modules, namely the sensor clustering module and outlier detection module. As shown in the figure, the time-series data is first passed through the filter to remove noise and is then fed to the clustering and the outlier detection modules.

The clustering process is performed offline on the recorded time-series data from every sensor. We assume that the data used for the sensor clustering does not contain any outlier. Before clustering, we use the Principal Component Analysis (PCA) on time-series to reduce its dimensions. Thus, the data fed to the Hierarchical clustering has lower dimensionality and makes clustering process computationally efficient. After PCA analysis, the Hierarchical clustering determines the correlated sensors.

The outlier detection module is performed both offline and online. During offline phase, the LSTM neural network is trained on known data-set for its optimized performance. During online phase, the module is used for detecting outliers. Outlier detection module contains a statistical module (M-estimator) and LSTM neural network module. The time-series data

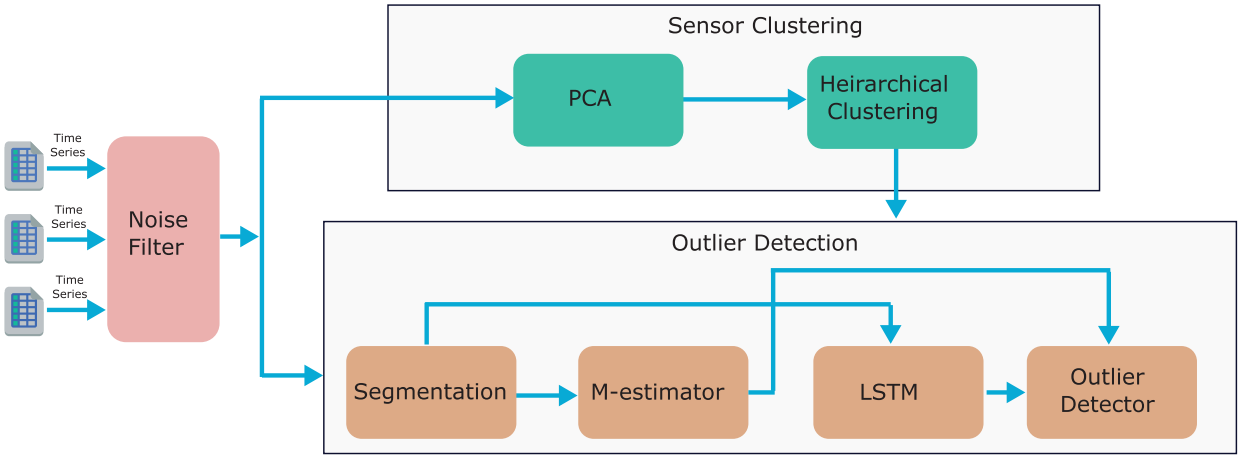

Fig. 2. Architecture describing outlier detection process.

is first segmented, and then on every segment statistical analysis is performed. The time-series segment is also fed to the LSTM neural network for estimating range in every segment. The LSTM neural network uses the processed information from clustering module. The processed data from both the M-estimator module and LSTM neural network module is combined in the outlier detector to find the outliers in every segment. The following section describes the different modules and their functioning in detail.

5. Methodology

We present below the detailed description of the proposed outlier detector. In our analysis, the time is discretized into the length of the interval τ, and every such interval is called a time-slot. We use variable i, j, and k to denote location, day, and time-slot respectively. The sensor at location i is represented using variable s_i. Variable L represents the total number of sensors such that $i \in \{1, 2, ..., L\}$. The data is stored in a matrix n. The values in matrix n are transformed by normalizing them between 0 and 1 using the min-max scaling.

The measured value at location i, day j, and time-slot k is represented using variable $n_{i,j,k}$. Subscript "_" is used to represent time-series. If one or more subscript (i, j, or k) in $n_{i,j,k}$ is replaced by "_" then the corresponding variable represents the time-series vector containing all the values of the replaced variable. For example, the time-series representing measured sensor data at location i on day j is given as $n_{i,j,\_}$. Here, the variable for the time-slot (k) is replaced by "_" and thus represents the data for every time-slot. The first element of vector $n_{i,j,\_}$ is the sensor data in the first time-slot, the second is the second data in second time-slot and so on. Similarly, $n_{i,\_,\_}$ is one big vector that represents the sensor data in a time-series format at location i for all days and time-slots. To represent the values for a range of time-slots between k and $k+x$ (x is an integer), we use k_k+x in the subscript. Thus, the vector n_{i,j,k_k+x} represents the sensor data between time slots k to $k+x$. Similar notation is used for representing range of days and locations. In this paper, we use symbol "$\times$" to denote scalar multiplication between two numbers or between a number and a vector/matrix. Dot product and element-wise multiplication between two vectors or matrix is represented using symbol "." and "$*$" respectively.

The time-series data ($n_{i,j,\_}$) contains noise. To reduce the effect of noise, the $n_{i,j,\_}$ is passed through the moving average filter. As shown in Fig. 2, the time-series is passed through a noise removal filter. Filtered data is used in clustering as well as in the outlier detection module. For detecting outliers, the first step is to find the cluster of correlated sensors and is described in the following section.

5.1. Clustering module

We reduce time-series vector near every sensor to a low dimensional representation using Principal Component Analysis (PCA). Reducing time-series to a low dimensional representation makes Hierarchical clustering in the subsequent step computationally efficient. Thus, the Hierarchical clustering, instead of using entire time-series vector, uses its low dimensional representation and is computationally efficient.

5.1.1. Principal Component Analysis

For clustering, we use the time-series data for one week from every sensor. The vector representing weekly time-series data is represented using variable v_i, where $v_i = n_{i,1\_7,\_}$. The vector v_i, for every i, is processed using Principal Component Analysis (PCA) and reduced to a low cardinality representation while preserving patterns in the original sample.

PCA is an unsupervised machine learning algorithm that is used for data analysis [29]. It reveals the hidden structure in high dimensional data-sets and provides a low-rank approximate matrix, given the original matrix. Since it is an efficient dimension reduction technique, we use it to extract the dominant components of the time-series vectors.

We use the matrix V, such that $V = \{v_1, v_2, \ldots v_L\}$, for reducing the cardinality of every v_i vector. v_i is a row of V and represents the sample data for one sensor. Since there are L number of sensors, the number of rows in matrix V are L. The number of columns is the cardinality of vector v_i and is represented as D. The primary objective of PCA is to find a d dimensional coordinate system, where $d < D$. The coordinates in the new subspace are orthogonal to each other. Also, they are linear combinations of the sample data and maintains maximum variability in data points. Mathematically, the problem is to find $\hat{V}$ such that $\hat{V} = V.P$, where the dimensions of P are $D \times d$. For the given time-series representation, we determine the different principal components such that the first principal component is one in the direction provided by Eq. 1.

$$p_1 = \underset{||p||=1}{arg\ max}\{\Sigma_i (v_i.p)^2\} \tag{1}$$

The value of p that maximizes the right hand side of the Eq. 1 is the first principal component p_1. In Eq. 1, the term $v_i.p$ is the projection of vector v_i in the direction of unit vector p. The objective is to find the direction p_1 that maximizes the variability of the vectors v_i. Proceeding similarly, if first $d-1$ principal components are determined, the d_{th} principal component is one of the residuals. The residual is obtained by removing the data mapped into the $d-1$ directions. Thus, d_{th} principal component p_d is obtained as given in Eq. 2.

$$p_d = \underset{||p||=1}{arg\ max}||(V - \Sigma_1^{d-1} V.(p_i.p_i^T)).p|| \tag{2}$$

Once the principal orthogonal axes are determined, the transformed data in their direction is obtained using Eq. 3.

$$\hat{V} = V.P \tag{3}$$

It can be proved that the first d principal components of V are the top d eigenvalues in the covariance matrix of V. However, the proof is beyond the scope of this paper and can be found in [29]. For the given data, using PCA, we have reduced the cardinality of vector v_i by 61%.

5.1.2. Hierarchical clustering

The matrix $\hat{V}$ is used to perform Hierarchical clustering and find correlated sensors. Hierarchical clustering is an unsupervised non-parametric clustering method that can be used to group objects according to similar patterns and put them in a cluster [30]. Every row of $\hat{V}$ represents reduced data sample for a sensor. Since the number of sensors are L, the number of rows in $\hat{V}$ is also equal to L. The clustering process for finding similar sensors is explained in Algorithm 1.

Algorithm 1 Hierarchical clustering.

Input: Reduced matrix- $\hat{V}$
Output: Clusters- c
1: Assign each data sample to a single cluster.
2: **repeat**
3: Find the two most similar clusters.
4: Merge clusters obtained in step 2.
5: **until** A single cluster is obtained.

The clustering process begins by assigning every row of $\hat{V}$ to a cluster. Thus, in the beginning, we have as many clusters as the number of sensors L such that every cluster has a single element. In successive steps, the two clusters which are similar to each other are merged to form a single cluster. Thus, as the algorithm progresses, the number of clusters decreases. The merging process continues until a single cluster is formed.

To find the similarity between two clusters, euclidean distance between cluster centroids is used. The two clusters having minimum euclidean distance between their centroids are merged in each step. For a cluster having single element, centroid is the value of multidimensional vector (row of $\hat{V}$). If it has more than one element, its centroid is obtained by taking the average or center point. The average is obtained across each principal directions (columns of $\hat{V}$). Thus, if any two row in $\hat{V}$ are in a cluster then its centroid is obtained as the vector whose elements are the average of values in every column of the corresponding rows.

5.1.3. Dendrogram and number of clusters

Fig. 3 shows the dendrogram plot explaining cluster formation process for a set of vehicular traffic sensors. The lowest end of the plot (leaf) represent single sensors. The vertical lines that are combined to a single point show the two clusters being merged. The vertical axis represents the euclidean distance between the cluster centroids. The difference in height of the vertical lines, that are combined, is the distance between two clusters. If a horizontal line is drawn through the plot, then the number of clusters, before the merge at the particular level, is equal to half the number of times the horizontal line cuts the vertical lines.

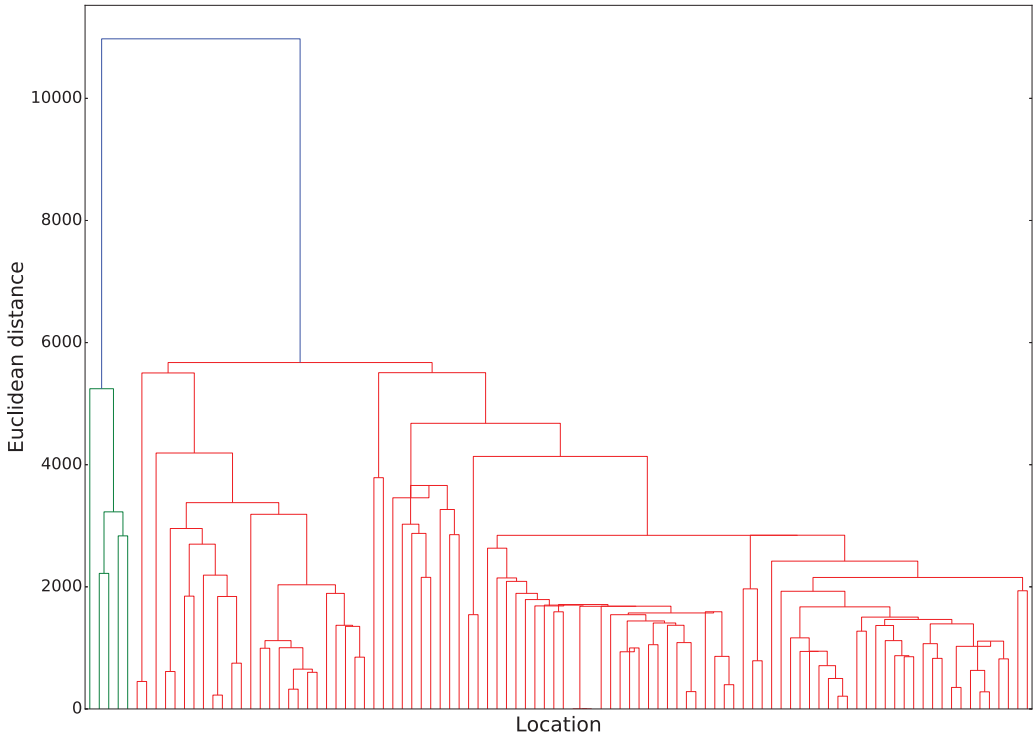

Fig. 3. Dendrogram explaining cluster formation.

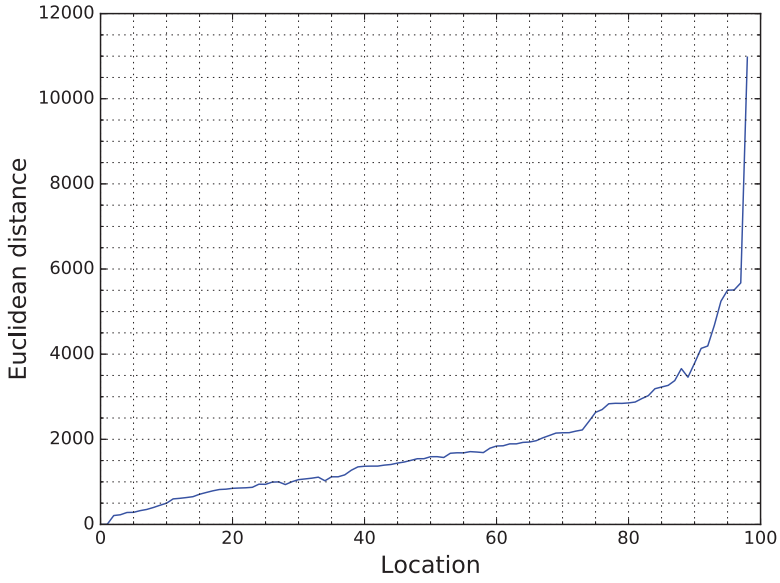

Fig. 4. Euclidean distance between clusters.

The number of clusters are selected such that the distance between the clusters being merged is less than a certain threshold value. If the threshold distance is low (at the lower end of the dendrogram plot), large number of clusters are formed. In this case, the samples within clusters will be very similar to each other. On the other hand, if the threshold distance is significant, the number of clusters will be less, but the samples within the cluster will have high variability. Thus, the number of clusters depends upon the chosen threshold value.

The variation in the euclidean distance between clusters being merged is plotted in Fig. 4. To select the number of clusters, the point where there is an abrupt change in the slope of the graph is chosen as the threshold distance. As we can see from the Fig. 4, the slope of the curve has an edge at three points, where distance approaches 2500, 3500, and 5500.

One of these points can be chosen as the threshold distance to determine the number of clusters. As explained above, for lower threshold distance 2500, the number of clusters will be more with less number of samples in a cluster, but points within a cluster will be close to each other. For higher threshold value (5500) the cluster size will be significant with high variability of sample points within cluster elements.

5.1.4. Cluster center

The cluster center can be obtained using various alternatives, like ward, average, single, complete, or centroid [30]. In this paper, we chose centroid because it was found to provide the highest cophenetic correlation coefficient *cc*. A higher cophenetic correlation coefficient implies that the clusters better represent the original data-set [31].

The cophenetic correlation value is determined as the linear correlation between the original pairwise distance between the sample $\hat{V}$ and the distances obtained from the dendrogram plot. Thus, if Y represents the vector containing pairwise distance of the elements in $\hat{V}$ and Z represents the euclidean distances between clusters where two points are first merged, then the cophenetic correlation coefficient is obtained as in Eq. 4. Here, Y_{st} is the distance between s_{th} and t_{th} original observation in $\hat{V}$ and Z_{st} is the distance between height of the dendrogram plot when s_{th} and t_{th} point are first combined in a cluster. Variables y and z are the average of Y and Z respectively.

$$cc = \frac{\Sigma_{s<t}(Y_{st} - y) \times (Z_{st} - z)}{\sqrt{\Sigma_{s<t}(Y_{st} - y)^2 \times \Sigma_{st}(Z_{st} - z)^2}} \tag{4}$$

5.2. Outlier detection

The obtained clusters are used for detecting outliers in time-series data. The outlier detector is developed for every cluster of sensors. The given model is scalable because an outlier detector is used for many sensors. We describe below the combination of LSTM neural network and M-estimator for detecting outliers in time-series data. The following analysis is for a cluster of sensors. The trained LSTM neural network can be used for any sensor in the particular cluster. Thus, there are as many models as the number of clusters obtained during clustering analysis.

5.2.1. Segmentation

As shown in Fig. 2, for detecting outliers, we first segment time-series. The time-series vectors are split into length of equal intervals Γ. The segments are used for detecting outliers. In a segment, we assume that 50% of the samples near median are true values. Thus, we verify for outliers in lower and upper ends of the segment. For example, if values in a time-series segment are sorted in increasing order of magnitude, then half of the values in the middle portion are considered as true values. The other half (25% at lower and 25% at upper ends) values may contain outliers. A particular segment starting at time-slot k is represented using the variable $s_k = \{n_{i,j,k}, n_{i,j,k+1}, \ldots n_{i,j,k+\Gamma}\}$. For convenience, we omit the subscript i and j in s_k. The segments of time-series are fed to the M-estimator for statistical analysis. Time-series segments are also used by the LSTM neural network for estimating deviations in it. The statistical analysis and estimated deviation are combined to find the outliers.

5.2.2. M-Estimator

The statistical analysis is performed on a time-series segment. We determine the M-estimator value for a segment. M-estimator is the robust statistical measure of a sample data-set [14]. The critical properties of the M-estimator is that it is resilient in the presence of outliers and does not depend on samples having normal distribution. Thus, even if the outliers are present, the M-estimator value is expected to be constant. In contrast, statistical properties like mean is not robust against outliers because the true sample mean changes even in the presence of a single outlier. For a segment, the M-estimator is obtained as the solution of the Eq. 5.

$$\Sigma_{k=k}^{k+\Gamma} \xi\left(\frac{n_{i,j,k} - \mu}{\sigma(s_k)}\right) = 0 \tag{5}$$

The denominator $\sigma(s_k)$ is a function on s_k and gives initial estimate of the solution. The solution μ of the equation is the robust M-estimator μ_k for segment s_k. ξ is a real valued Huber function $\xi(a) = a \times \min(1, \frac{b}{|a|})$, where b is a constant.

The M-estimator is used to find the deviation d_k for a time-series segment. For segment s_k, the deviation from the robust M-estimator μ_k is found using Eq. 6.

$$d_k = \max(|n_{i,j,k} - \mu_k|, |n_{i,j,k+1} - \mu_k|, \ldots |n_{i,j,k+\Gamma} - \mu_k|) \tag{6}$$

We use quantities s_k, μ_k, and d_k for training the LSTM neural network. That is based on the values s_k, μ_k, and d_k, a neural network model is developed that assists in determining outliers in time-series segments. The detailed analysis of LSTM neural network and subsequently the outlier detector is provided in the following sections.

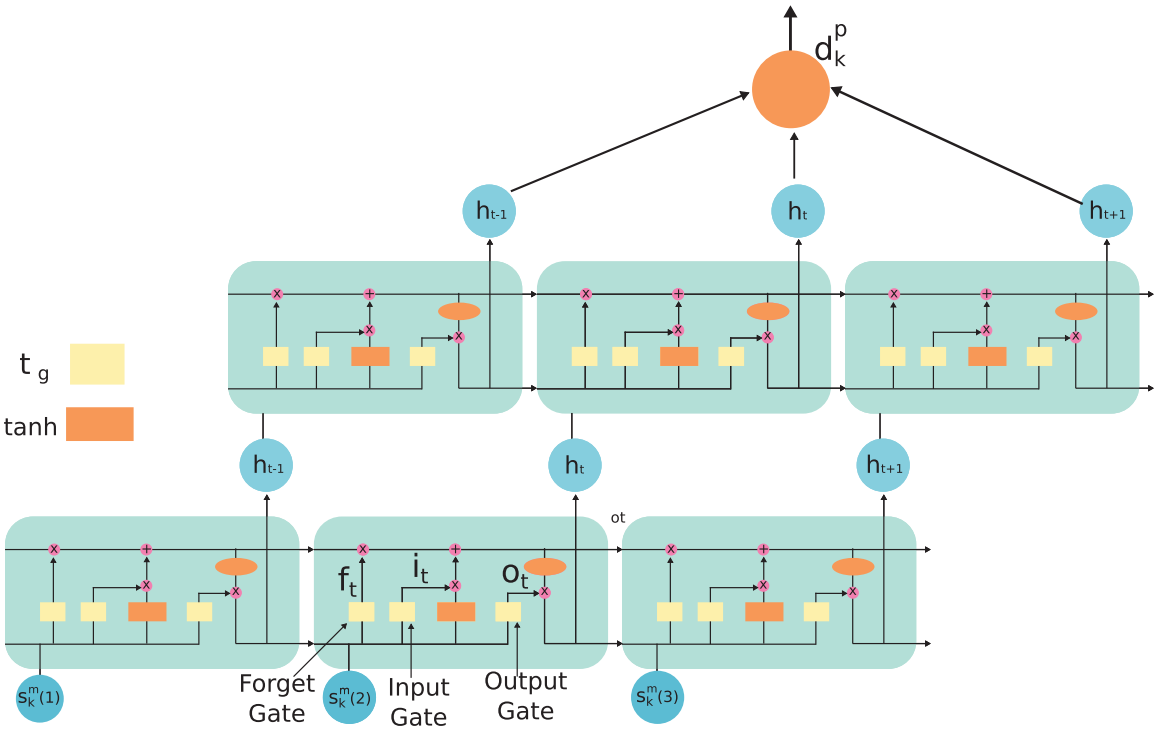

Fig. 5. LSTM neural network.

5.2.3. LSTM neural network

We describe the LSTM neural network that is used for predicting deviation from robust M-estimator. Eq. 6 shows the actual deviation from M-estimator in a time-series segment. However, in the presence of outliers, actual deviation in the segment is different as calculated using Eq. 6 because the sample points are not true values but outliers. Thus, the proposed method predicts the deviation from the M-estimator d_k^p, rather calculating it using Eq. 6.

From segment s_k, we find the vector s_k^m, containing 50% of s_k elements near the median. The elements of the vector s_k^m are represented as $s_k^m(1)$, $s_k^m(2)$,\ldots , and so on. Since the cardinality of s_k is Γ, the vector s_k^m has the cardinality $\Gamma/2$. The prediction problem is, given the input s_k^m, design a LSTM neural network to accurately estimate d_k^p.

The LSTM neural network architecture is shown in Fig. 5. The LSTM neural network has cells that process the inputs fed to it. The cells are sequentially joined to each other to form a chain as shown in the figure. Each such chain is called a layer of the LSTM neural network. The multiple layers are stacked above each other such that output of a layer is the input for layer above it. The cell has a memory called as its state. The LSTM cell processes the data based on an external input, output of the previous cell, and state of the previous cell [32].

The variable t is used to represent a cell. Thus, the cells adjacent to t, in same layer, are $t-1$ and $t+1$. Every cell has forget, input, and output gates. Forget gate determines the information from the previous cell to be discarded in the current cell. The input gate determines the information from the current input to be saved. The output gate determines the amount of information to be passed to the next stage. Each gate receives external input (an element of vector s_k^m) and output from the previous cell. If r is a number between 1 and $\Gamma/2$, gates process the input as given in Eq. 7, to get the outputs, f_t, i_t, and o_t, of forget, input, and output gates respectively, for cell t.

$$f_t = \sigma_g \times (W_f.s_i^m(r) + U_f.h_{t-1} + b_f) \tag{7a}$$

$$i_t = \sigma_g \times (W_i.s_i^m(r) + U_i.h_{t-1} + b_i) \tag{7b}$$

$$o_t = \sigma_g \times (W_o.s_i^m(r) + U_o.h_{t-1} + b_o) \tag{7c}$$

Here, W_f, W_i, and W_o are the weight matrices between external input and the three gates. U_f, U_i, and U_o are the weights between output from cell $t-1$ and the gates. b_f, b_i, and b_o are the bias vectors, h_{t-1} is the output from the cell $t-1$, and σ_g is the sigmoid activation function.

The memory of LSTM cell (cell state) is evaluated using Eq. 8, where W_c and U_c are the weight matrices and b_c is the bias vector. The cell state depends upon current external input $s_i^m(r)$, output of $t-1$ cell h_{t-1}, and state of the $t-1$ cell

C_{t-1}. As depicted in Eq. 8b, the cell t forgets some information from state of the cell $t-1$ (term $f_t * C_{t-1}$) and keeps some information from current input and output of the cell $t-1$ (term $i_t * \tilde{C}_t$), to get the updated state C_t.

$$\tilde{C}_t = \tanh(W_c.s_i^m(r) + U_c.h_{t-1} + b_c) \tag{8a}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8b}$$

The output of a cell depends upon the output gate (o_t) and cell state C_t, is given in Eq. 9.

$$h_t = o_t * \tanh(C_t) \tag{9}$$

The above analysis is for a single cell in a layer (input layer). The outputs of cells in a layer are the external inputs to the cells in layer stacked above it.

5.2.4. LSTM neural network design

In the given prediction problem, the vector s_k^m is the input and d_k^p is the output to the LSTM neural network. The cells in the input layer of LSTM neural network are added based on the cardinality of the vector s_k^m. Since, for an input vector s_k^m, there is a single valued output d_k^p, a neuron is added at the output. The outputs from the uppermost LSTM layer is fed as input to the feed-forward neuron. The output of the neuron is the required predicted deviation d_k^p. Due to data normalization, the output value ranges between 0 to 1. Therefore, the activation function of the output neuron is chosen as tanh, as it is a continuous function and its range include required limits between 0 and 1. Between input layer and output neuron, three LSTM layers are added so that the network accurately predicts the d_k^p value. The activation function for LSTM cells is also chosen as tanh.

Initially, the LSTM neural network operates in training phase to set its weights and biases to optimum values. Once the network is trained, it is used for predicting deviation. The training phase is an offline process that uses the known input vectors s_k^m and output values d_k. The quantities s_k^m and d_k are collected for one week data. During the training phase, the weights and biases associated with the network are adjusted so that the difference between predicted deviation d_k^p and the actual deviation d_k is minimum. Thus, the input s_k^m is fed to the network to obtain the output d_k^p. Then, based on the mean squared error between predicted d_k^p and actual d_k, the weights of the neural network are adjusted. For neural network training, we tested different optimization methods, like Adam, Adagrad, Adadelta, Gradient descent, and RMSprop [33]. Our analysis resulted in using Adam optimization since it converges fast as compared to other methods. For training purpose, all the input vectors are collected in a matrix X such that a row of X is one input vector s_k^m. The network is trained for 10000 iteration. To avoid the over-fitting of the network, we selected a dropout of 20% [34]. Thus, in every iteration, 20% of the weights in each layer are ignored and not updated. Once the weights are optimally updated such that the error between expected and known output is very low, they are saved and represent the proposed trained model for a cluster of sensors.

The trained model is used in prediction phase, when the outlier detector operates. During prediction phase, a vector s_k^m is fed as input to the neural network. The output provides the estimated value of the deviation from the M-estimator d_k^p. The following section describes how the M-estimator and predicted deviation d_k^p are combined for detecting outliers.

5.2.5. Outlier detector

The process of finding outliers using M-estimator and LSTM neural network is described in the Algorithm 2. The algorithm takes time-series segment s_k as input. It uses a hyper-parameter α to determine the expected range of values within which a segment lies. The algorithm employs trained LSTM neural network models. There are as many LSTM neural network models as there are the number of clusters. The algorithm provides the number of positive η_p, negative η_n, and the total number of outliers η_t. Positive outliers η_p have value higher than the expected maximum value of a segment. Negative outliers η_n have value less than the expected minimum value.

In step 1, algorithm determines the M-estimator μ_k for the time-series segment s_k. Here, it should be noted that s_k may contain outliers. Thus, using M-estimator is useful as it is robust against the presence of outliers. Then, step 2 finds the deviation d_k^p using the LSTM neural network model. For determining deviation, out of different LSTM neural network models, one for the cluster in which segment s_k belongs is chosen. The number of outliers is set to zero in step 3.

After that, steps 4-11 iterates for every element of s_k. In step 5, the algorithm checks whether an element is less than the expected minimum value. The expected minimum value is set to $\mu_k - \alpha \times d_k^p$. If the point is less than the expected minimum value, it is considered as a negative outlier. Correspondingly, the negative and total number of outliers are incremented in step 6. Similarly, step 8 verifies if a point is more than the expected maximum value $\mu_k + \alpha \times d_k^p$. If the condition is satisfied, the values of the positive and the total number of outliers are incremented in step 9. Tunable hyper-parameter α is adjusted to get the different range of expected minimum and maximum values.

6. Results

This section describes in detail the evaluation results of the proposed method. We test the performance of the outlier detector using two different data-sets; (1) vehicular traffic count and (2) environmental pollutant Carbon Mono-oxide (CO) level.

Algorithm 2 Outlier detector algorithm.

Input: Time-series segment- s_k
Hyper-parameter- α_k
Trained LSTM neural network models
Output: Number of outliers- η_p, η_n, η_t

$\mu_k = Find\_Mestimator(s_k)$
$d_k^p = Find\_Deviation(s_k^m)$
$\eta_p,\ \eta_n,\ \eta_t = 0$
for $r \leftarrow 1$ **to** Γ **do**
  if $s_k(r) < \mu_k - \alpha \times d_k^p$ **then**
    $\eta_n = \eta_n + 1,\ \eta = \eta + 1$
  end if
  if $s_k(r) > \mu_k + \alpha \times d_k^p$ **then**
    $\eta_p = \eta_p + 1,\ \eta = \eta + 1$
  end if
end for

6.1. Materials and methods

The vehicular traffic data used for testing the given method is collected from the Performance Measurement System (PeMs). PeMs provides freeway traffic at California Highways [28]. The pollutant level is obtained from the California Air Resource Board [35] website. We implemented the simulation atmosphere in a Python-based framework. In our framework, different python tools and libraries are employed. The clustering module is implemented using Python-Scikit tool [36]. The LSTM neural network is developed using Keras framework running on top of the TensorFlow environment [33]. The simulation is executed in Ubuntu 16.04 operating system running on Intel Core-i5 processor at 2.60 GHz and containing four cores.

We simulate outliers using the mean of the standard deviations in the training data segments. The average of the standard deviations in all s_k vectors is obtained for training sample. The outliers proportional to the obtained average are injected in the test data.

6.2. Evaluation metrics

The efficacy of the proposed method is evaluated using precision P, recall R, and F-measure F. The precision is the ratio of true positives tp and the sum of true positives tp and false positives fp (Eq. 10).

$$P = \frac{tp}{tp + fp} \tag{10}$$

The recall R is the ratio of true positives tp and the sum of true positives tp and false negatives fn and is calculated using Eq. 11.

$$R = \frac{tp}{tp + fn} \tag{11}$$

The F-measure is obtained using Precision and Recall values as given by Eq. 12.

$$F = 2 \times \frac{P \times R}{P + R} \tag{12}$$

To find F-measure, we put equal weight to precision and recall values. The generalized form of F-measure is given in Eq. 13, where the parameter $\beta \geq 0$. The F-measure is said to be recall-oriented, if $\beta < 1$, and precision oriented, if $\beta > 1$ [37].

$$F_g = (1 + \beta^2) \times \frac{P \times R}{\beta^2 \times P + R} \tag{13}$$

6.3. Performance evaluation

We evaluate performance as the magnitude of outliers increases for both the data-sets. In Tables 1 and 2, the first column represents the strength of the injected outliers. For example, *mean_std_*.5 is the actual value of data plus 0.5 times the mean of recorded standard deviation in the training data.

Table 1 depicts the performance measure in the presence of positive outliers. From the table, we can observe that as the magnitude increases, the performance of the outlier detector improves for both vehicular traffic and pollution level data. It can be observed that the recall values are close to 1 in most of the rows. It led to conclude that the number of false negatives are negligible. The precision values are also high for different magnitudes. The maximum precision value for

Table 1
Analysis for positive outlier.

Strength	Vehicular traffic			Air pollutant		
	P	*R*	*F*	*P*	*R*	*F*
mean_std_.5	0.89	0.55	0.68	0.95	0.7	0.81
mean_std_2	0.91	0.72	0.8	0.96	0.86	0.91
mean_std_3	0.93	0.89	0.91	0.97	0.99	0.98
mean_std_4	0.93	0.96	0.94	0.97	1.0	0.98
mean_std_5	0.93	0.99	0.96	0.97	1.0	0.98
mean_std_6	0.93	1	0.96	0.97	1.0	0.98
mean_std_7	0.93	1	0.96	0.97	1.0	0.98
mean_std_8	0.93	1	0.96	0.97	1.0	0.98
mean_std_9	0.93	1	0.96	0.97	1.0	0.98
mean_std_10	0.93	1	0.96	0.97	1.0	0.98
mean_std_11	0.93	1	0.96	0.97	1.0	0.98
mean_std_12	0.93	1	0.96	0.97	1.0	0.98

Table 2
Analysis for negative outlier.

Strength	Vehicular traffic			Air pollutant		
	P	*R*	*F*	*P*	*R*	*F*
mean_std_.5	0.95	0.5	0.66	0.86	0.22	0.35
mean_std_2	0.97	0.72	0.83	0.97	0.29	0.45
mean_std_3	0.99	0.9	0.94	0.99	0.76	0.86
mean_std_4	0.98	0.95	0.96	0.99	1.0	0.99
mean_std_5	0.97	0.99	0.98	0.98	1.0	0.99
mean_std_6	0.98	0.98	0.98	0.97	1.0	0.98
mean_std_7	0.99	0.98	0.98	0.97	1.0	0.98
mean_std_8	0.98	0.99	0.98	0.97	1.0	0.98
mean_std_9	0.99	0.99	0.99	0.96	1.0	0.98
mean_std_10	0.99	1	0.99	0.96	1.0	0.98
mean_std_11	0.98	1	0.99	0.97	1.0	0.98
mean_std_12	0.98	1	0.99	0.97	1.0	0.98

Table 3
Analysis as percentage of outliers vary.

% outliers	Vehicular traffic			Air pollutant		
	P	*R*	*F*	*P*	*R*	*F*
8.3%	0.88	0.97	0.92	0.85	1	0.92
16.7%	0.98	0.96	0.97	0.93	0.97	0.95
25%	0.99	0.96	0.97	0.96	0.95	0.95
33.3%	1	0.96	0.98	1	0.97	0.98
41.7%	1	0.95	0.97	0.99	0.95	0.97
50%	1	0.96	0.98	1	0.98	0.99

vehicular traffic is 93% while for air pollutant is 97%. The F-measure approaches 96% for vehicular traffic and 98% for air pollutant data. Thus, the metrics show that the given method accurately detects the outliers for different data-sets and has a negligible number of false negatives.

Table 2 presents the performance analysis for negative outliers. The algorithm effectively captures the negative outliers too in the sample. The precision values approach 95% for vehicular traffic count and 99% for air pollutant data. For high injected magnitude, the recall values become 100% for both the data-sets. The corresponding F-measure also approaches to 98% for traffic and pollutant data. Thus, for negative outliers also the performance measure in terms of precision, recall, and F-measure values are high.

Further, we observe the performance when both positive and negative outliers are present. For this case, the percentage of outliers is varied, and the metrics values are given in Table 3. We observe that the performance in terms of the three metrics improve as the number of outliers increases. For example, the precision approaches to around 1 for both data-sets.

Thus, the Tables 1–3 concludes that the given method accurately detects outliers, has less number of false positives, and as the percentage of outliers increases accuracy increases.

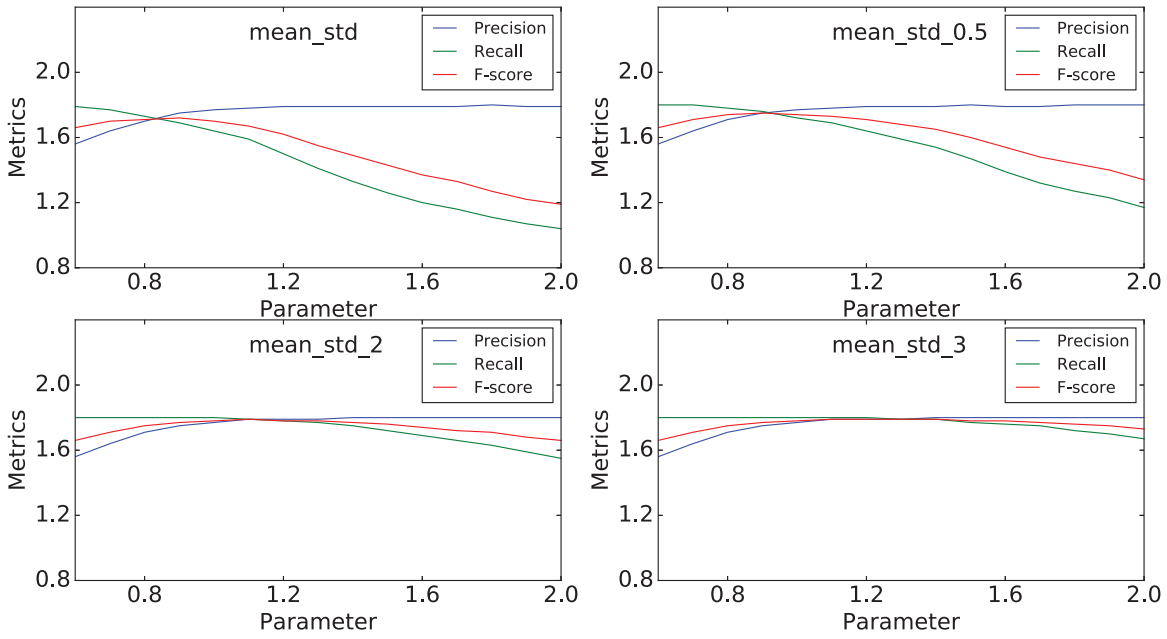

Fig. 6. Comparison of performance metrics as α varies.

6.4. Performance analysis as parameter α is tuned

Fig. 6 shows the variation in performance metrics as the parameter α in Algorithm 2 is tuned. We plot the graphs for 4 different strength of injected outliers. The figure portrays that for a low value of α, recall is better than the precision. However, as the value of α increases, the precision becomes higher than the recall. Also, as the magnitude of outliers increases, the plots become parallel to the horizontal axis, and the performance of the outlier detector improves. Thus, the value of α, where the three metrics are equal to each other, increases on increasing the injection magnitude.

The figure infers that for an application requiring low false negatives, the α should be set to a low value. On the other hand, applications requiring low false positives, α can set to a high value. Thus, the parameter α can be tuned, satisfying the need of an application. Further, for effectively detecting outliers of low strength, α can be tuned to a small value. However, for detecting outliers having high strength, α can be set to a high value.

6.5. Performance analysis as cluster size varies

Fig. 7 shows the performance variation as the threshold euclidean distance increases for different magnitudes of outlier strength. As we presented earlier in Section 5.1.3, on increasing the euclidean distance, the number of sensors in a cluster increases and thus the system scales up. Figure interprets that the accuracy decreases as the threshold euclidean distance (cluster size) increases. For small cluster size, the accuracy is greater as the model can better represent less number of sensors within a cluster. For large cluster size, the sensor diversity within a cluster increases, and due to this, the accuracy decreases. Furthermore, for high strength of injected outliers, there is a smaller decrease in accuracy as euclidean distance increases. This is because those outliers tend to be easily detected.

Fig. 8 shows the number of models required to be developed as euclidean distance increases. For low euclidean distance, the number of clusters is large, and thus the number of required models are large. Therefore, computational effort is high. For high euclidean distance, the number of clusters is small, and thus the less number of models are required, and the computational effort is low.

Thus, Figs. 7 and 8 show that as the threshold euclidean distance or cluster size increases, the accuracy decreases, and the number of models to be developed becomes low, requiring less computational efforts. Therefore, there is a trade-off between performance and computational efforts.

6.6. Comparison with other approaches

We compare the proposed method with statistical-based techniques. The Generalized Extreme Studentized Deviate (GESD) test is used as a benchmark for the performance comparison [13]. Traditionally, GESD uses the mean and standard deviation of the sample as a statistical measure. The median and median absolute deviation (MAD) have also been

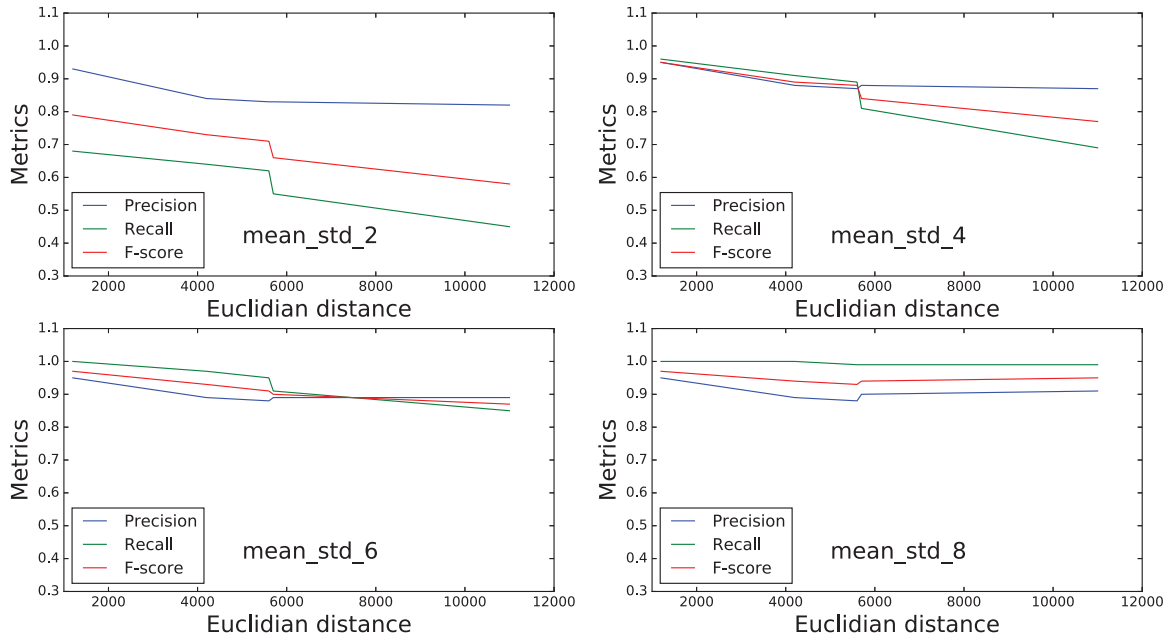

Fig. 7. Performance metrics as euclidian distance varies.

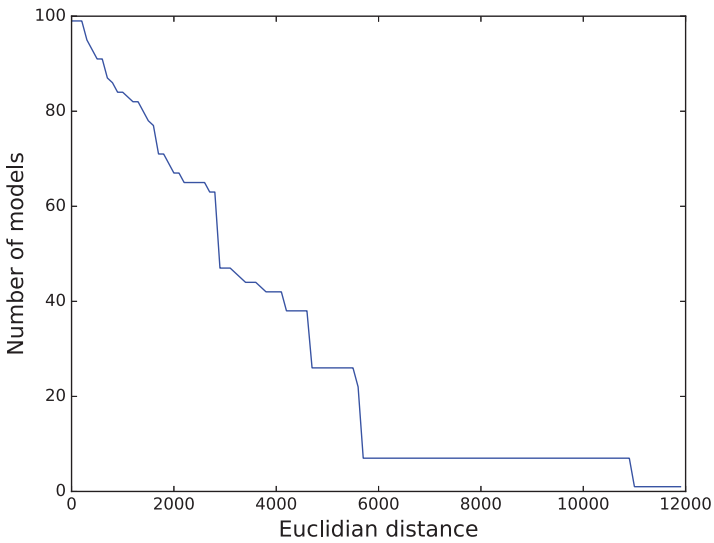

Fig. 8. Number of models as euclidian distance is varied.

used in GESD test [22]. We evaluate the proposed method against the above methods such that using mean (and standard deviation) and median (and MAD) in GESD. Furthermore, we also test the performance using M-estimator and deviation from it in GESD. The comparison for the different metrics is provided in Figs. 9–11 for vehicular data and in Figs. 12–14 for pollutant data.

Figs. 9 and 12 show that the proposed LSTM neural network and M-estimator based method outperforms the GESD-based techniques irrespective of the statistical measure chosen. For GESD, using mean and standard deviation as statistical properties provides the highest precision values, that is lower than the proposed method. It should be noted that, by tuning α, the precision can be further improved, but at the cost of the recall. The GESD-based methods do not have such tunable parameter, and thus, the obtained precision values are fixed.

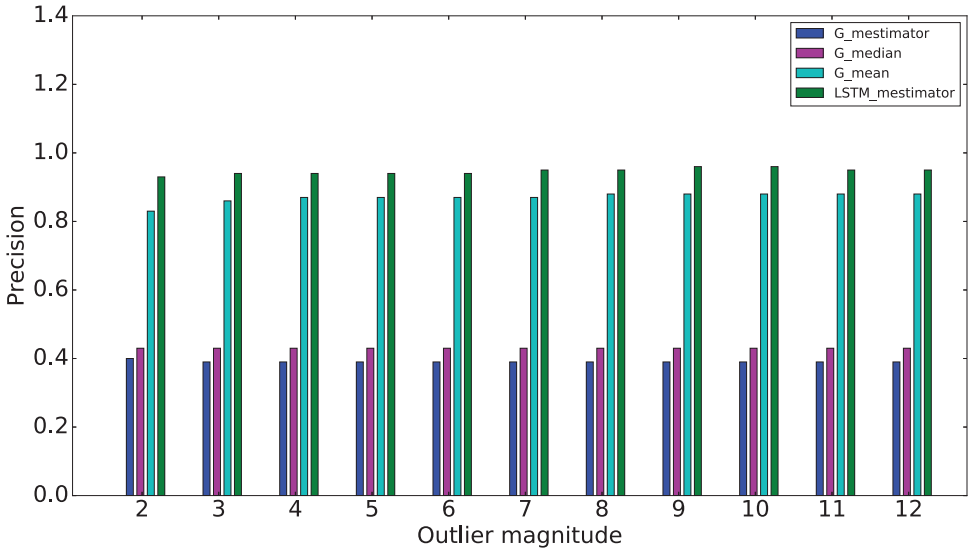

Fig. 9. Precision comparison for traffic data.

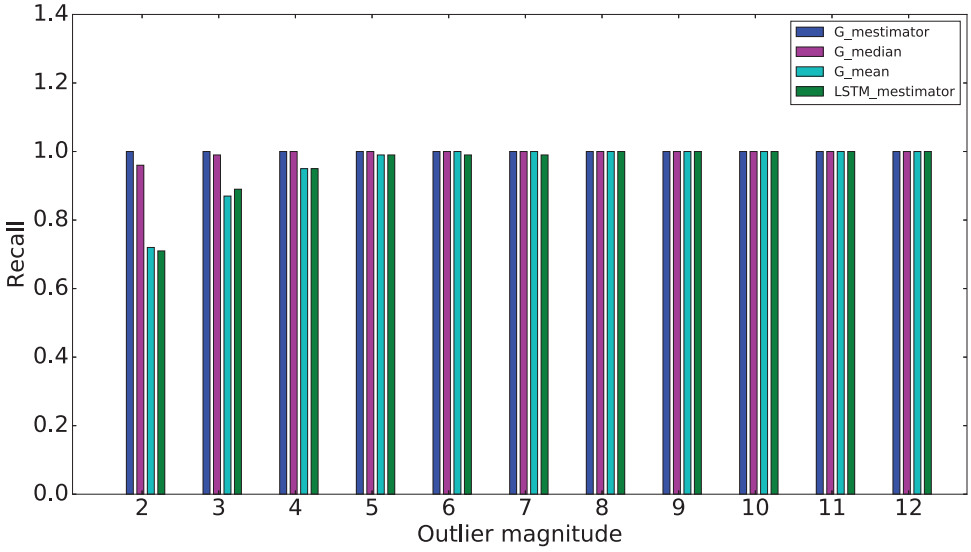

Fig. 10. Recall comparison for traffic data.

Figs. 10 and 13 compares the recall values for both the data-sets. As shown in the figure, for the low strength of the outliers, using GESD with M-estimator and deviation from it provides the highest recall. For high strength, the recall is comparable for different methods. However, as shown in Figs. 9–14, precision and F-measures are lowest when using GESD with M-estimators. Thus, although the GESD with M-estimator and deviation from it provides comparable recall, it is not a viable option since the performance of the other two metrics deteriorates significantly.

Figs. 11 and 14 compares the F-measure of the GESD-based techniques and the proposed method. The figures show that regarding F-measure, the given method outperforms different GESD-based techniques. Therefore, as presented, the proposed outlier detector has better performance as compared to the statistical-based techniques in terms of precision, recall, and F-measures. Also, the proposed method is scalable and can be tailored according to the application requirements. The flexibility is not available in the statistical-based techniques, and thus the given architecture is better than contemporary methods.

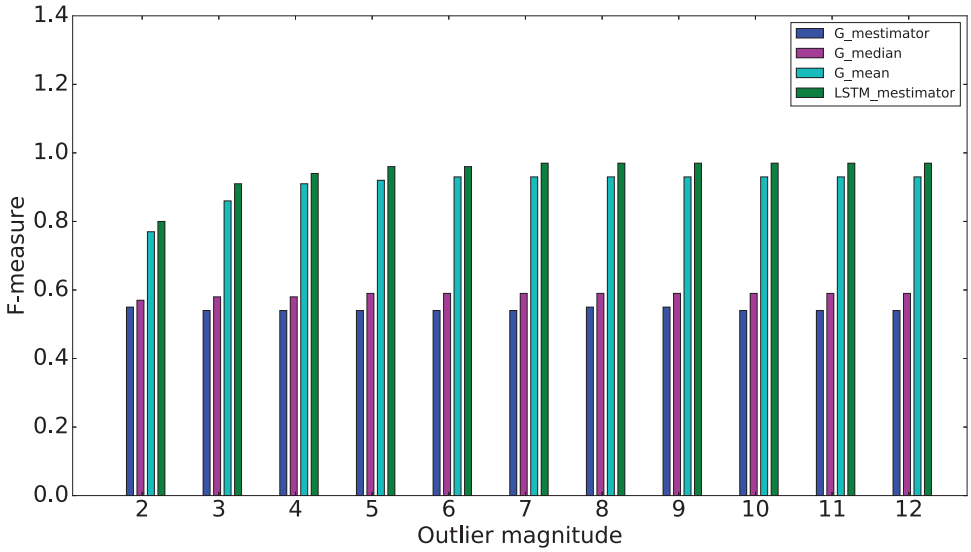

Fig. 11. F-measure comparison for traffic data.

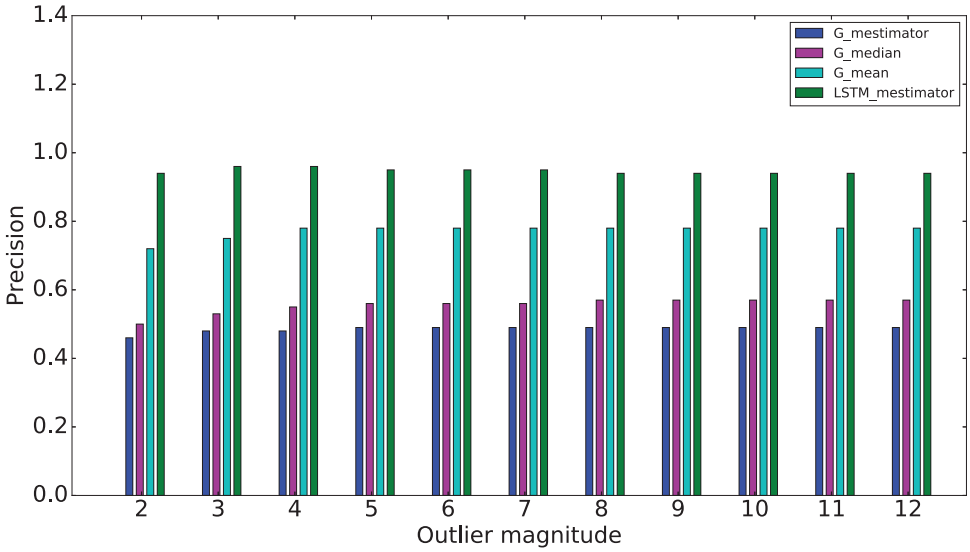

Fig. 12. Precision comparison for pollutant data.

6.7. Computation complexity

In the given outlier detector, first, we train the LSTM neural network, and then it is used for the operation. While training phase is computation-intensive and takes a longer time, it is performed only once. The average time the outlier detector takes, during the operating phase, to perform a test on a segment is 0.18 seconds. Thus, operating phase still takes a pretty low execution time. Furthermore, we tested the performance using limited computing resources. The deployment of the algorithm in the cloud is expected to further reduce execution time because of the presence of a large pool of computing resources.

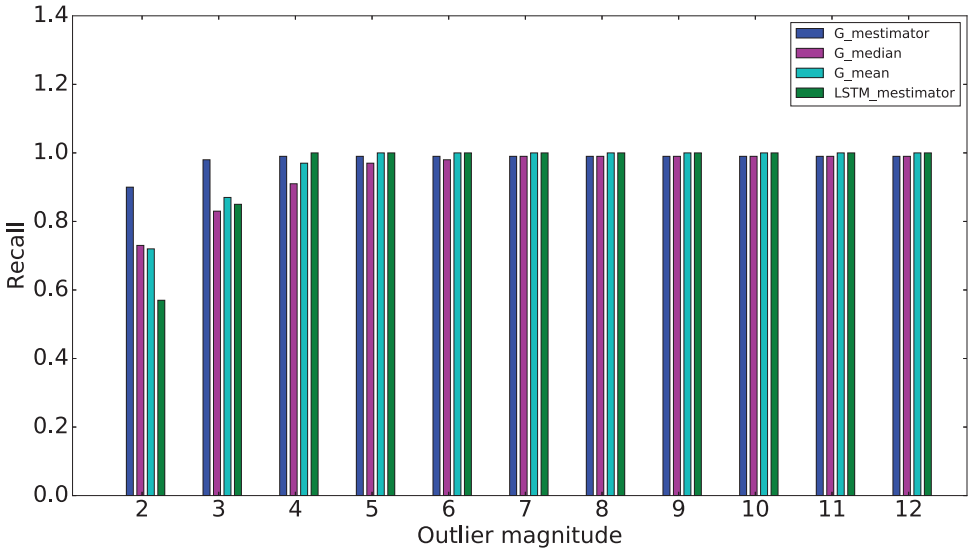

Fig. 13. Recall comparison for pollutant data.

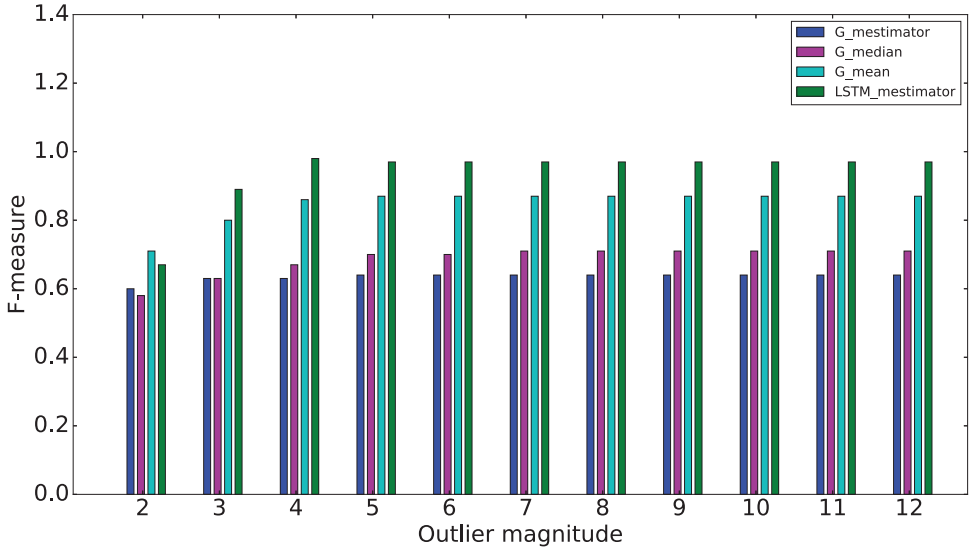

Fig. 14. F-measure comparison for pollutant data.

7. Conclusions and future work

This paper has highlighted the importance of a scalable outlier detector for IoT-based automated applications. We presented an outlier detector that not only accurately determines the anomalies but is also scalable. The performance of the proposed architecture is tested on two different data-sets. The obtained results show that our method has high precision, recall, and F-measure values. The given outlier detector can be tuned according to the application's performance requirement, and either precision or recall values can be further improved. The performance is also tested as the number of sensors in a cluster increases. There is a trade-off between performance and computational efforts as the cluster size increases. The proposed method outperforms the contemporary methods that use statistical properties of the time-series sample.

In future, we aim to explore the different problems related to this field. Our plan is to research out the outlier detector when abnormal values are present in entire time-series, rather than its segment. In this regard, our idea is to compare between different diverse time-series. Subsequently, we plan to study the problem of classifying outliers originating from different sources; variation in environmental parameters, compromised sensors, or faults in the sensor. Furthermore, in IoT the softwares are open-source because it assists in easy to modify or configure them. Outlier detector is no exception and

the adversary may know about the detection mechanism. Therefore, analyzing the problem when adversary manipulates data to bypass existing outlier detection technique is also need to be investigated.

Declaration of Competing Interest

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or revising it critically for important intellectual content; and (c) approval of the final version.

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

Acknowledgments

This research is supported by NSF award #1723814.

References

[1] R.M. Shukla, S. Sengupta, M. Chatterjee, Software-defined network and cloud-edge collaboration for smart and connected vehicles, in: Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking, in: Workshops ICDCN '18, 2018, pp. 6:1–6:6.
[2] R.M. Shukla, S. Sengupta, A.N. Patra, Smart plug-in electric vehicle charging to reduce electric load variation at a parking place, in: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018, pp. 632–638.
[3] R.M. Shukla, P. Kansakar, A. Munir, A neural network-based appliance scheduling methodology for smart homes and buildings with multiple power sources, in: Proceedings of the IEEE International Symposium on Nanoelectronic and Information Systems (iNIS), 2016, pp. 166–171.
[4] R.M. Shukla, S. Sengupta, Cop: An integrated communication, optimization, and prediction unit for smart plug-in electric vehicle charging, Internet of Things 9 (2020) 100148, doi:10.1016/j.iot.2019.100148. http://www.sciencedirect.com/science/article/pii/S2542660519301398.
[5] Iot based smart traffic signal monitoring using vehicle count, https://www.embedded-computing.com/guest-blogs/iot-based-smart-traffic-signal-monitoring-using-vehicle-count.
[6] Apache iotdb: Time series database for industrial iot, https://aceu19.apachecon.com/session/apache-iotdb-time-series-database-industrial-iot.
[7] R. Shukla, S.Sengupta, Analysis and detection of anomaly due to data falsification attacks on traffic prediction applications, in: The 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (IEEE UEMCON 2018), 2018.
[8] W. Jia, R.M. Shukla, S. Sengupta, Anomaly detection using supervised learning and multiple statistical methods, in: Special Topics on Machine and Deep Learning in Cyber Security and Privacy Issues, IEEE International Conference on Machine Learning and Applications (ICMLA), 2019.
[9] S. Bhattacharjee, A. Thakur, S.K. Das, Towards fast and semi-supervised identification of smart meters launching data falsification attacks, in: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, in: ASIACCS '18, 2018, pp. 173–185.
[10] B. Kailkhura, S. Brahma, P.K. Varshney, Data falsification attacks on consensus-based detection systems, IEEE Transactions on Signal and Information Processing over Networks 3 (1) (2017) 145–158.
[11] Mobile application performance testing, http://media.shunra.com/whitepapers/MobilePerfTesting_27913.pdf.
[12] N.A. James, A. Kejariwal, D.S. Matteson, Leveraging cloud data to mitigate user experience from breaking bad, in: 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 3499–3508, doi:10.1109/BigData.2016.7841013.
[13] Generalized esd test for outliers, https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h3.html.
[14] P.J. Rousseeuw, M. Hubert, Anomaly detection by robust statistics, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (2) (2018) e1236.
[15] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, in: Proceedings, Presses universitaires de Louvain, 2015, p. 89.
[16] N.K. Thanigaivelan, E. Nigussie, S. Virtanen, J. Isoaho, Hybrid internal anomaly detection system for iot: Reactive nodes with cross-layer operation, Security and Communication Networks 2018 (2018).
[17] W. Li, H. Song, F. Zeng, Policy-based secure and trustworthy sensing for internet of things in smart cities, IEEE Internet of Things Journal 5 (2) (2018) 716–723, doi:10.1109/JIOT.2017.2720635.
[18] G. Shafer, A mathematical theory of evidence, 42, Princeton university press, 1976.
[19] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, Lof: Identifying density-based local outliers, SIGMOD Rec. 29 (2) (2000) 93–104.
[20] S. Su, L. Xiao, L. Ruan, F. Gu, S. Li, Z. Wang, R. Xu, An efficient density-based local outlier detection approach for scattered data, IEEE Access 7 (2019) 1006–1020, doi:10.1109/ACCESS.2018.2886197.
[21] N. Pandeeswari, G. Kumar, Anomaly detection system in cloud environment using fuzzy clustering based ann, Mobile Networks and Applications 21 (3) (2016) 494–505.
[22] O. Vallis, J. Hochenbaum, A. Kejariwal, A novel technique for long-term anomaly detection in the cloud., in: HotCloud, 2014.
[23] https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68.
[24] H.K. Kanagala, V.V. Jaya Rama Krishnaiah, A comparative study of k-means, dbscan and optics, in: 2016 International Conference on Computer Communication and Informatics (ICCCI), 2016, pp. 1–6, doi:10.1109/ICCCI.2016.7479923.
[25] X. Jin, J. Han, Expectation Maximization Clustering, Springer US, Boston, MA, pp. 382–383.
[26] X. Zhang, Support Vector Machines, Springer US, Boston, MA, pp. 941–946.
[27] A.K. Jain, J. Mao, K. Mohiuddin, Artificial neural networks: A tutorial, Computer (3) (1996) 31–44.
[28] California department of transportation, http://pems.dot.ca.gov/.
[29] I. Jolliffe, Principal Component Analysis, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1094–1096.
[30] F. Murtagh, A survey of recent advances in hierarchical clustering algorithms, The computer journal 26 (4) (1983) 354–359.
[31] J.S. Farris, On the cophenetic correlation coefficient, Systematic Zoology 18 (3) (1969) 279–285.
[32] X. Ma, Z. Tao, Y. Wang, H. Yu, Y. Wang, Long short-term memory neural network for traffic speed prediction using remote microwave sensor data, Transportation Research Part C: Emerging Technologies 54 (2015) 187–197.
[33] Keras documentation. https://keras.io/optimizers/.
[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.
[35] California air resource board. https://ww3.arb.ca.gov/html/ds.htm.
[36] Scikit-learn, https://scikit-learn.org/stable/.
[37] O. Vallis, J. Hochenbaum, A. Kejariwal, A novel technique for long-term anomaly detection in the cloud, in: 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14), USENIX Association, Philadelphia, PA, 2014. https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/vallis.