Anomaly Detection On Sensor Data

By:
Ahmed Eltabakh
Osama Ayman
Amr Moneer

Supervised By:
Eng. Bassem Boshra

Jan 2022

Introduction:

Through increasingly powerful hardware, The computational resources of IoT-Gateways are growing, enabling more sophisticated data analytics, service deployments and virtualisation capabilities. However, the difficulty of ensuring high reliability of IoT-Gateways and its deployed services increases along with the complexity of digital systems. Reliable operation of such gateways and devices therefore depends on automated methods for detecting abnormal behavior as early as possible, in order to remediate the situation before the complete failure of a system component.

There are Supervised approaches for anomaly detection that use labelled system information to train the machine learning models. Put briefly, the anomaly detection algorithms employ either classification or regression models trained using data containing the information whether the data point is anomalous or not. But all these traditional algorithms are offline that require full labelled dataset from the business domain which is actually very difficult practically. So, these models require manual retraining with time if the distribution of the data changed, In addition they can not be adaptive with time.

We propose an unsupervised anomaly detection algorithm using the concept of Half-Space Trees through to get an online adaptive algorithm. The anomaly detector consumes multidimensional time series data from an external monitoring component and is fixed in computational complexity to support real-time operation.

This algorithm will satisfy our needs to become an online, adaptive algorithm than makes inference & prediction in real time once the data point reaches the Master Of Things platform and visualize these results for user.

We implemented this algorithm using Javascript on Master of Things IOT platform as a custom event to perform our online adaptive anomaly detection algorithm using actual sensors on Master Of Things IOT Kit.

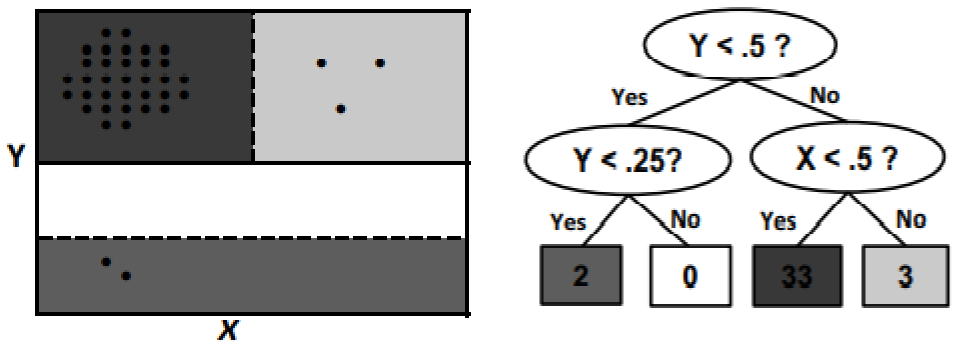
Proposed Method:

The proposed method is an ensemble of HS-Trees. Each HSTree consists of a set of nodes, where each node captures the number of data items (a.k.a. mass) within a particular subspace of the data stream. Mass is used to profile the degree of anomaly because it is simple and fast to compute in comparison to distance-based or density-based methods.

To facilitate learning of mass profiles in evolving data streams, the algorithm segments the stream into windows of equal size (where each window contains a fixed number of data items). The system operates with two consecutive windows, the reference window, followed by the latest window. During the initial stage of the anomaly detection process, the algorithm learns the mass profile of data in the reference window. Then, the learned profile is used to infer the anomaly scores of new data subsequently arriving in the latest window—new data that fall in high-mass subspaces is construed as normal, whereas data in low-mass or empty sub spaces is interpreted as anomalous. As new data arrive at the latest window, the new mass profile is also recorded. When the latest window is full, the newly recorded profile is used to override the old profile in the reference window; thus the reference window will always store the latest profile that can be used to score the next batch of newly arriving data. Once this is done, the latest window erases its stored profile and get ready to capture profile of the next batch of newly arriving data. This process continues as long as the stream exists.

An HS-Tree of depth h is a full binary tree consisting of 2h+1 − 1 nodes, in which all leaves are at the same depth, h. When constructing a tree, the algorithm expands each node by picking a randomly selected dimension, q, in the work space (to be described later in this section) associated with the node. Using the mid-point of q, the algorithm bisects the work space into two half-spaces, thus creating the left child and right child of the node. Node expansion continues until the maximum depth (i.e., h or maxDepth) of all nodes is reached.

Each node records the mass profile of data in a work space that it represents, and has the following elements: (i) arrays min and max, which respectively store the minimum and maximum values of each dimension of the work space rep resented by the node; (ii) variables r and l, which record the mass profiles of data stream captured in the reference window and latest window, respectively; (iii) variable k, which records the depth of the current node; and (iv) two nodes representing the left child and right child of the current node, each associated with a half-space after the split. Figure 1 depicts an example window of (two-dimensional) data that is partitioned by a simple HS-Tree.



Pseudo Code:

Here We present the pseudo code for the main functions of our algorithm:

First we Initialise Work Space, right before the construction of each tree. Assume that attributes' ranges are normalised to [0, 1] at the outset. Let sq be a real number randomly and uniformly generated from the interval [0, 1]. A work range, $s_q \pm 2 \cdot \max(s_q, 1 - s_q)$, is defined for every dimension q in the feature space D. This produces a work space that is a random perturbation of the original feature space. Since each HS-Tree is built from a different work space (defined as min and max in Algorithm 1), the result is an ensemble of diverse HS-Trees.

Algorithm 1 shows the procedure for building a single HS Tree. Each internal node is formed by randomly selecting a dimension q (Line 4) to form two half-spaces; the split point is the mid-point of the current range of q. The mass variables of each node, r and l, are initialised to zero during the tree construction process.

Algorithm 1 : BuildSingleHS-Tree(min, max, k)

Inputs: min & max - arrays of minimum and maximum values for every dimension in a Work Space,
k - current depth level
Output: an HS-Tree

```
1:  if $k == maxDepth$ then
2:      return $Node(r \leftarrow 0, l \leftarrow 0)$ {External node}
3:  else
4:      randomly select a dimension $q$
5:      $p \leftarrow (max_q + min_q)/2$
6:      {Build two nodes ($Left$ & $Right$) from a split into
        two equal-volume half-spaces.}
7:      $temp \leftarrow max_q$; $max_q \leftarrow p$
8:      $Left \leftarrow$ BuildSingleHS-Tree($min, max, k + 1$)
9:      $max_q \leftarrow temp$; $min_q \leftarrow p$
10:     $Right \leftarrow$ BuildSingleHS-Tree($min, max, k + 1$)
11:     return $Node(Left, Right, SplitAtt \leftarrow q,$
        $SplitValue \leftarrow p, r \leftarrow 0, l \leftarrow 0)$
12: end if
```

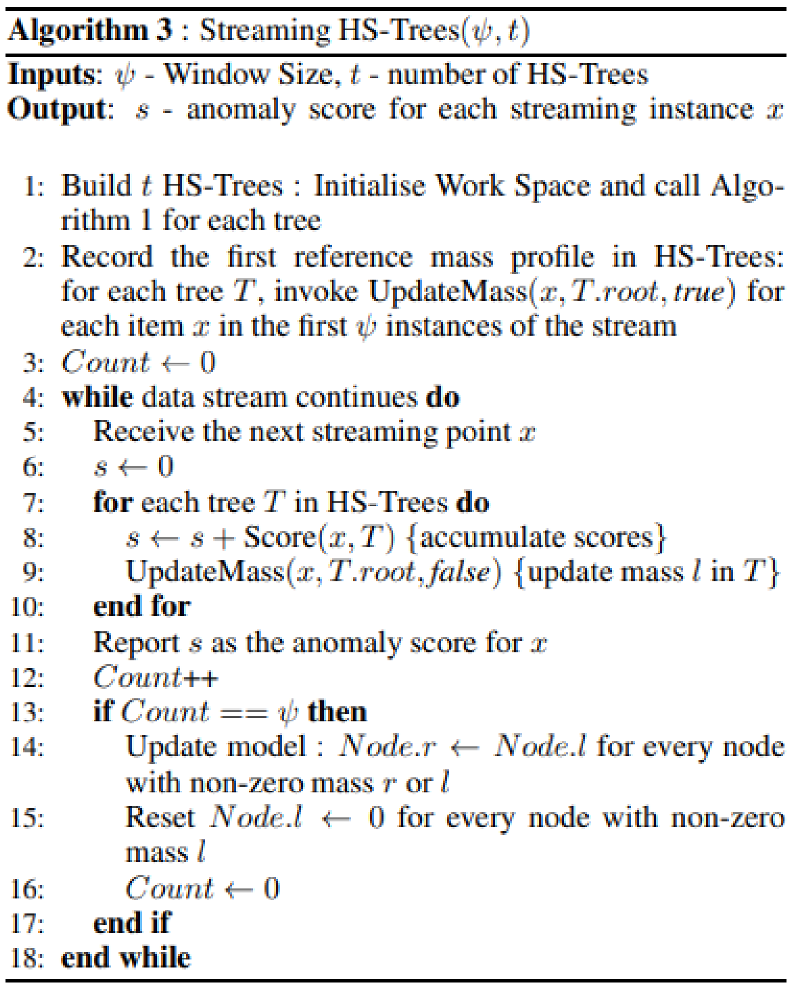
Algorithm 2 shows the process where instances in the reference window will update mass r; and mass l is updated using instances in the latest window. These two collections of mass values at each node, r and l, represent the data profiles in the two different windows.

Algorithm 2 : UpdateMass($x, Node,$ *referenceWindow*)

Inputs: x - an instance, $Node$ - a node in an HS-Tree
Output: none

```
1: (referenceWindow)? $Node.r$++ : $Node.l$++
2: if ($Node.k < maxDepth$) then
3:     Let $Node'$ be the next level of $Node$ that $x$ traverses
4:     UpdateMass($x, Node'$, referenceWindow)
5: end if
```

Algorithm 3 shows the operational procedure for Streaming HS-Trees. Line 1 builds an ensemble of Half-Space Trees. Line 2 uses the first ψ instances of the stream to record its initial reference mass profile in the HS-Trees. Since these in stances come from the initial reference window, only mass r of each traversed node is updated. After these two steps, the model is ready to provide an anomaly score for each subsequent streaming point. Mass r is used to compute the anomaly score for each streaming point (Line 8). The recording of mass for each subsequent streaming point in the latest window is then carried out on mass l (Line 9). At the end of each window, the model is updated. The model update procedure is simple before the start of the next window, the model is updated to the latest mass by simply transferring the non-zero mass l to r (Line 14). This process is fast because it involves no structural change of the model. After this, each node with a non-zero mass l is reset to zero.



Algorithm 3 : Streaming HS-Trees(ψ, t)

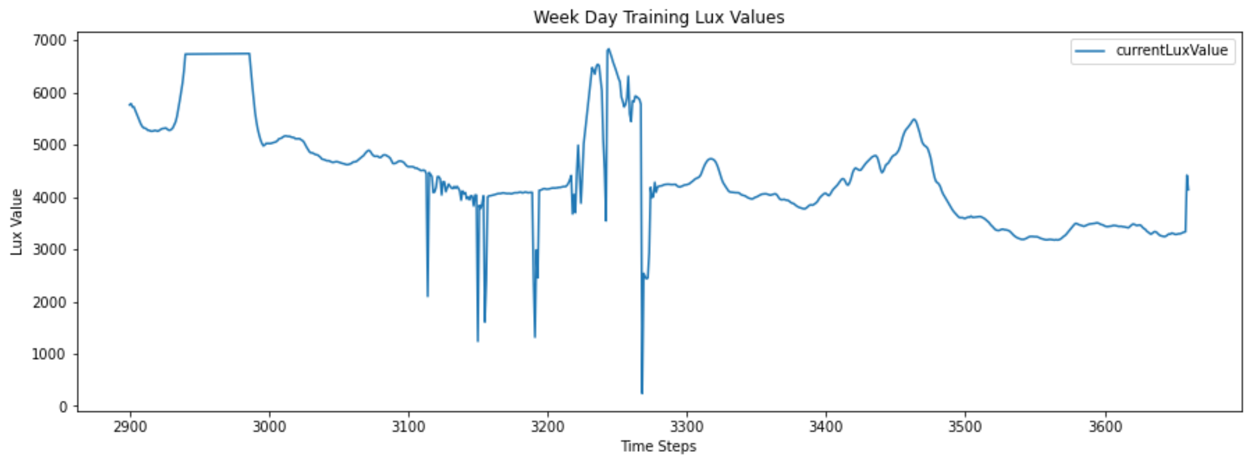
Inputs: ψ - Window Size, t - number of HS-Trees
Output: s - anomaly score for each streaming instance x

```
 1: Build $t$ HS-Trees : Initialise Work Space and call Algo-
    rithm 1 for each tree
 2: Record the first reference mass profile in HS-Trees:
    for each tree $T$, invoke UpdateMass($x, T.root, true$) for
    each item $x$ in the first $\psi$ instances of the stream
 3: $Count \leftarrow 0$
 4: while data stream continues do
 5:   Receive the next streaming point $x$
 6:   $s \leftarrow 0$
 7:   for each tree $T$ in HS-Trees do
 8:     $s \leftarrow s +$ Score($x, T$) {accumulate scores}
 9:     UpdateMass($x, T.root, false$) {update mass $l$ in $T$}
10:   end for
11:   Report $s$ as the anomaly score for $x$
12:   $Count$++
13:   if $Count == \psi$ then
14:     Update model : $Node.r \leftarrow Node.l$ for every node
        with non-zero mass $r$ or $l$
15:     Reset $Node.l \leftarrow 0$ for every node with non-zero
        mass $l$
16:     $Count \leftarrow 0$
17:   end if
18: end while
```

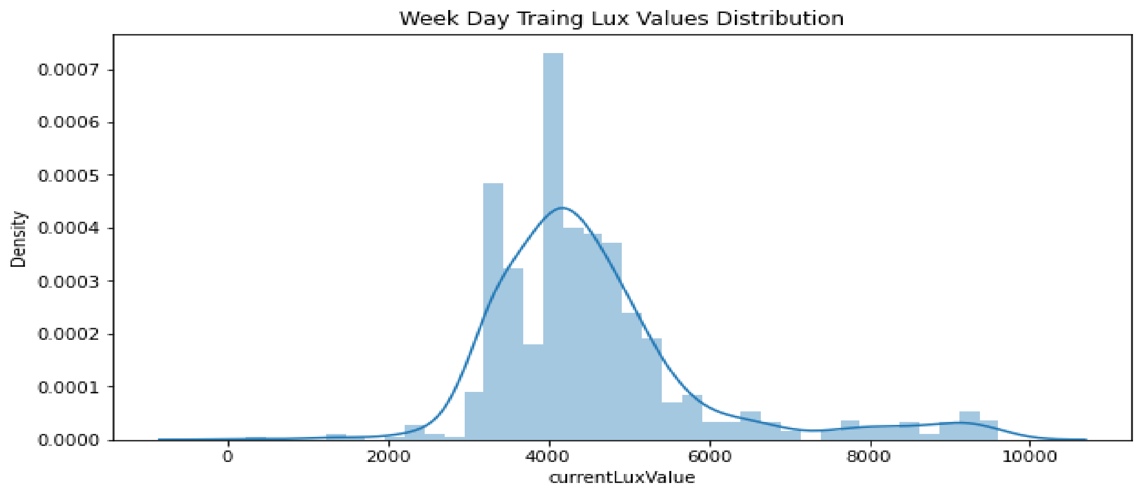
Results:

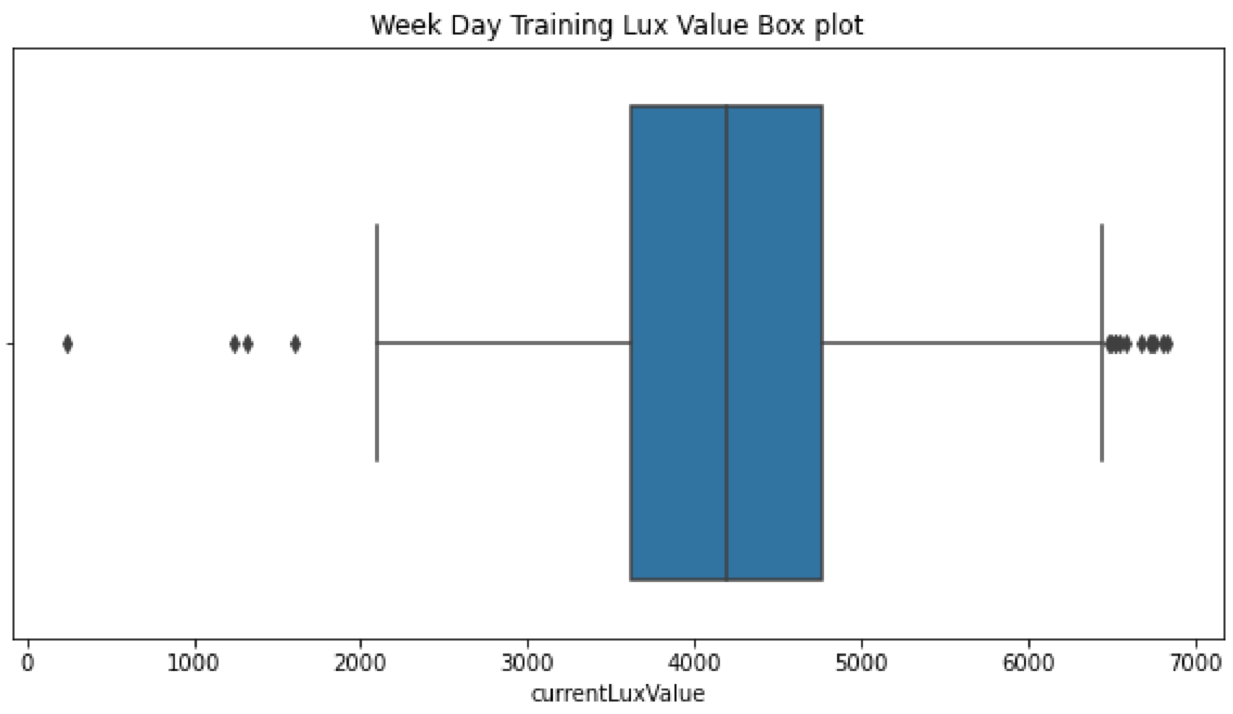
We performed two use cases using two sensors on two different mobile phones on Master of things IOT KIT. The sensor is Lux which measures the light intensity. The first sensor was trained weekend indoor so the light intensity values were small. But the second sensor was trained outdoor in weekdays so the light intensity values were high.

This graph represents the sensor data of week days which has high Lux values:

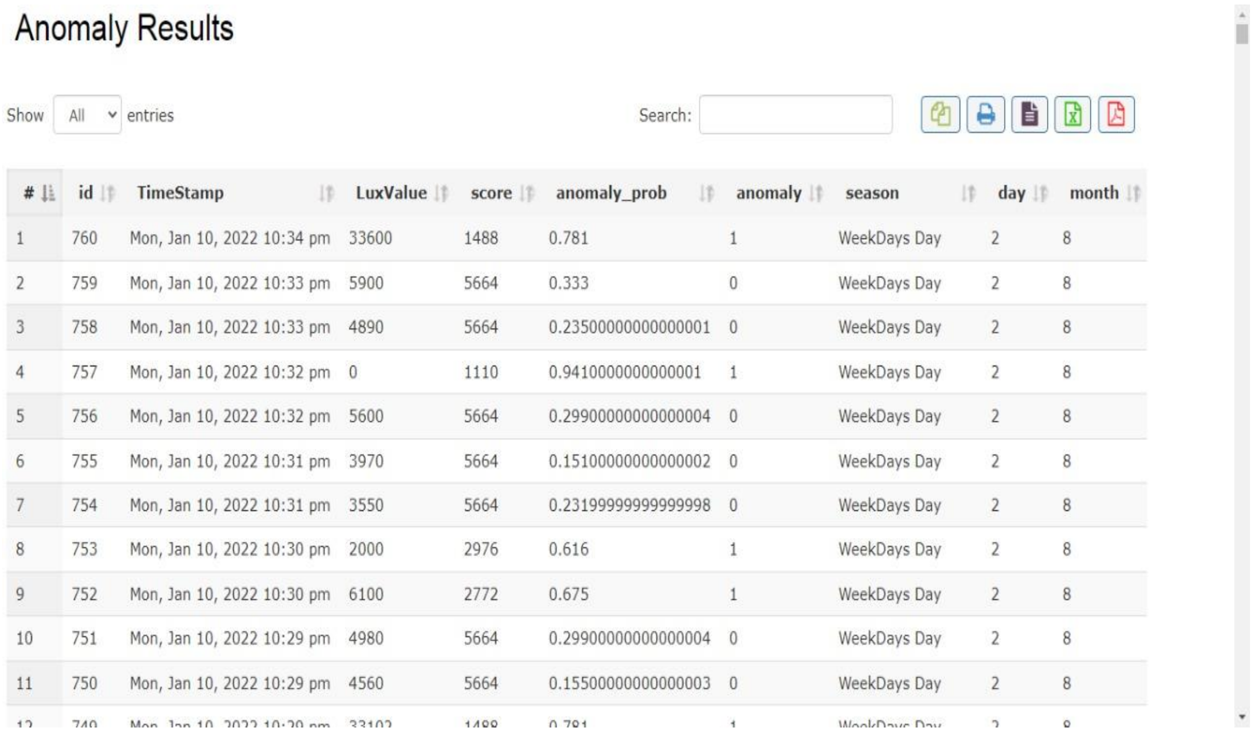


We notice that most data was in range around 4000 Lux light intensity values. This is shown also in the below figures for this data:





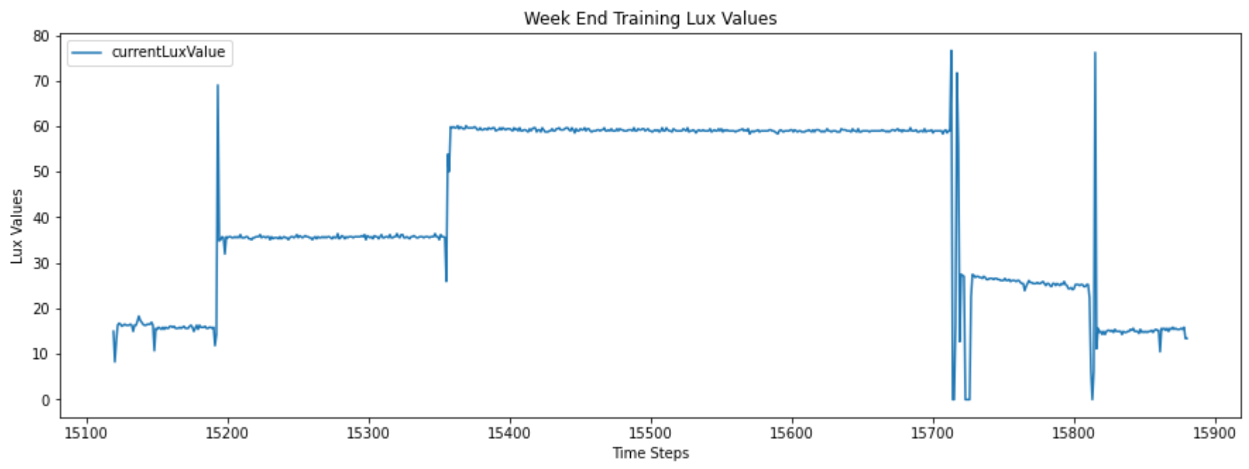
The result tests for this sensor can be shown below in table: the values which lie in range around 4000 are zero anomaly while the very high or very low values are 1 anomaly.


Anomaly Results

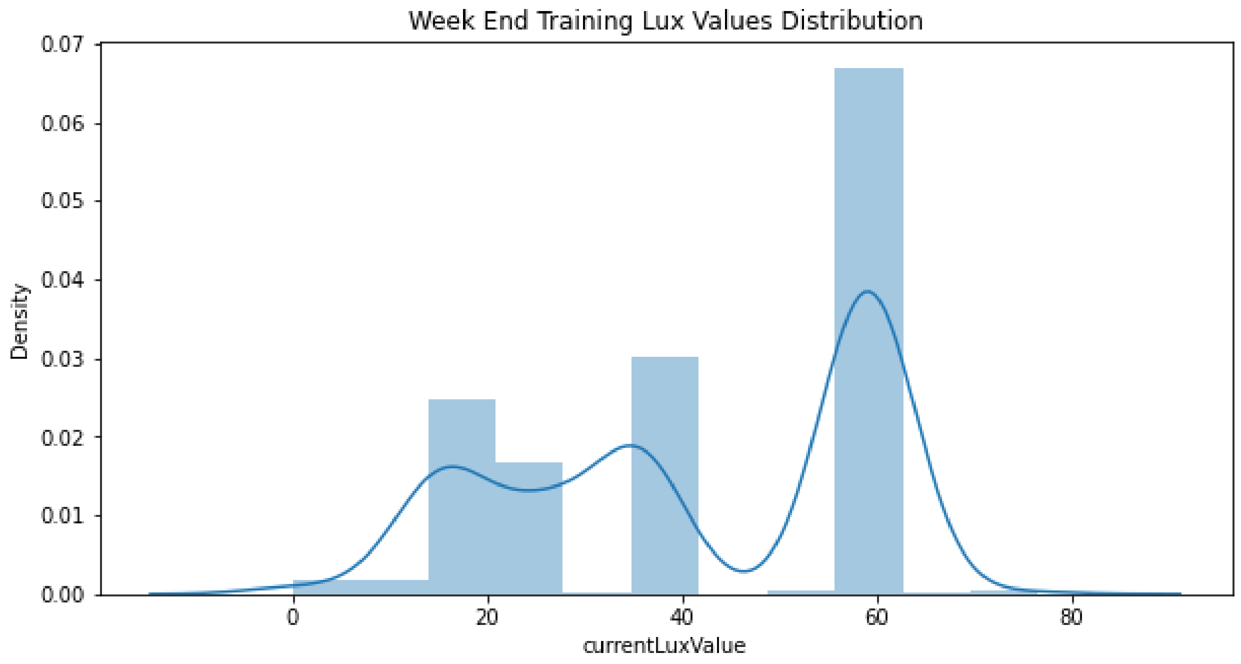
Show All entries Search:

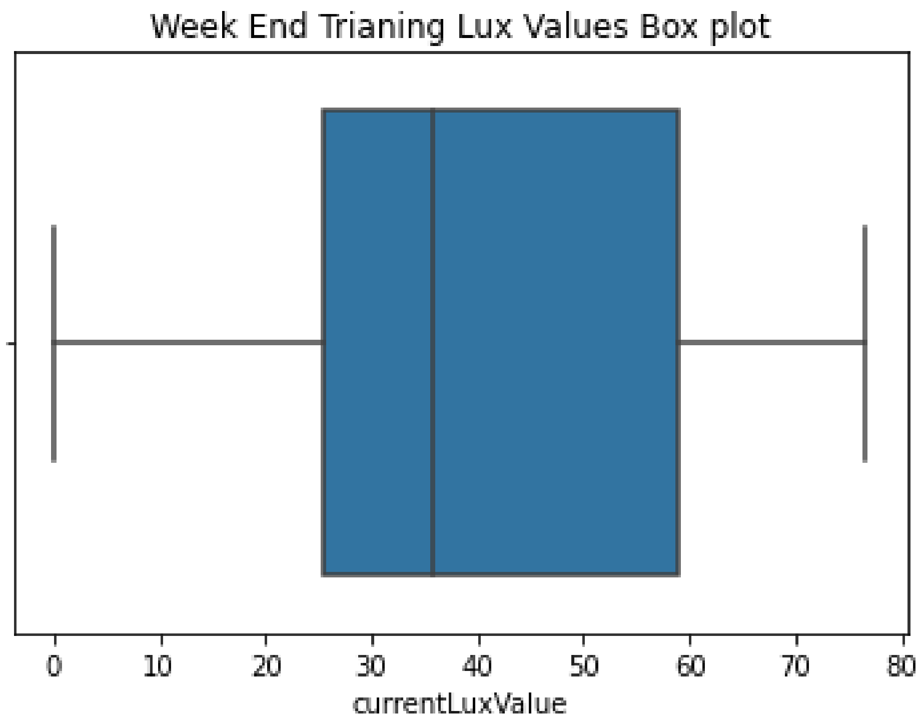
#	id	TimeStamp	LuxValue	score	anomaly_prob	anomaly	season	day	month
1	760	Mon, Jan 10, 2022 10:34 pm	33600	1488	0.781	1	WeekDays Day	2	8
2	759	Mon, Jan 10, 2022 10:33 pm	5900	5664	0.333	0	WeekDays Day	2	8
3	758	Mon, Jan 10, 2022 10:33 pm	4890	5664	0.23500000000000001	0	WeekDays Day	2	8
4	757	Mon, Jan 10, 2022 10:32 pm	0	1110	0.9410000000000001	1	WeekDays Day	2	8
5	756	Mon, Jan 10, 2022 10:32 pm	5600	5664	0.29900000000000004	0	WeekDays Day	2	8
6	755	Mon, Jan 10, 2022 10:31 pm	3970	5664	0.15100000000000002	0	WeekDays Day	2	8
7	754	Mon, Jan 10, 2022 10:31 pm	3550	5664	0.23199999999999998	0	WeekDays Day	2	8
8	753	Mon, Jan 10, 2022 10:30 pm	2000	2976	0.616	1	WeekDays Day	2	8
9	752	Mon, Jan 10, 2022 10:30 pm	6100	2772	0.675	1	WeekDays Day	2	8
10	751	Mon, Jan 10, 2022 10:29 pm	4980	5664	0.29900000000000004	0	WeekDays Day	2	8
11	750	Mon, Jan 10, 2022 10:29 pm	4560	5664	0.15500000000000003	0	WeekDays Day	2	8
12	749	Mon, Jan 10, 2022 10:29 pm	33102	1488	0.781	1	WeekDays Day	2	8

For the second sensor which was trained on week end indoor for low lux values the trained data for the first time window are shown below:

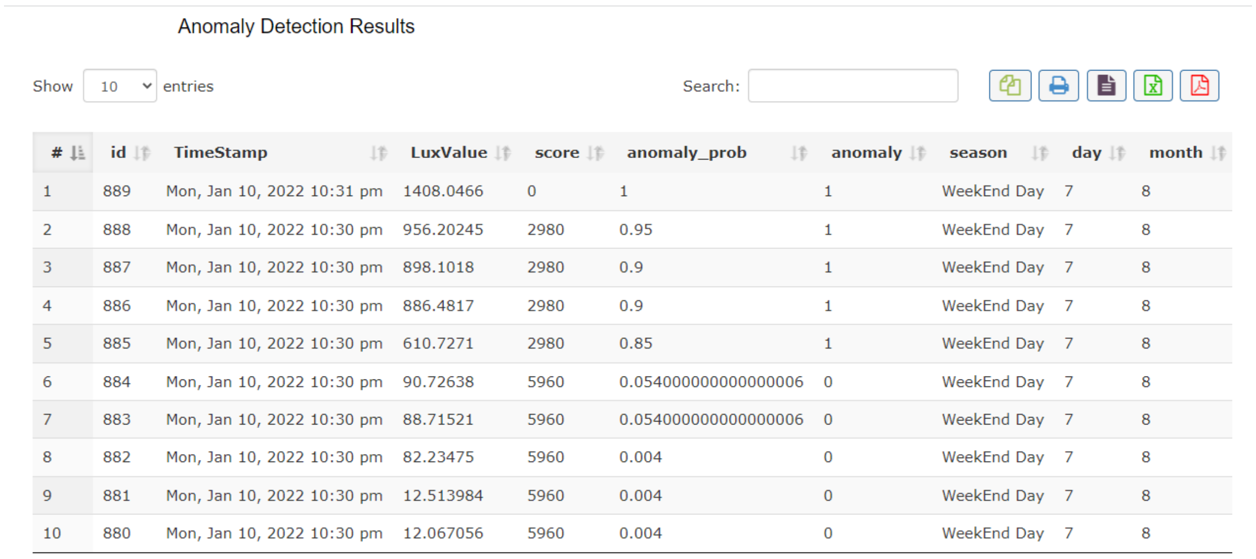


We notice that the Lux values are very low in range below 100 as it is trained indoor. And the results are shown also below:





We tested this also on MOT to show results on the below table: the very high values are anomaly while the low ones are normal behavior so zero anomaly.



Anomaly Detection Results

#	id	TimeStamp	LuxValue	score	anomaly_prob	anomaly	season	day	month
1	889	Mon, Jan 10, 2022 10:31 pm	1408.0466	0	1	1	WeekEnd Day	7	8
2	888	Mon, Jan 10, 2022 10:30 pm	956.20245	2980	0.95	1	WeekEnd Day	7	8
3	887	Mon, Jan 10, 2022 10:30 pm	898.1018	2980	0.9	1	WeekEnd Day	7	8
4	886	Mon, Jan 10, 2022 10:30 pm	886.4817	2980	0.9	1	WeekEnd Day	7	8
5	885	Mon, Jan 10, 2022 10:30 pm	610.7271	2980	0.85	1	WeekEnd Day	7	8
6	884	Mon, Jan 10, 2022 10:30 pm	90.72638	5960	0.054000000000000006	0	WeekEnd Day	7	8
7	883	Mon, Jan 10, 2022 10:30 pm	88.71521	5960	0.054000000000000006	0	WeekEnd Day	7	8
8	882	Mon, Jan 10, 2022 10:30 pm	82.23475	5960	0.004	0	WeekEnd Day	7	8
9	881	Mon, Jan 10, 2022 10:30 pm	12.513984	5960	0.004	0	WeekEnd Day	7	8
10	880	Mon, Jan 10, 2022 10:30 pm	12.067056	5960	0.004	0	WeekEnd Day	7	8

References:

- https://ieeexplore.ieee.org/abstract/document/8939201/

- https://www.semanticscholar.org/paper/Fast-Anomaly-Detection-for-Streaming-Data-Tan-Ting/e6826171c63f591e361dca90c5e6f6efb1cf9e5e

- https://github.com/scikit-multiflow/scikit-multiflow/tree/master/src/skmultiflow/anomaly_detection

- https://www.semanticscholar.org/paper/Anomaly-Detection-for-Data-Streams-Based-on-Forest-Togbe-Barry/f0fe48c7357cf98b7b34eed30c9c8cb22e1918ad