The Journal of Open Source Software

eif: Extended Isolation Forest for Anomaly Detection

Sahand Hariri[1] **and Matias Carrasco Kind**[2]

1 Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, USA **2** National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. 1205 W Clark St, Urbana, IL USA 61801

DOI: 10.21105/joss.00994

Software

- Review
- Repository
- Archive

Submitted: 02 October 2018
Published: 08 October 2018

Summary

The problem of anomaly detection has wide range of applications in various fields and scientific applications. Anomalous data can have as much scientific value as normal data or in some cases even more, and it is of vital importance to have robust, fast and reliable algorithms to detect and flag such anomalies. In this paper, we present an extension to the model-free anomaly detection algorithm, Isolation Forest (Liu, Ting, & Zhou, 2008, 2012). This extension, named Extended Isolation Forest (EIF), improves the consistency and reliability of the anomaly score produced by standard methods for a given data point. We show that the standard Isolation Forest produces inconsistent anomaly score maps, and that these score maps suffer from an artifact produced as a result of how the criteria for branching operation of the binary tree is selected.

Our method allows for the slicing of the data to be done using hyperplanes with random slopes which results in improved score maps. The consistency and reliability of the algorithm is much improved using this extension. This is briefly mentioned in (Liu, Ting, & Zhou, 2010) although in the context of optimization for detection of anomaly clusters. Here we show the need for an improvement on the source algorithm to improve the scoring of anomalies and the robustness of the score maps especially around edges of nominal data. We discuss the sources of the problem, and we present an efficient way for choosing these hyperplanes which give way to multiple extension levels in the case of higher dimensional data. The standard Isolation Forest is therefore a special case of the Extended Isolation Forest as presented it here. We find no appreciable difference in the rate of convergence nor in computational time between the standard Isolation Forest and EIF which highlights its potential as anomaly detection algorithm.

This method has been written completely in Python and the only requirement is NumPy (Oliphant, 2006) and can be easily installed using `pip install eif` or directly from the source code located at https://github.com/sahandha/eif where we have also added some example Jupyter(Kluyver et al., 2016) Notebooks and documented code on how to get started.

Motivation

While various techniques exist for approaching anomaly detection, Isolation Forest (Liu et al., 2008) is one with unique capabilities. This algorithm can readily work on high dimensional data, it is model free, and it scales well. It is therefore highly desirable and easy to use. However, looking at score maps for some basic example, we can see that the anomaly scores produced by the standard Isolation Forest are inconsistent, . To see this we look at the three examples shown in Figure 1.

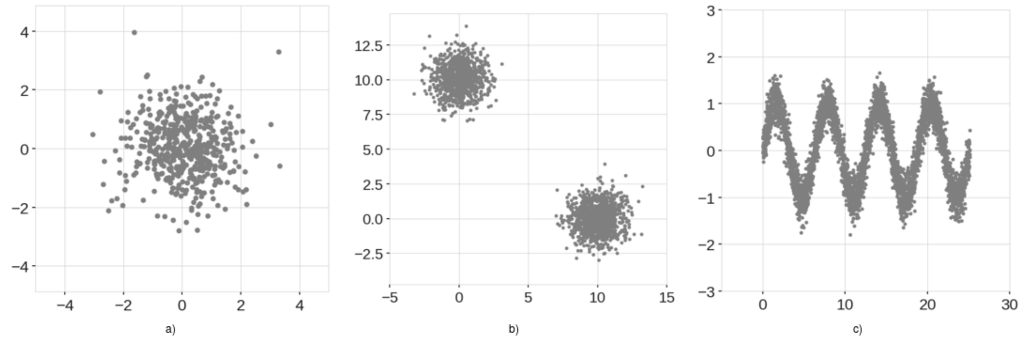

Figure 1: Example training data. a) Normally distributed cluster. b) Two normally distributed clusters. c) Sinusoidal data points with Gaussian noise.

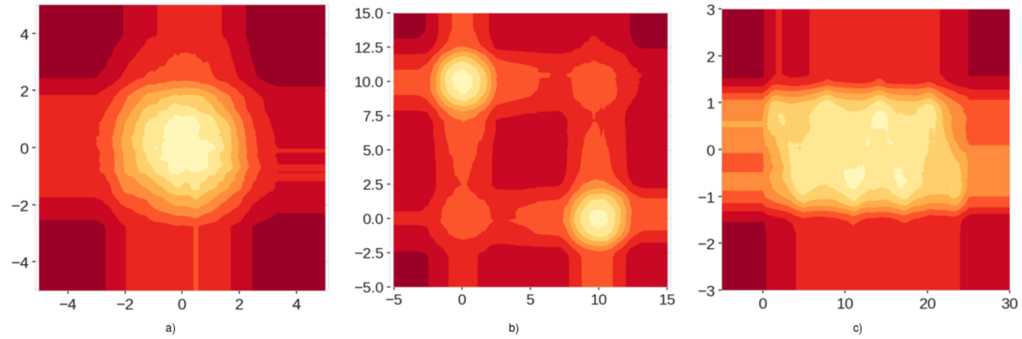

Figure 2: Score maps using the Standard Isolation Forest for the points from Figure 1. We can see the bands and artifacts on these maps

In each case, we use the data to train our Isolation Forest. We then use the trained models to score a square grid of uniformly distributed data points, which results in score maps shown in Figure 2. Through the simplicity of the example data, we have an intuition about what the score maps should look like. For example, for the data shown in Figure 1a, we expect to see low anomaly scores in the center of the map, while the anomaly score should increase as we move radially away from the center. Similarly for the other figures.

Looking at the score maps produced by the standard Isolation Forest shown in Figure 2, we can clearly see the inconsistencies in the scores. While we can clearly see a region of low anomaly score in the center in Figure 2a, we can also see regions aligned with x and y axes passing through the origin that have lower anomaly scores compared to the four corners of the region. Based on our intuitive understanding of the data, this cannot be correct. A similar phenomenon is observed in Figure 2b. In this case, the problem is amplified. Since there are two clusters, the artificially low anomaly score regions intersect close to points (0,0) and (10,10), and create low anomaly score regions where there is no data. It is immediately obvious how this can be problematic. As for the third example, figure 2c shows that the structure of the data is completely lost. The sinusoidal shape is essentially treated as one rectangular blob.

We present a brief description of how Isolation Forest works in order to explain our extension.

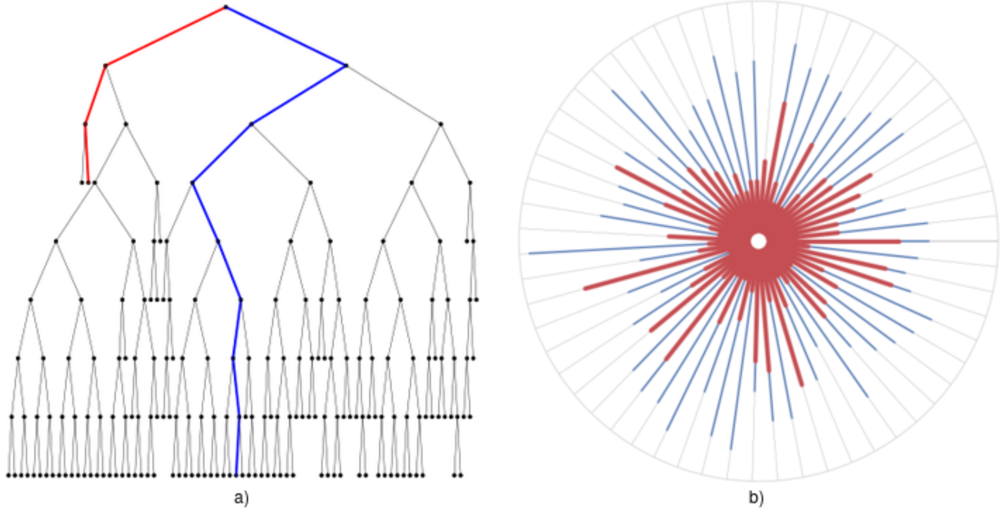

Figure 3: a) Shows an example tree formed from the example data while b) shows the forest generated where each tree is represented by a radial line from the center to the outer circle. Anomalous points (shown in red) are isolated very quickly,which means they reach shallower depths than nominal points (shown in blue).

Isolation Forest

Given a dataset of dimension N, the algorithm chooses a random sub-sample of data to construct a binary tree. The branching process of the tree occurs by selecting a random dimension x_i with i in $\{1, 2, ..., N\}$ of the data (a single variable). It then selects a random value v within the minimum and maximum values in that dimension. If a given data point possesses a value smaller than v for dimension x_i, then that point is sent to the left branch, otherwise it is sent to the right branch. In this manner the data on the current node of the tree is split in two. This process of branching is performed recursively over the dataset until a single point is isolated, or a predetermined depth limit is reached. The process begins again with a new random sub-sample to build another randomized tree. After building a large ensemble of trees, i.e. a forest, the training is complete. The creation of the forest is summarized in Algorithm 1 and the Class `iForest()` in the code.

```
Algorithm 1 $iForest(X, t, \psi)$
Require: $X$ - input data, $t$ - number of trees, $h$ - sub-sampling size
Ensure: a set of $t$ $iTrees$
  1. Initialize $Forest$
  2. set height limit $l = ceiling(\log_2 \psi)$
  3. for $i = 1$ to $t$ do
  4. $X' \leftarrow sample(X, \psi)$
  5. $Forest \leftarrow Forest \cup iTree(X', 0, l)$
  6. end for
```

During the scoring step, a new candidate data point (or one chosen from the data used to create the trees) is run through all the trees, and an ensemble anomaly score is assigned based on the depth the point reaches in each tree. Figure 3 shows an schematic example of a tree and a forest plotted radially.

It turns out the splitting process described above is the main source of the bias observed in

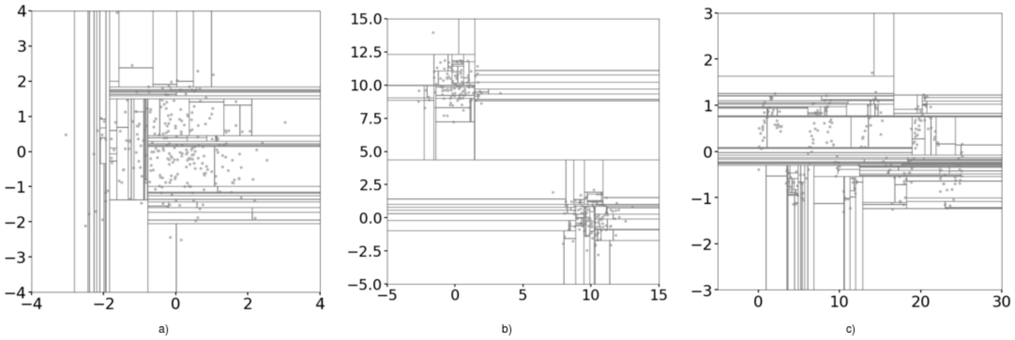

Figure 4: Splitting of data in the domain during the process of construction of one tree.

the score maps. Figure 4 shows the process described above for each one of the examples considered thus far. The branch cuts are always parallel to the axes, and as a result over construction of many trees, regions in the domain that don't occupy any data points receive superfluous branch cuts.

The branching process shouldn't be affected by the choice of our coordinate system.

Extended Isolation Forest

The Extended Isolation Forest remedies this problem by allowing the branching process to occur in every direction. The process of choosing branch cuts is altered so that at each node, instead of choosing a random feature along with a random value, we choose a random normal vector along with a random intercept point. The criteria of splitting the node in the binary tree is determined by the equation:

$$(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0 \tag{1}$$

where n is the normal vector, p is the random intercept. x is a give data point. If the inequality is satisfied, point x is passed down to the left branch of the tree. Otherwise, it is passed down to the right branch as explained in Algorithm 2. Each tree is a Python Class `iTree()` in the code.

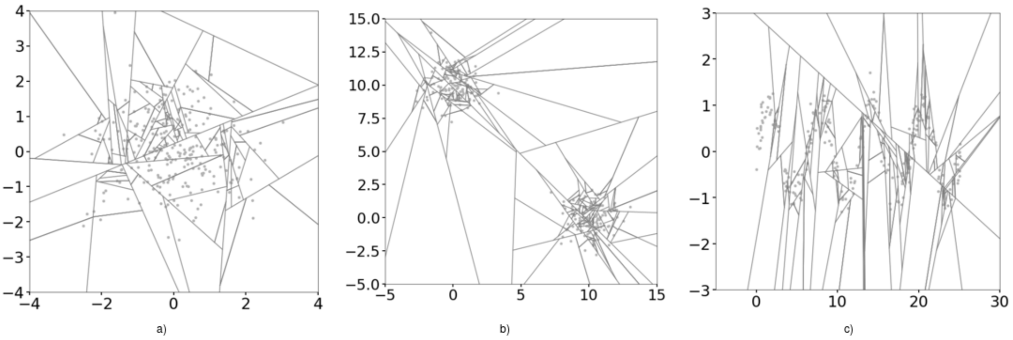

Figure 5: Same as Figure 5 but using Extended Isolation Forest

Algorithm 2 $iTree(X, e, l)$

```
Require: $X$ - input data, $e$ - current tree height, $l$ - height limit
Ensure: an $iTree$
 1. if $e \geq l$ or $|X| \leq 1$ then
 2.     return $exNode\{Size \leftarrow |X|\}$
 3. else
 4. get a random normal vector $\vec{n} \in \mathbb{R}^{|X|}$ where each coordinate is $\sim \mathcal{N}(0, 1)$
 5. randomly select an intercept point $\vec{p} \in \mathbb{R}^{|X|}$ in the range of $X$
 6. set coordinates of $\vec{n}$ to zero according to extension level
 7. $X_l \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} \leq 0)$
 8. $X_r \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} > 0)$
 9. return $inNode\{$
        $Left \leftarrow iTree(X_l, e + 1, l)$,
        $Right \leftarrow iTree(X_r, e + 1, l)$,
        $Normal \leftarrow \vec{n}$,
        $Intercept \leftarrow \vec{p}\}$
10. end if
```

Figure 5 shows the resulting branch cuts int he domain for each of our examples.

We can see that the region is divided much more uniformly, and without the bias introducing effects of the coordinate system. As in the case of the standard Isolation Forest, the anomaly score is computed by the aggregated depth that a given point reaches on each *iTree*. The depth computation is defined by the Class `PathFactor()` in the code and is given by Algorithm 3, with a normalizing factor defined as the average depth in an unsuccessful search in a Binary Search Tree (BST):

$$c(n) = 2H(n-1) - (2(n-1)/n) \tag{2}$$

where $H(i)$ is the harmonic number and can be estimated by $\ln(i) + 0.5772156649$ (Euler's constant) (Liu et al., 2012) and n is the number of points used in he construction of the trees. The final anomaly score s for a given point x is given by :

$$s(x, n) = 2^{-E(h(x))/c(n)} \tag{3}$$

where $E(h(x))$ is the average of the depths the point reached in all trees.

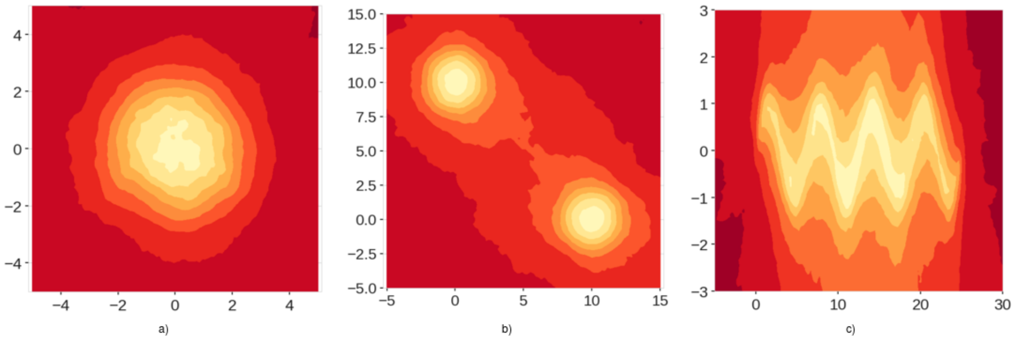

Figure 6: Score maps using the Extended Isolation Forest.

```
Algorithm 3 $PathLength(x, T, e)$
Require: $\vec{x}$ - an instance, $T$ - an iTree, $e$ - current path length; initialized to 0
Ensure: path length of $\vec{x}$
  if $T$ is an external node then
      return $e + c(T.size)\{c(.)$ is defined in Equation 2$\}$
  end if
  $\vec{n} \leftarrow T.Normal$
  $\vec{p} \leftarrow T.Intercept$
  if $\{(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0\}$ then
      return $PathLength(\vec{x}, T.left, e + 1)$
  else if $\{(\vec{x} - \vec{p}) \cdot \vec{n} > 0\}$ then
  return $PathLength(\vec{x}, T.rigth, e + 1)$
  end if
```

As we see in Figure 6, these modifications completely fix the issue with the score maps that we saw before and produce reliable results. Clearly, these score maps are a much better representation of anomaly score distributions.

Figure 7 shows a very simple example of anomalies and nominal points from a Single blob example as shown in Figure 1a. It also shows the distribution of the anomaly scores which can be used to make hard cuts on the definition of anomalies or even assign probabilities to each point.

Conclusions

We present an robust, open source extension to the Isolation Forest algorithm which produces accurate anomaly score maps. We provide the motivation behind this code and the comparison between the standard and the extended methods. Given the scalability of this method, this algorithm can be extended to work on distributed computing systems as shown in (Hariri & Carrasco Kind, 2018) and can be readily used in scientific applications to find anomalies in the data.

References

Hariri, S., & Carrasco Kind, M. (2018). Batch and online anomaly detection for scientific applications in a kubernetes environment. In *Proceedings of the 9th workshop on*

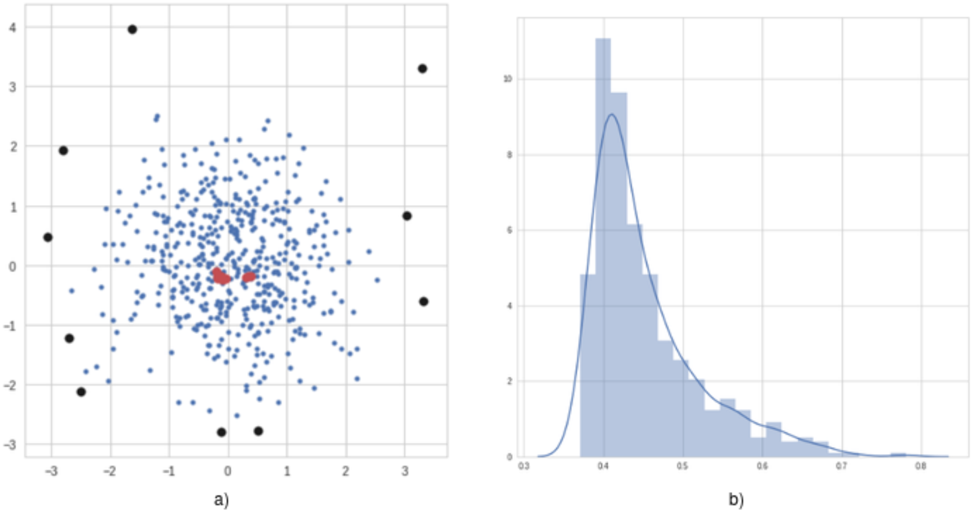

Figure 7: a) Shows the dataset used, some sample anomalous data points discovered using the algorithm are highlighted in black. We also highlight some nominal points in red. In b), we have the distribution of anomaly scores obtained by the algorithm.

scientific cloud computing, ScienceCloud'18 (pp. 3:1–3:7). New York, NY, USA: ACM. doi:10.1145/3217880.3217883

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., et al. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. (F. Loizides & B. Schmidt, Eds.). IOS Press.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In *Data mining, 2008. ICDM'08. Eighth ieee international conference on* (pp. 413–422). IEEE.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2010). On detecting clustered anomalies using sciforest. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 274–290). Springer.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, *6*(1), 3:1–3:39. doi:10.1145/2133360.2133363

Oliphant, T. E. (2006). *A guide to numpy*. Trelgol Publishing.