Graphical Abstract (Optional)

To create your abstract, please type over the instructions in the template box below. Fonts or abstract dimensions should not be changed or altered.

Generalized Isolation Forest for Anomaly Detection

Julien Lesouple, Cédric Baudoin, Marc Spigai and Jean-Yves Tourneret

This letter introduces a generalization of Isolation Forest (IF) based on the existing Extended IF (EIF). EIF has shown some interest compared to IF being for instance more robust to some artefacts. However, some information can be lost when computing the EIF trees since the sampled threshold might lead to empty branches. This letter introduces a generalized isolation forest algorithm called Generalized IF (GIF) to overcome these issues. GIF is faster than EIF with a similar performance, as shown in several simulation results associated with reference databases used for anomaly detection.

Research Highlights (Required)

It should be short collection of bullet points that convey the core findings of the article. It should include 3 to 5 bullet points (maximum 85 characters, including spaces, per bullet point.)

- We propose a new unsupervised Anomaly Detection (AD) algorithm
- This algorithm is based on Isolation Forest with random hyperplanes instead of random dimensions
- The proposed method improves the existing Extended Isolation Forest (EIF) in terms of computation time

Pattern Recognition Letters
journal homepage: www.elsevier.com

Generalized Isolation Forest for Anomaly Detection

Julien Lesouple[a,**], Cédric Baudoin[b], Marc Spigai[b], Jean-Yves Tourneret[a,c]

[a]TéSA, 7 Boulevard de la Gare, 31000 Toulouse, France
[b]Thales Alenia Space, 26 Avenue Jean-François Champollion, 31100 Toulouse France
[c]University of Toulouse/INP-ENSEEIHT/IRIT, 2 Rue Charles Camichel, 31071, Toulouse, France

ABSTRACT

This letter introduces a generalization of Isolation Forest (IF) based on the existing Extended IF (EIF). EIF has shown some interest compared to IF being for instance more robust to some artefacts. However, some information can be lost when computing the EIF trees since the sampled threshold might lead to empty branches. This letter introduces a generalized isolation forest algorithm called Generalized IF (GIF) to overcome these issues. GIF is faster than EIF with a similar performance, as shown in several simulation results associated with reference databases used for anomaly detection.

1. Introduction

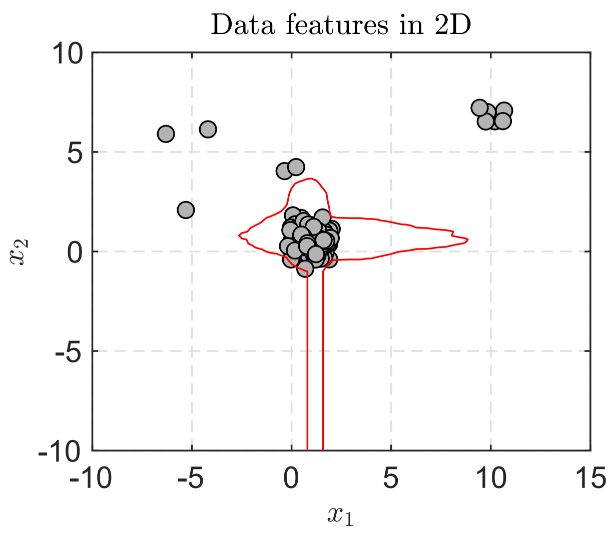
Anomaly Detection (AD, Chandola et al. (2009)) has gained attention in the past few years, due to the enhancement of modern computers and the increasing interest for machine learning algorithms. AD consists in detecting rare patterns or unobserved samples in data, referred to as anomalies. It is widely used in potentially critical environments, e.g., in credit fraud detection (Brause et al. (1999)), crowd surveillance (Leach et al. (2014)), or in satellite telemetry monitoring (Yairi et al. (2017); Pilastre et al. (2020)). AD has received an increasing interest for satellite monitoring in the past few years, with new satellite constellations, resulting in a huge amount of data to be processed at the same time. Time-series resulting from satellite telemetry are of course used for the constellation mission but also for system monitoring and failure prevention.

This letter focuses on unsupervised AD algorithms, which learn the normal behavior of unlabeled data using a so-called training dataset. The performance of the algorithm can then be tested using a labeled dataset called test set. Various AD algorithms have been proposed in the literature including those based on nearest neighbors (Local Outlier Factor, Breunig et al. (2000), Local Outlier Probability (LoOP), Kriegel et al. (2009) or Neighborhood Construction (NC), İnkaya et al. (2015)), support vector machines (Support Vector Data Description, Tax and Duin (2004), One Class Support Vector Machines, Schölkopf et al. (2001)), Sparse Coding (Dutta and Banerjee (2019)), or Isolation Forest (IF, Liu et al. (2008)).

A specific attention is devoted in this letter to IF, which aims at finding anomalies with the idea that in some feature space, anomalies should be "far" from other data. To look for these anomalies, IF generates random isolation trees in order to isolate each data point. The number of branches required to isolate each point is then computed for each tree. The mean of this number of branches defines the expected path length, which is used to isolate a point of interest. The expected path length is generally small for anomalies (contrary to nominal data) since anomalies are far from the majority of nominal data. However, the trees generated by IF are considering a random feature at each node, which can lead to some artefacts in the score map function, as shown in Hariri et al. (2019). In order to improve the isolation of data points, tree branches with random hyperplanes can be considered (Hariri et al. (2019)). Random hyperplanes are not necessarily parallel to one of the components of the feature vector and have been used in the extended IF (EIF) algorithm. Unfortunately, this strategy generates a lot of empty branches, which increases the complexity of the trees belonging to the forest. This letter goes a step further by proposing a new IF construction inspired by the work of Hariri et al. (2019) leading to the generalized isolation forest (GIF) algorithm. The GIF algorithm generates trees without any empty branch, which significantly improves the execution times when compared to EIF.

This letter is organized as follow: Section II recalls the principles of IF and EIF and introduces the proposed GIF algorithm. Section III evaluates the performance of GIF using experiments

**Corresponding author: Tel.: +33-5-61-24-73-64;
e-mail: julien.lesouple@tesa.prd.fr (Julien Lesouple)



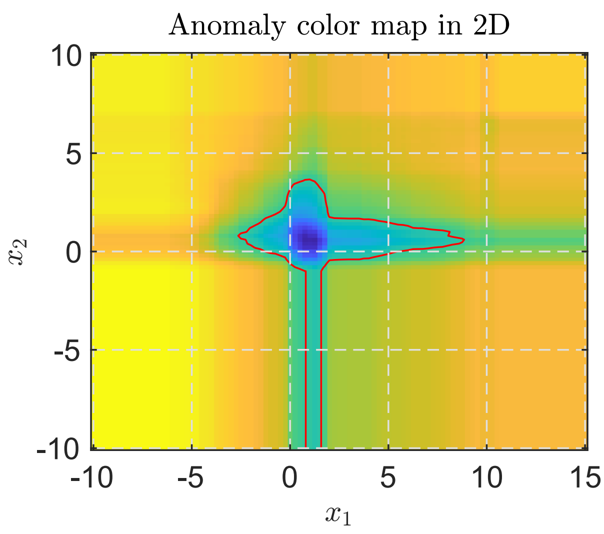


Fig. 1: Illustration of IF problems using artificial 2D data. Training data are depicted in the left figure as well as the curve $s(\boldsymbol{x}, n) = s_0$ (displayed in red). The right figure shows the heat map of the anomaly score (dark blue corresponds to values next to 0 and light yellow to value close to 1).

on both synthetic and real benchmark datasets. Conclusion are reported in Section IV.

2. Isolation Forest

2.1. Original Formulation

IF generates $t > 0$ random trees to partition the data, and computes for each tree the number of nodes required to isolate each training vector. Anomalies are then detected as the vectors whose average path lengths are the smallest, motivated by the fact that nominal data are more concentrated than anomalies and thus require more nodes to be isolated.

To create a random isolation tree, assume that we have n training data $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} & \ldots & x_{i,d} \end{bmatrix}^T \in \mathbb{R}^d$. We will also use the notation $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \ldots & \boldsymbol{x}_n \end{bmatrix}^T \in \mathbb{R}^{n\times d}$ for the matrix gathering all the training data. To create a random node and split the dataset into two subsets, one component of $\mathbb{R}^d$ (denoted as q) is chosen randomly, and a split value p is sampled uniformly in the interval $[\min_{i=1,\ldots,n} x_{i,q}; \max_{i=1,\ldots,n} x_{i,q}]$. The dataset is then split into two parts: the so-called left branch corresponding to the set $\{\boldsymbol{x}_i, x_{i,q} \leq p\}$ and the so-called right branch, corresponding to the set $\{\boldsymbol{x}_i, x_{i,q} > p\}$. The tree is created by applying this procedure iteratively to each branch until a branch contains a unique data point, or until some depth l has been reached. To create an IF, this procedure could be applied several times to the whole learning dataset. However, authors in Liu et al. (2008) have shown that for each tree, a sub-sample of the whole dataset of size $\psi > 0$ (chosen to $\psi = 256$ in this letter) can be considered with similar performance and improved computation time.

Once the forest has been created by generating t random isolation trees, the expected path length $h(\boldsymbol{x})$ to isolate a point $\boldsymbol{x}$ is computed using the mean of the path lengths required to isolate the point using each generated tree. Finally, an anomaly score is defined as

$$s(\boldsymbol{x}) = 2^{-\frac{E[h(\boldsymbol{x})]}{c(\psi)}}, \tag{1}$$

where $c(n)$ is the average value of $h(\boldsymbol{x})$ for a dataset of size n, which can be computed as

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

where $H(n)$ is the nth harmonic number (that can be approximated by $\ln(n) + \gamma$, where $\gamma \approx 0.577$ is the Euler-Mascheroni's constant). Thus, when $E[h(\boldsymbol{x})] = c(n)$, the anomaly score of $\boldsymbol{x}$ is $s(\boldsymbol{x}, n) = 0.5$. When $h(\boldsymbol{x})$ tends to $+\infty$, i.e., when $\boldsymbol{x}$ is not an isolated point, the anomaly score tends to 0. Finally, when $h(\boldsymbol{x})$ is small compared to $c(n)$, i.e., when $\boldsymbol{x}$ is an isolated point, the corresponding anomaly score tends to 1. Thus we can define an anomaly threshold $s_0 \in [0, 1]$ such that $\boldsymbol{x}$ is detected as an anomaly when $s(\boldsymbol{x}) > s_0$, and as a nominal data when $s(\boldsymbol{x}) \leq s_0$. Of course, the closer the anomaly score to 1, the more likely $\boldsymbol{x}$ is an anomaly, and the closer the anomaly score to 0, the more likely $\boldsymbol{x}$ is a nominal vector. Thus, a trade-off has to be made to determine an appropriate value of s_0. Authors in Liu et al. (2008) have proposed values for the different parameters that are summarized in Table 1. The resulting IF

Table 1: Proposed values for the various parameters of IF.

Parameters	Meaning	Proposed value
t	Number of trees	100
ψ	Sub-sample size	256
l	Tree maximum depth	$\text{ceil}(\log_2 \psi) = 8$
s_0	Anomaly detection threshold	0.6

algorithm is a convenient solution to detect anomalies without assumptions on the data distribution and it is computationally efficient. However, this algorithm suffers from a bias due to the way trees are created. Indeed, by randomly choosing one dimension to split the data, parallel hyperplanes are used (with a normal vector collinear to the selected dimension), and data spread around stripes parallel to the axis and passing through the cluster have a lower anomaly score, as depicted in Figure 1.

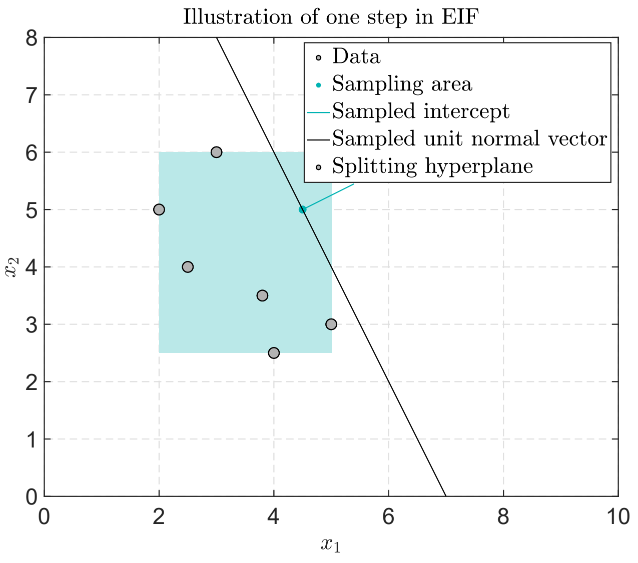


Fig. 2: Illustration of an EIF drawback using artificial 2D data. A splitting hyperplane is created by sampling a random unit vector and a random intercept in the sampling area. As one can see, using this strategy, all the data points are below the hyperplane (for this outcome). Thus the corresponding right branch of the tree will be empty.

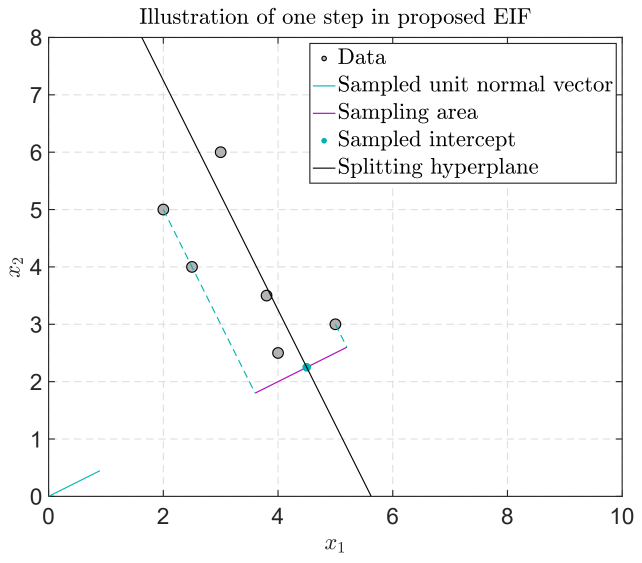


Fig. 3: Illustration of the proposed GIF approach. A splitting hyperplane is created by sampling a random unit vector and a random intercept in the sampling area, which reduces to a line (and not a square). This strategy has the advantage of having data points on each side of the splitting hyperplane.

2.2. *Extended IF*

To avoid artefacts such as those illustrated in Fig. 1, an improved solution was presented in Hariri et al. (2019) referred to as EIF. As explained in Hariri et al. (2019), the main drawback of IF is due to the way hyperplanes are constructed to split the data. Indeed, since the drawn normal vectors are chosen according to each dimension of $\mathbb{R}^d$, a discrete set of orthogonal directions is generated, which is at the origin of these vertical lines appearing in the level sets of $s(\boldsymbol{x})$. To mitigate this problem, a normal vector $\boldsymbol{w}$ can be sampled for each decision hyperplane randomly chosen in the unit sphere of $\mathbb{R}^d$ (Hariri et al. (2019)), i.e., a Gaussian vector is sampled according to $\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d) \in \mathbb{R}^d$ and normalized leading to $\boldsymbol{w} = \boldsymbol{u}/\|\boldsymbol{u}\|_2$ (Muller (1959)). To select the split value, an intercept vector $\boldsymbol{p} \in \mathbb{R}^d$ is sampled uniformly in the smallest axis-bouding hypercube enclosing all the samples at a branching point (as illustrated in Fig. 2). The two branches of the tree are defined depending on whether $(\boldsymbol{x} - \boldsymbol{p})^T \boldsymbol{w} > 0$ (right branch of the tree) or $(\boldsymbol{x} - \boldsymbol{p})^T \boldsymbol{w} \leq 0$ (left branch of the tree). The hyperplane is thus the one defined by the normal vector $\boldsymbol{w}$ and containing the intercept point $\boldsymbol{p}$. One drawback of this method is that it can lead to empty branches in the tree, which goes against the idea of IF (whose idea is to split the tree until the number of points equals one or until a given maximal depth has been reached in order to efficiently isolate the data). This situation is depicted in Fig. 2 for the previous 2D example.

This letter studies a variation of EIF avoiding empty branches in each tree, referred to as generalized isolation forest (GIF), which is detailed in the next section.

2.3. *Generalized Isolation Forest*

In order to avoid empty branches in EIF, we transpose the EIF problem into the original one defining IF. More precisely, we propose to project all the data on the sampled normal unit vector, look for the minimum and maximum values of the projections (identified by the dotted lines in Fig. 3) and sample a split value uniformly between these two values. Note that this sampling ensures that there is at least one data in each branch of a tree: the first branch being defined from the min value and the second branch associated with the max value. This is equivalent to sample an intercept point on the restriction of the line spanned by the normal vector to the segment between the minimum and maximum values of the projected data points as shown in Fig. 3. This strategy ensures that the two branches of a tree are not empty, contrary to EIF. Note that it is equivalent to EIF where the sampling volume has been reduced to the convex hull of the data. Empty branches in EIF are due to intercepts sampled outside the convex hull of the considered samples and inside the axis-bounding hypercube. For EIF, the probability of sampling an intercept leading to an empty branch is therefore the volume between the hypercube and the convex hull, divided by the volume of the hypercube. Conversely, this volume equals 0 for GIF. Note that probability of having an empty branch in EIF increases as the number of dimensions increases, due to the curse of dimensionality, which motivates the need to avoid such situations. Finally, the proposed method can be defined by three algorithms summarized in Alg. 1, 2 and 3, inspired by Hariri et al. (2019) and Liu et al. (2008).

3. Experiments

This section evaluates the performance of the proposed GIF algorithm using synthetic 2D data and some benchmark datasets considered in Hariri et al. (2019) and Goldstein (2015).

3.1. *Synthetic datasets*

In order to appreciate the benefits of GIF with respect to EIF and IF, we first consider three datasets of synthetic 2D samples

Algorithm 1 Create the forest

```
Input: $X$ - input data, $t$ - number of trees, $\psi$ - subsampling size
Output: $Forest$ - a set of $iTrees$
 1: function iForest($X, t, \psi$)
 2:     initialize $Forest \leftarrow$ struct                    ▷ Empty structure
 3:     set $l$ = ceil($\log_2 \psi$)                            ▷ Height limit
 4:     for all $i$ = 1 to $t$ do
 5:         $X' \leftarrow$ Sample($X, \psi$)                    ▷ Subsample of size $\psi$
 6:         $Forest.Tree(i) \leftarrow$ iTree($X'$, 0, $l$)
 7:     end for
 8: end function
```

Algorithm 2 Create a tree

```
Input: $X$ - input data, $e$ - current tree height, $l$ - height limit
Output: $Tree$ - an iTree
 1: function iTree($X, e, l$)
 2:     initialize $Tree \leftarrow$ struct                      ▷ Empty structure
 3:     if $e \geq l$ or $|X| \leq 1$ then
 4:         $Tree.Size \leftarrow |X|$                           ▷ Number of remaining data
 5:         $Tree.Type \leftarrow$ 'ext'                         ▷ No nodes after this one
 6:     else
 7:         draw $\boldsymbol{w} \sim \mathcal{N}(0, I_d)$
 8:         $\boldsymbol{w} \leftarrow \boldsymbol{w}/||\boldsymbol{w}||_2$   ▷ Random unit vector of $\mathbb{R}^d$
 9:         $p_{\min} \leftarrow \min(X\boldsymbol{w})$
10:         $p_{\max} \leftarrow \max(X\boldsymbol{w})$
11:         draw $p \sim \mathcal{U}([p_{\min}; p_{\max}])$
12:         $X_l \leftarrow X(X\boldsymbol{w} \leq p, :)$
13:         $X_r \leftarrow X(X\boldsymbol{w} > p, :)$
14:         $Tree.Level \leftarrow e$                            ▷ Level of the node
15:         $Tree.Left \leftarrow$ iTree($X_l, e+1, l$)
16:         $Tree.Right \leftarrow$ iTree($X_r, e+1, l$)
17:         $Tree.Normal \leftarrow \boldsymbol{w}$
18:         $Tree.Threshold \leftarrow p$
19:         $Tree.Type \leftarrow$ 'int'                         ▷ Nodes after this one
20:     end if
21: end function
```

Algorithm 3 Compute isolation score (Path Length)

```
Input: $\boldsymbol{x}$ - input vector, $Tree$ - an iTree, $e$ - current path length
 1: # $e$ must be initialized to 0 when first called
Output: $Length$ - isolation score
 2: function PL($\boldsymbol{x}, Tree, e$)
 3:     if $Tree.Type$ = 'ext' then
 4:         if $Tree.size > 1$ then
 5:             $Length \leftarrow e + c(Tree.size)$             ▷ see (2)
 6:         else
 7:             $Length \leftarrow e$
 8:         end if
 9:     else
10:         $\boldsymbol{w} \leftarrow Tree.Normal$
11:         $\boldsymbol{p} \leftarrow Tree.Threshold$
12:         if $\boldsymbol{x}^T\boldsymbol{w} \leq p$ then
13:             $Length \leftarrow$ PL($\boldsymbol{x}, Tree.Left, e+1$)
14:         else
15:             $Length \leftarrow$ PL($\boldsymbol{x}, Tree.Right, e+1$)
16:         end if
17:     end if
18: end function
```

displayed in Fig. 4. For each dataset, IF, EIF and GIF are run on the same data to learn the corresponding isolation forest. After building the isolation forests, a square area containing all the samples is transformed into a 100×100 grid. The anomaly score is computed for each point of this grid in order to build heat maps that are displayed in Fig. 5. The advantages of EIF and GIF with respect to IF, as already highlighted in Hariri et al. (2019), are clear: the "cross" on the single blob, the sinusoid, and the ghost blobs for the second example disappear for GIF and EIF. In order to have a quantitative appreciation of the various methods, the next experiments consider several benchmark datasets whose anomalies are detected using the different algorithms.

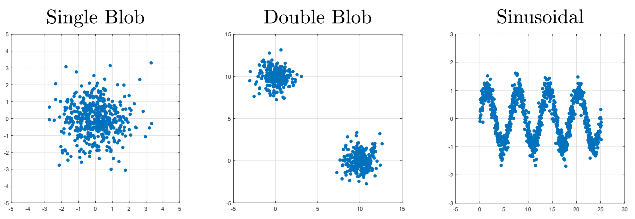


Fig. 4: Synthetic 2D datasets used to visualize the gain of EIF and GIF.

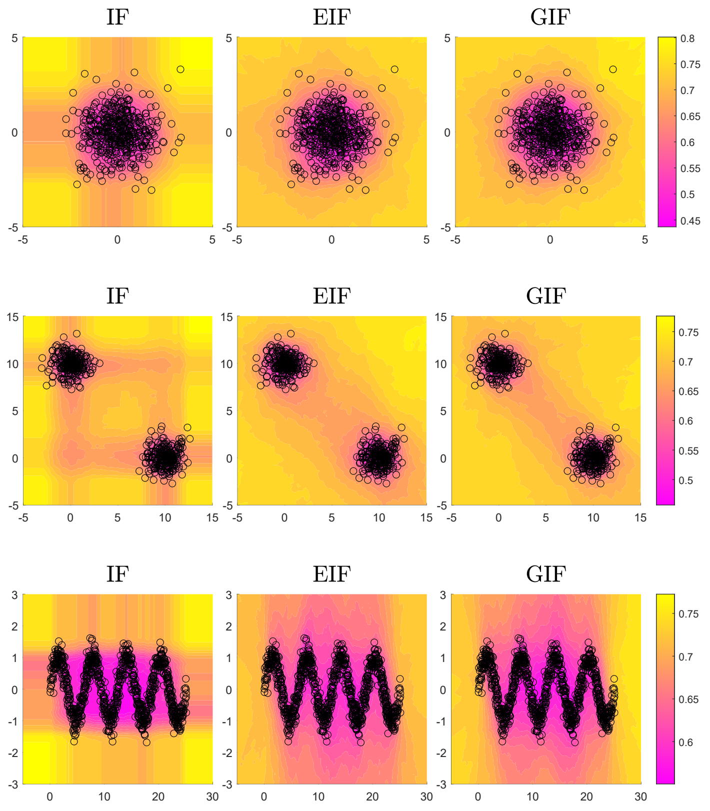


Fig. 5: Heat maps for the three algorithms and the three datasets: IF, EIF, GIF from left to right, and single blob, dual blob and sinusoidal from top to bottom. Pink values correspond to low anomaly scores and yellow to high.

3.2. Benchmark datasets

This section evaluates the performance of GIF on the datasets investigated in Hariri et al. (2019)[1] and Goldstein (2015). Note that the different datasets are described in Table 2 and are ranked in increasing order regarding the anomaly proportion (datasets in italic are those used in Hariri et al. (2019)). Since

Table 2: Datasets used in the experiments.

Name	Samples n	Features d	Anomalies
Pen Local	6724	16	0.15%
Forest Cover	286048	10	0.96%
Speech	3686	400	1.65%
Shuttle	46464	9	1.89%
Mammography	11183	6	2.32%
Breast Cancer	367	30	2.72%
Aloi	50000	27	3.02%
ANN Thyroid	6916	21	3.61%
Letter	1600	32	6.25%
Cardio	1831	21	9.60%
Pen Global	809	16	11.12%
Satellite	6435	36	31.64 %
Ionosphere	351	33	35.90 %

IF-based anomaly detectors include some randomness due to the way the trees are built, Monte-Carlo simulations (using 100 iterations) were performed for all the datasets and the three methods (IF, EIF and GIF) to compute the average area under the curve (AUC) for both receiver operational characteristics (ROC) and precision recall (PR) curves, as well as quantiles $\alpha/2$ and $1 - \alpha/2$ where $\alpha = 5\%$ in order to obtain 95% confidence intervals. The results are gathered in Fig. 6 for the ROC and in Fig. 7 for PR curves. Note that the computations were

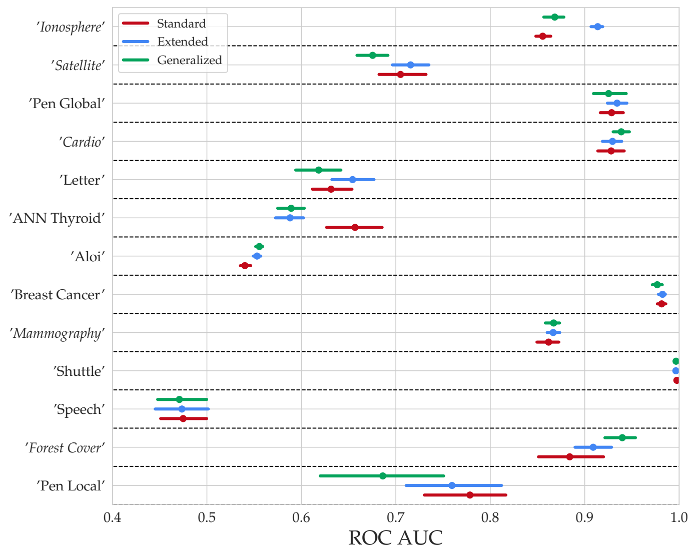


Fig. 6: Comparison of ROC AUC for several datasets (a line represents a 95% confidence interval and a dot the corresponding mean).

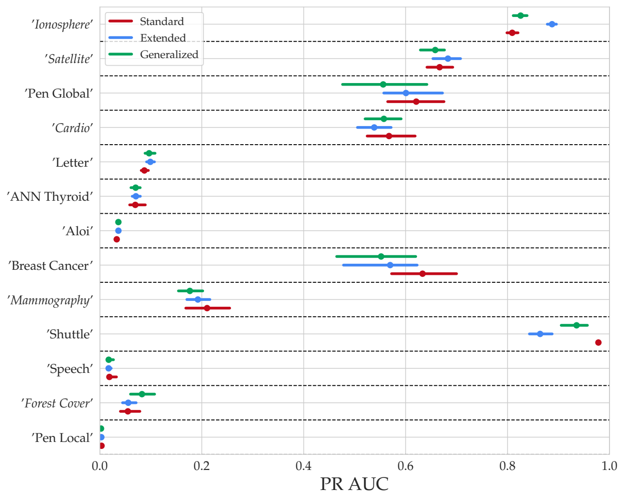


Fig. 7: Comparison of PR AUC for several datasets (a line represents a 95% confidence interval and a dot the corresponding mean).

made using Python, with the IF algorithm from scikit learn[2], EIF from the author's github[3], and our own implementation of GIF. The whole code as long as the datasets are available on the first author's webpage[4]. Note that all datasets have been preprocessed in order to obtain zero mean and unit variance for each feature. This preprocessing is not necessary for IF because splittings are made along a single feature. However, they are useful for EIF ad GIF especially in high dimensions since these algorithms are sensitive to scaling.

As one can see, there is not a significant difference between EIF and GIF in terms of ROC AUC, except for the datasets Ionosphere, Satellite, Pen Global, and Letter, where EIF seems to give a better result, and datasets Forest Cover and Cardio in favor of GIF. EIF and GIF also provide good results when compared to IF, except for the dataset ANN Thyroid. Regarding PR AUC, EIF and GIF seem to have the same behavior, except for datasets Forest Cover and Shuttle, where GIF outperforms EIF. Finally note that EIF performs better than IF and GIF for the dataset Ionosphere. From these experiments, we conclude that the performances of EIF and GIF are globally similar. In order to appreciate the interest of GIF, we have compared the execution times of the different algorithms, i.e., the time required to produce the forest (for both EIF and GIF) and the average proportion of external nodes at the maximum depth among all the external nodes computed for all the trees of a forest. The results are shown in Figs 8 and 9.

As one can see, the times to compute the forests are significantly smaller for GIF compared to EIF, with generally smaller confidence intervals. The mean proportion of limit nodes among all the external nodes shows the capability of the method to isolate data. Indeed, an external node is either due to a reach of the given maximal depth, or to an isolated data. Therefore, if this number is close to one, few data are isolated

[1]The datasets can be downloaded from `http://odds.cs.stonybrook.edu/`

[2]`https://scikit-learn.org/stable/`

[3]`https://github.com/sahandha/eif`

[4]`http://perso.tesa.prd.fr/jlesouple/codes.html`

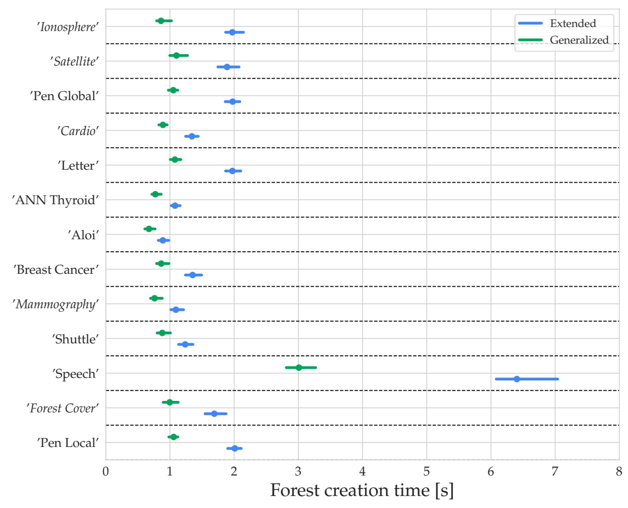


Fig. 8: Comparison of EIF and GIF computation times for several datasets (a line represents a 95% confidence interval and a dot the corresponding mean).

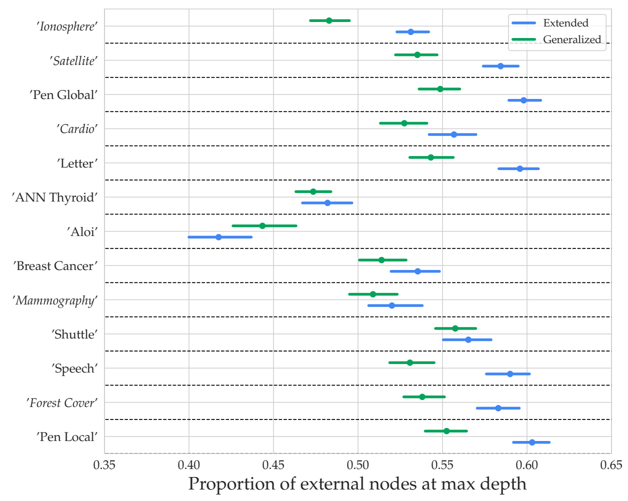


Fig. 9: Comparison of external nodes at maximum depth proportion for several datasets (a line represents a 95% confidence interval and a dot the corresponding mean).

(and conversely, the lower the mean proportion of limit nodes, the more data are isolated by the method). As the purpose of IF methods is precisely to isolate data, this ratio should be as low as possible. As one can see in Fig. 9, GIF leads to smaller proportions of this ratio than EIF (except for the Aloi dataset), which was expected.

3.3. *Anomaly scores*

To assess the robustness of the EIF to the artefacts presented in Fig. 1, authors in Hariri et al. (2019) proposed to analyze the anomaly scores of the algorithm when applied to isotropic Gaussian data. Indeed, for such data, the anomaly score should remain almost the same for data located at the same distance from the mean. To extend these experiments to the proposed method, IF, EIF and GIF were trained on the 2D single blob synthetic dataset, and testing points were generated around constant radii, as shown in Fig. 10. The mean scores versus constant radius and the corresponding standard deviations are plotted for the various algorithms in Fig. 11a.

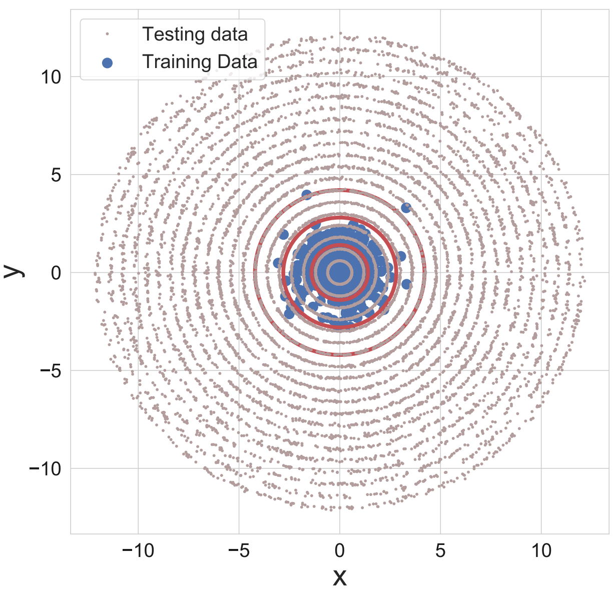
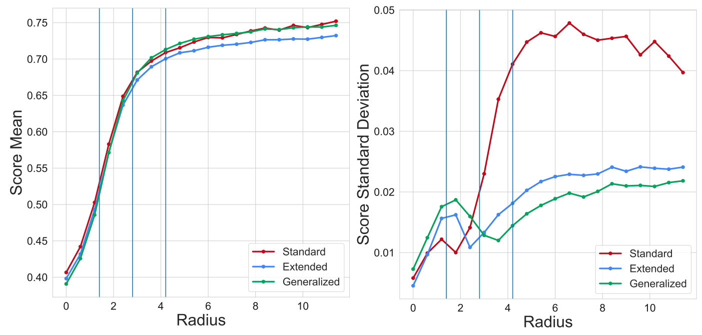


Fig. 10: Single blob with additional concentric testing data to compute mean statistics for a given distance to the blob center. The red circles represent 1, 2 and 3 data standard deviations.

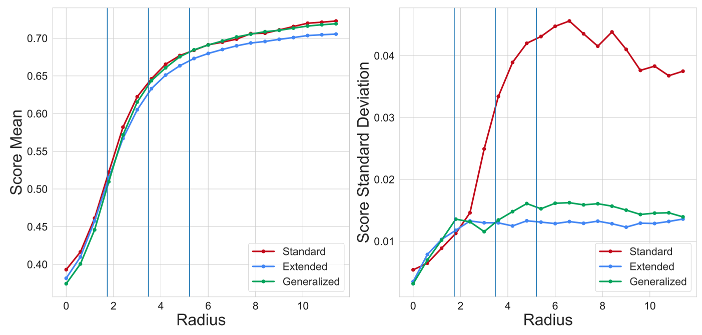
As one can see, the anomaly scores are equivalent for all the algorithms: there is a fast increase from zero to a value in the interval (2, 3) when the radius increases, and slower variations afterwards. One can observe that the standard deviations of the scores are significantly larger for IF than for EIF and GIF, which is explained by the absence of the "cross" effect for this dataset. These results were already shown in Hariri et al. (2019) and are repeated here to show that the proposed GIF performs similarly to EIF, with the advantage of being faster, thanks to the absence of empty branches in the trees. The same experiments were run on a 3D blob and a 4D blob, as shown in Figs. 11b and 11c leading to the same conclusions.
The convergence of the mean anomaly scores was also studied, as in Hariri et al. (2019) . The average anomaly scores for the inner an outer shell of each blob and the corresponding standard deviations were computed for each blob for various numbers of trees in the forest. The results are depicted in Figs. 12a, 12b and 12c for the 2D, 3D and 4D blobs respectively. As one can see, EIF and GIF provide similar results, with lower standard deviations when compared to the standard IF algorithm. Moreover, the anomaly scores for EIF and GIF seem to converge to a constant value using a relatively small number of trees (around 100 trees in each forest).

4. Conclusion

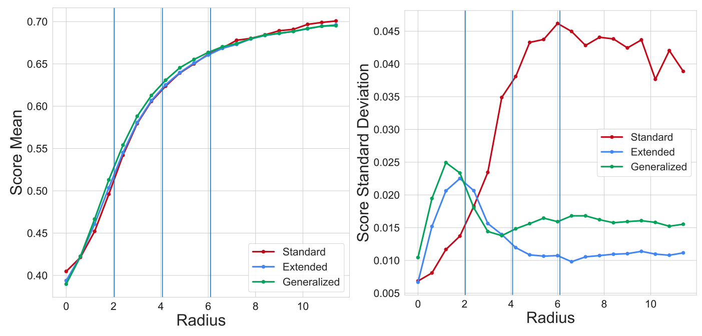
This letter studied a new isolation forest algorithm referred to as generalized isolation forest for anomaly detection. This algorithm allows some artefacts of isolation forest to be bypassed and produces trees without empty branches, which is a drawback of the extended isolation forest (EIF) algorithm. Experimentations on both synthetic and benchmark datasets allowed



(a) 2D blob.



(b) 3D blob.



(c) 4D blob.

Fig. 11: Mean anomaly scores (left) and corresponding standard deviations (right) for the various algorithms versus the radius. The vertical blue lines represented the 1, 2 and 3 standard deviations.

us to evaluate the performance of the proposed method, which is similar to that obtained with EIF. However, the proposed algorithm has a significantly reduced execution time when compared to EIF, and requires few parameters to store (a threshold at each node for GIF versus an intercept vector for each node for EIF). Future work will consider active learning and the injection of user feedback into the anomaly detectors to reduce the false alarm rate and improve anomaly detection.

References

Brause, R., Langsdorf, T., Hepp, M., 1999. Neural Data Mining for Credit Card Fraud Detection, in: Proc. Int. Conf. on Tools with Artificial Intelligence, pp. 103–106.

Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J., 2000. LOF: Identifying Density-Based Local Outliers, in: Proc. Int. Conf. on Management of Data (SIGMOD), Dallas, Tx. pp. 93–104.

Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly Detection: A Survey. ACM Computing Surveys 41, 15:1–15:58.

Dutta, J.K., Banerjee, B., 2019. Comparison of Sparse Coding-based versus Traditional Outlier Detection Methods. Pattern Recognition Letters 122, 99–105.

Goldstein, M., 2015. Unsupervised Anomaly Detection Benchmark. URL: `https://doi.org/10.7910/DVN/OPQMVF`, doi:10.7910/DVN/OPQMVF.

Hariri, S., Kind, M.C., Brunner, R.J., 2019. Extended Isolation Forest. IEEE Trans. Knowl. Data Eng. , 1–1.

İnkaya, T., Kayalgil, S., Özdemirel, N.E., 2015. An Adaptive Neighbourhood Construction Algorithm Based on Density and Connectivity. Pattern Recognition Letters 52, 17–24.

Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A., 2009. LoOP: Local Outlier Probabilities, in: Proc. Int. Conf. on Information and Knowledge Management (CIKM), Hong-Kong, China. pp. 1649–1652.

Leach, M.J.V., Sparks, E.P., Robertson, N.M., 2014. Contextual Anomaly Detection in Crowded Surveillance Scenes. Pattern Recognition Letters 44, 71–79.

Liu, F.T., Ting, K.M., Zhou, Z.H., 2008. Isolation Forest, in: Proc. Int. Conf. on Data Mining (ICDM), Pisa, Italy. pp. 413–422.

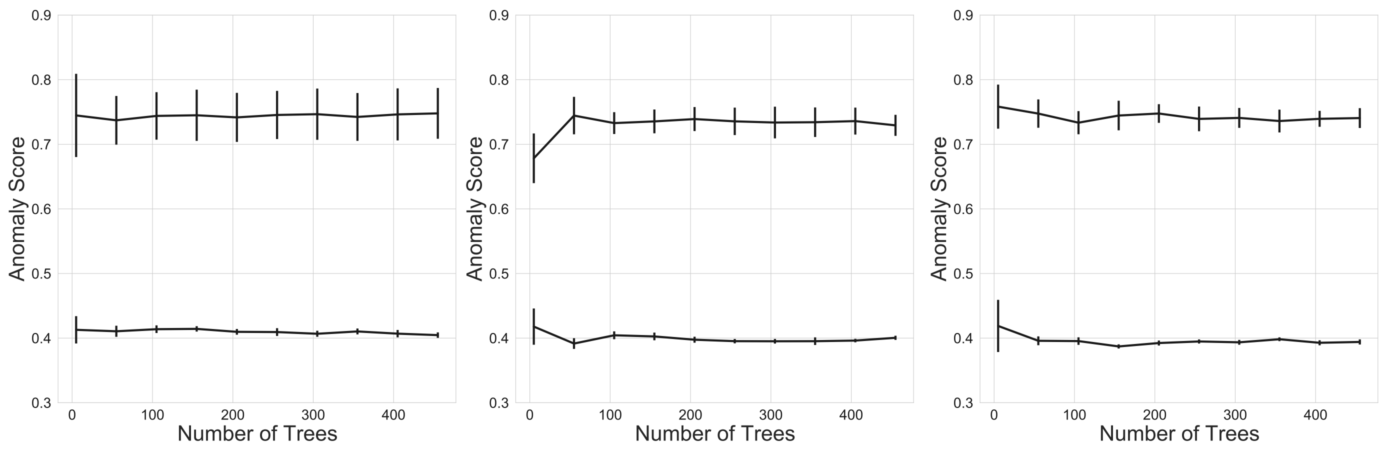
Muller, M.E., 1959. A Note on a Method for Generating Points Uniformly on *N*-Dimensional Spheres. Commun. ACM 2, 19–20.

Pilastre, B., Boussouf, L., d'Escrivan, S., Tourneret, J.Y., 2020. Anomaly Detection in Mixed Telemetry Data Using a Sparse Representation and Dictionary Learning. Signal Processing 168, 107474.

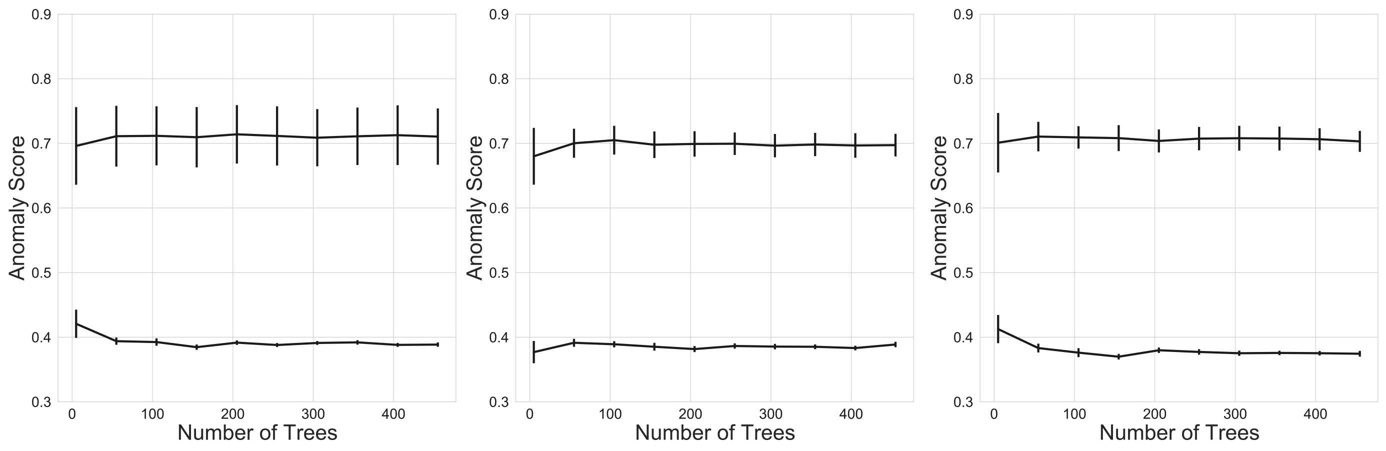
Schölkopf, B., Platt, J.C., Shawe-Taylor, J.C., Smola, A.J., Williamson, R.C., 2001. Estimating the Support of a High-Dimensional Distribution. Neural Computation 13, 1443–1471.

Tax, D.M., Duin, R.P., 2004. Support Vector Data Description. Machine Learning , 45–66.

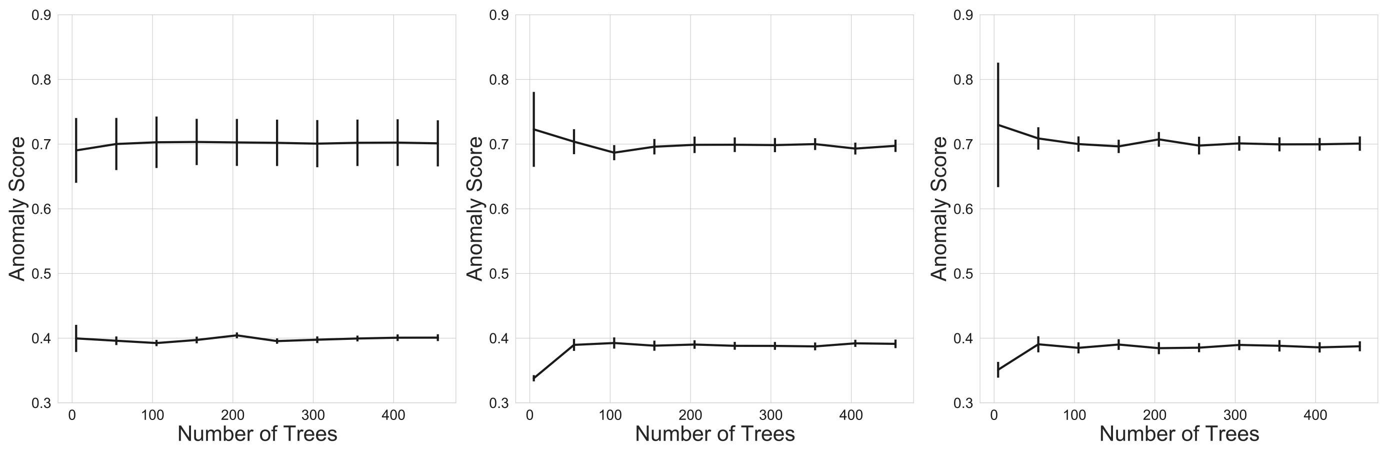
Yairi, T., Takeishi, N., Oda, T., Nakajima, Y., Nishimura, N., Takata, N., 2017. A Data Driven Health Monitoring Method for Satellite Housekeeping Data Based on Probabilistic Clustering and Dimensionality Reduction. IEEE Trans. Aerosp. Electron. Syst. 53, 1384–1401.



(a) 2D blob.



(b) 3D blob.



(c) 4D blob.

Fig. 12: Mean score for inner (bottom) and outer (top) shells versus the number of trees in the forest with corresponding standard deviations (vertical lines) for IF (left), EIF (center) and GIF (right).