A Systematic Approach for Generalizing Isolation Forest

1st Benjamin Schneider
Department of Computer Science
University of Manitoba
Winnipeg, Canada
schnei19@myumanitoba.ca

2nd Brett Downey
Department of Computer Science
University of Manitoba
Winnipeg, Canada
downeyb1@myumanitoba.ca

3rd Ryan Petrillo
Department of Computer Science
University of Manitoba
Winnipeg, Canada
petrillr@myumanitoba.ca

4th Denys Popov
Department of Computer Science
University of Manitoba
Winnipeg, Canada
popovd@myumanitoba.ca

***Abstract* — In this research paper, we present a systematic approach to modifying the Isolation Forest algorithm for anomaly detection. Our approach is based on the identification of common properties that characterize successful variations of Isolation Forest. We propose two new generalizations of the algorithm. The Middle Method and Feature Bias, which are designed to detect different kinds of anomalous data and demonstrate that these generalizations conform to the identified properties. Through experiments on real-world data, we show that our generalized methods can significantly improve the performance of Isolation Forest compared to the basic implementation. Overall, our work provides a foundation for understanding the fundamentals of Isolation Forest and introduces novel ways to extend the algorithm for improved performance in specific application domains.**

Keywords* — *anomaly detection, Isolation Forest algorithm, anomalous data, anomaly patterns, performance improvement, clustering, anomaly scores

I. Introduction

Anomaly detection is a problem that arises in many contexts. These application areas include cybersecurity, finance, healthcare, and manufacturing [1]. There are a variety of approaches to anomaly detection. Most approaches try to represent the dataset and then report anomalies as data that is dissimilar from the generated model's representation of the distribution. However, these methods often require significant information about the distribution of the dataset. For example, k-means based anomaly detection methods still require knowledge about the number of clusters in the dataset to work effectively. The *Isolation Forest* utilizes the fundamental scarcity of anomalous data to characterize what data is anomalous [2]. Since its initial publication in 2008, many authors have presented various ways to extend Isolation Forest. Most variations of Isolation Forest create an ensemble of *iTree* estimators that are used to generate anomaly scores. In this paper, we seek a more systematic approach to modifying Isolation Forest. Therefore, we start by inspecting the common properties that characterize successful variations of Isolation Forest. Although these properties are not sufficient for creating an innovative variation of Isolation Forest, they provide an important foundation for understanding the fundamentals of the algorithm. Isolation Forest also has very fast convergence, resulting in training that requires minimal data compared to other methods. However, Isolation Forest has limited hyperparameters that can be used to customize the model on specific use cases. Once the model has sufficient ensemble size and tree height, the only parameter a user can tune is the anomaly score threshold. This hyperparameter is a value representing the model's sensitivity. We continue Isolation Forest's objective to be anomaly focused. We introduce hyperparameters to help users apply knowledge of anomalies within their domain to create performant anomaly detection models. Just like non-anomalous data is highly regular, we expect that patterns exist within anomalous data in application domains. *Intuitively, we expect an anomaly like a fraudulent credit card transaction to have similarities with other fraudulent credit card transactions*. By introducing ways for the Isolation Forest model to leverage information about the nature of anomalies in specific application domains, we achieve better performance compared to generic implementations of Isolation Forest. Our novel contributions are the following:

- We present and explain several properties required for effective generalizations of the *iTree* algorithm.
- We create two new generalizations on the *iTree* algorithm for detecting different kinds of anomalous data
- We prove that our generalizations conform to the earlier defined properties
- We show these generalized methods can be applied to real-world data to improve performance over the basic implementation of Isolation Forest

II. Background and Related Works

In this section, we go over algorithms and terminology that are relevant to this paper. This includes defining the fundamental algorithms used in Isolation Forest. We also introduce several important parameters from the original version of Isolation Forest that control the size and shape of the forest data structure. Finally, we introduce equations for scoring inputs and mathematics used for proofs in this paper.

A. Background

This paper expands upon the original Isolation Forest algorithm by modifying how points are isolated in the *iTree* algorithm. The original version of Isolation Forest chooses a feature uniformly randomly, then chooses the split point uniformly randomly in the minimum to the maximum range of that feature. This process is done recursively until all points are isolated at the node, or the maximum height is reached [3].

Algorithm 1 : *iTree(X, e, l)*

Inputs: X - input data, e - current tree height, l - height limit
Output: an iTree

```
1:  if $e \geq l$ or $|X| \leq 1$ then
2:      return exNode{Size ← $|X|$}
3:  else
4:      let $Q$ be a list of attributes in $X$
5:      randomly select an attribute $q \in Q$
6:      randomly select a split point $p$ from max and min
        values of attribute $q$ in $X$
7:      $X_L \leftarrow filter(X, q < p)$
8:      $X_R \leftarrow filter(X, q \geq p)$
9:      return inNode{Left ← iTree($X_L$, $e + 1$, $l$),
10:                   Right ← iTree($X_R$, $e + 1$, $l$),
11:                   SplitAtt ← $q$,
12:                   SplitValue ← $p$}
13: end if
```

This process constructs an *iTree*. Which is a data structure that represents this spatial partitioning method by having each internal node of the tree represent each split around a feature. The leaves in an *iTree* represent isolated points that were used to construct the tree. For computational simplicity, the *iTree* is often set to have a maximum height *l*. Crucially, due to the inherent scarcity of anomalous data the spatial partitioning in each *iTree* tends to create shallow leaves corresponding to anomalous data. This is the property that Isolation Forest exploits to detect anomalies. Our paper uses the same ensemble of *iTrees* as the original Isolation Forest paper to construct the *iForest* [3]. Our modifications will modify the *iTree* subroutine that is called by *iForest*.

Algorithm 2 : *iForest(X, t, ψ)*

Inputs: X - input data, t - number of trees, ψ - sub-sampling size
Output: 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:     $X^i \leftarrow sample(X, \psi)$
5:     $Forest \leftarrow Forest \cup iTree(X^i, 0, l)$
6: end for
7: return Forest
```

We also adopt the recommended ensemble size $t = 100$ and sub-sample size $\psi = 256$ for all examples and applications in this paper [2]. Anomaly scores will be calculated using the same *pathLength* algorithm and anomaly scoring calculation as in the original version of Isolation Forest [3].

Algorithm 3 : *PathLength(x, T, e)*

Inputs : x - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
Output: path length of x

```
1: if $T$ is an external node then
2:     return $e + c(T.size)$ {$c(.)$ is defined in Equation 1}
3: end if
4: $a \leftarrow T.splitAtt$
5: if $x_a < T.splitValue$ then
6:     return PathLength($x$, $T.left$, $e + 1$)
7: else {$x_a \geq T.splitValue$}
8:     return PathLength($x$, $T.right$, $e + 1$)
9: end if
```

C(·) is the average path length of an unsuccessful binary search in an *iTree* [3]

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

H(n) is an approximation for the n_{th} harmonic number [3].

$$H(n) \approx \ln(i) + 0.57721566492 \tag{2}$$

Anomaly scores for a given point are calculated as follows [3]:

$$S(x) = 2^{-\frac{E(h(x))}{c(\psi)}} \tag{3}$$

Where E(·) is the average across all *iTrees* in the *forest* and h(·) is the height of point x in an *iTree* given by algorithm 3. S(x) is the outputted anomaly score for a point that is compared to a threshold value to decide whether a point is anomalous or nominal. We use the term *anomalous* to refer to points that are irregular when compared to the distribution of the dataset the model is trained on. It is important to note that what is considered anomalous is domain dependent. In different application contexts, what points are ideally classified as anomalous differ. We use the term *nominal* to refer to data that matches the distribution of the greater dataset.

We use several basic facts about continuous distributions in the proofs in this paper. In particular, we use the fact that

$$P(a \leq X \leq b) = \int_a^b f(x)\, dx \tag{4}$$

For any continuous distribution *f*(x) and its corollary [4]:

$$P(a \leq X \leq a) = \int_a^a f(x)\, dx = 0 \tag{5}$$

That is, the probability of generating a value that is exactly equal to some value a is 0. This result follows from applying the fundamental theorem of calculus to equation 4 [5].

$$P(a \leq X \leq a) = \int_a^a f(x)\, dx$$
$$= F(a) - F(a) = 0$$

This is used in section VI to discard cases where two randomly generated points are exactly equal, as those cases are 0 probability events.

B. Related Works

The purpose of this section is to discuss the different modifications that have been made to Isolation Forest. How the Isolation Forest application can be applied to different types of data, and to detail the different types of anomaly detection methods that exist. We will first discuss the differences and improvements that other Isolation Forest versions have focused on. Then we will discuss alternative anomaly detection methods and how they differ from Isolation Forest.

Work has been done on a variety of methods for extending Isolation Forest. Other versions of Isolation Forest modify how data is partitioned at each node during the construction of an *iTree*. *Extended Isolation Forest* (EIF) modifies Isolation Forest to create a uniform anomaly scoring along all feature directions [6]. There have also been other variations on Isolation Forest that use statistical methods to improve performance by increasing sampling in low density areas of the distribution. *Probabilistic Generalization of Isolation Forest* (PGIF) uses information about the distribution and density to choose split points between clusters [7]. Due to the versatility of the algorithm, there are many ways in which Isolation Forest has been modified. Combining Isolation Forest with techniques from mathematics and clustering has produced several novel and effective variations of Isolation Forest. Karczmarek, Paweł, *et al*. [8][9] found that combining Isolation Forest with K-means and fuzzy C-means to be an effective way to construct *iTrees* in Isolation Forest. As Isolation Forest is fundamentally an ensemble of simple *iTrees*, there are many ways to construct different variations of Isolation Forest by changing how *iTrees* are constructed. Other modifications entirely replace the way an *iTree* is constructed. For example, Galka *et al*. [10] proposes an Isolation Forest modification that merges *iTree* nodes by making use of minimal spanning trees.

As many applications require real-time monitoring of anomalies, there has also been work to modify Isolation Forest to process and detect anomalies in live data streams, and to also improve the efficiency of Isolation Forest data stream anomaly detection. Additionally, streaming data is infinite and can vary over time, which presents a challenge for the original Isolation Forest as a predetermined contamination parameter or anomaly threshold will not work and must be overcome. Laskar *et al*. [11] applied Isolation Forest in an industrial big data stream environment searching for network security anomalies using K-Means to eliminate the requirement for a contamination parameter. Other techniques have been developed to allow Isolation Forest to handle data streams, such as using Mondrian decision trees to create a hybrid Isolation Mondrian Forest in order to handle data stream batches [12]. These data stream modifications allow Isolation Forest to be effectively applied to more application domains that require real-time anomaly detection on continuous streams of data.

Isolation Forest is an isolation-based anomaly detection algorithm [3]. Other anomaly detection algorithms approaches include statistical, depth-based, distance-based, clustering-based, deviation-based, and others [1]. Isolation-based anomaly detection is unique from all these approaches by focusing on isolated data points. Some versions of Isolation Forest can incorporate aspects of other anomaly detection approaches such as statistical methods when generating *iTrees*, however the key concept of isolated data points is what differs isolation-based anomaly detection from other anomaly detection methods.

III. Partitioning Schemes for multiple cluster Gaussian distributions

When creating new methods for partitioning it is important to understand the properties that create effective Isolation Forest methodologies. In this section we define and justify important properties for creating effective partitioning methods in Isolation Forest.

1. $|X_l| > 0 \; and \; |X_r| > 0$
2. *For any two points $x_1 \neq x_2$, there exists an ITree such that the leaf containing x_1 is not same as the leaf that contains x_2*
3. *Path length of anomalous data is statistically shorter compared to nominal data*
4. *The partitioning method splits the feature where the distribution of feature is sparse with high probability*

Property 1: $|X_l| > 0 \; and \; |X_r| > 0$

The idea behind *iForest* is to split the data until each datapoint is isolated in a leaf or maximal height is reached [13]. If this property doesn't hold it could allow for empty branches in the constructed ITree. As the scoring of points corresponds to the depth of a point in ensembles of *ITrees*, empty branches distort the evaluation of points in an *ITree*.

Property 2: *For any two datapoints $x_1 \neq x_2$, there exists an ITree such that the leaf containing x_1 is not same as the leaf that contains x_2*

This property shows that our scheme can isolate all points. That is, if we continue our partitioning process, any two points would eventually end up isolated in leaves. If this property does not hold, that means our algorithm cannot distinguish between certain sets of points [14]. As *iTrees* are spatial partitioning data structures, we need to be able to partition between any two non-identical points in the feature space.

Property 3: *Path length of anomalous data is statistically shorter compared to nominal data*

As path length dictates anomaly scoring in Isolation Forest, it is important that data that is anomalous tends to have shorter paths in *iTrees*. This results in higher anomaly scores as calculated by equation 3.

Property 4: *The partitioning method creates partitions in sparse areas of the feature space with high probability*

This property allows for generalization of the model to multi-cluster datasets. If a dataset has multiple clusters, then it is important that ITrees create partitions between the clusters [7]. If an algorithm does not do this, then the distribution of larger clusters will dictate what points are considered anomalous in smaller clusters. This results in poor generalization to multi-cluster distributions. Anomalies should be calculated with respect to the clusters that the point is close to. That is, we want our anomaly detection method to operate *locally* not *globally*. For example, consider the following case of a small cluster and a large cluster:

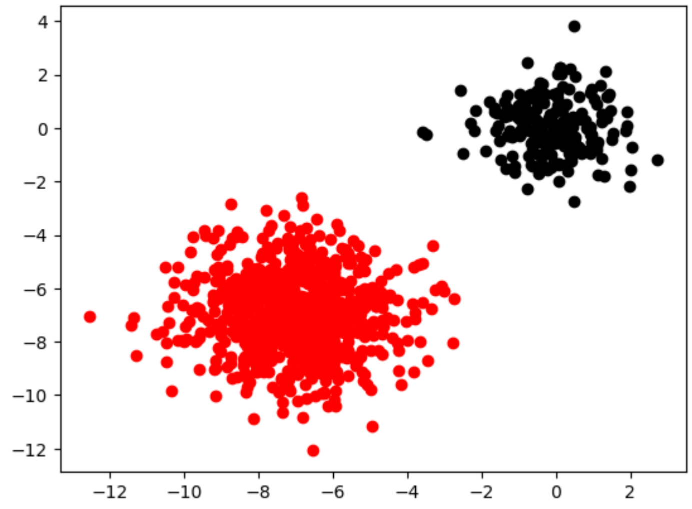

Fig. 1. Two disjoint clusters of different sizes

If we choose a partitioning method that tends to generate split points in more dense areas of the dataset, then we will almost always split in the red cluster because the red cluster in much denser than the black cluster. To create a model that learns the features of both clusters, we need our *iTrees* to generate split points in the sparse regions between clusters with significant probability.

These properties form the basis for creating novel ways for generating *iTrees*. Any method that does not satisfy these properties will struggle to create *iTrees* that work as individual estimators in the *iForest* ensemble. Using these properties as a foundation for what makes effective *iTree* estimators we created the methods discussed in sections IV and V.

IV. THE MIDDLE METHOD

In this section, we consider how to create a model that defines anomalous data as points that are outliers in multiple feature directions (multi-directional anomalies). However, we also wanted the technique to demonstrate localized behavior. We want points to be considered anomalous if they are multi-directional anomalies with respect to near clusters in the dataset. To accomplish this, we created a version of Isolation Forest that chooses the split point in a constrained range around the middle value of the feature in the data partition. We introduce a hyperparameter $\alpha \in [0,1]$ to control how strongly the model prefers multi-directional anomalies. The split point is calculated through uniform sampling of the following interval.

$$\left[\frac{h+l}{2} - \alpha(h-l)/2,\ \frac{h+l}{2} + \alpha(h-l)/2\right] \qquad (4)$$

Where l and h are the minimum and maximum values of the feature in the data sample.

If $\alpha = 1$ the model has no preference for multi-directional anomalies. If $\alpha = 0$ then the model demonstrates a strong preference towards classifying points as anomalies if they are anomalous in multiple feature direction.

Algorithm 4 : *Middle Method iTree*(X, e, l, α)

Inputs: X - input data, e - current tree height, l - height limit, α - multi-directional anomaly bias
Output: an iTree

```
1:  if $e \geq l$ or $|X| \leq 1$ then
2:      return exNode{Size ← $|X|$}
3:  else
4:      let $Q$ be a list of attributes in $X$
5:      randomly select an attribute $q \in Q$
6       $min$ ← the minimum value of feature $q$ in X
7:      $min$ ← the minimum value of feature $q$ in X
        $p$ ← uniformly sample from
    $[\frac{max + min}{2} - \alpha(max - min)/2, \frac{max + min}{2} + \alpha(max - min)/2$
8:       $X_l$ ← filter($X$, $q < p$)
9:       $X_r$ ← filter($X$, $q \geq p$)
10:      return inNode{Left ← iTree($X_l$, $e + 1$, $l$, $\alpha$),
11:                    Right ← iTree($X_r$, $e + 1$, $l$, $\alpha$),
12:                    SplitAtt ← $q$,
13:                    SplitValue ← $p$}
14: end if
```

When visualizing this method, we see a clear tendency for the model to evaluate points that are outliers in a combination of feature directions to be classified as outliers. By adding multiple clusters to the dataset, we can demonstrate that the model has localized behavior. We choose 2 dimensions, X and Y for ease of visualization. The dataset is generated through the following methodology.

1. $fix\ u_1 = \begin{pmatrix}0\\0\end{pmatrix},\ u_2 = \begin{pmatrix}-7\\-7\end{pmatrix},\ \Sigma = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$
2. Define Multivariate Gaussian distributions $N_1(u_1,\ \Sigma)$ and $N_2(u_2,\ \Sigma)$
3. Generate dataset D by sampling $N_1(u_1,\ \Sigma)$ 15000 times and $N_2(u_2,\ \Sigma)$ 20000 times

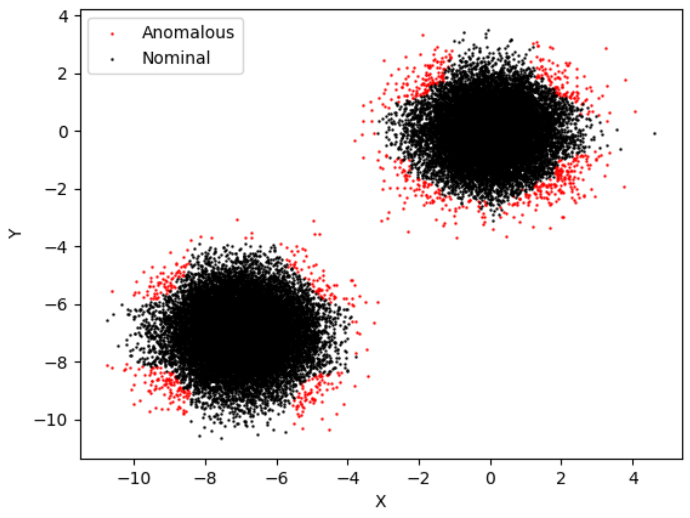

Fig. 2. Middle Method with anomaly threshold of 0.563 and α = 0)

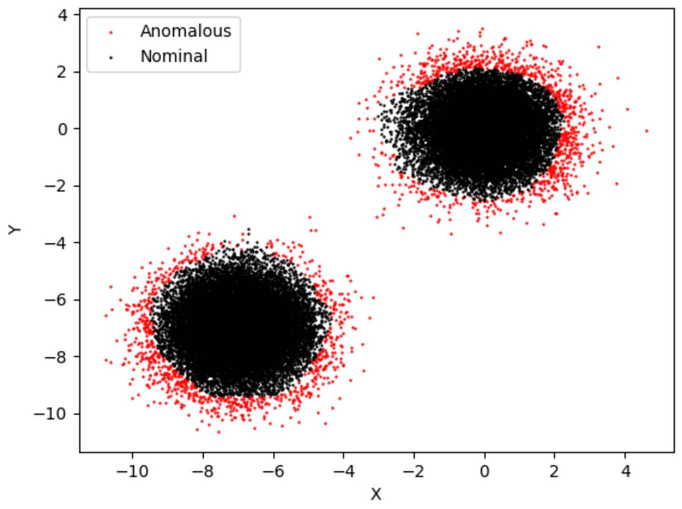

Fig. 3. Scikit-Learn's implementation of Isolation Forest with contamination = .05

As can be seen from the visualization, our model has been trained to consider points in the X-Y feature direction more anomalous than points that are only outliers in a single feature. This is not a capability offered by the Scikit-learn implementation of the Isolation Forest algorithm. Although this behaviour is consistent with normally distributed clusters, this method is unlikely to be effective for datasets with non-normally distributed clusters.

V. FEATURE BIAS METHOD

By statistically weighting which features are used to generate partitions at each node in the *iTree*, we have found that we can control which features are most impactful when the model is scoring inputs. A feature q that is weighted with higher than uniform probability will result in the model being more sensitive to inputs that are outliers in feature q. Intuitively, the average constructed *iTree* will have more nodes where q is selected as the partition feature, leading to inputs that are outliers in q to be isolated in shallower paths in the *iTree*. This leads to higher anomaly scores for these inputs when they are scored against the trained model. The addition of weighted features allows for a much higher degree of customization when training the model. Depending on the application domain, certain features are more impactful when deciding whether inputs are anomalies. Through the introduction of a weighting hyperparameter $w = (w_1, w_2, \dots w_N)$ such that $\sum_{i=1}^{N} w_i = 1$ *and* $w_i > 0$ we create a much more versatile version of Isolation Forest. Our version of Isolation Forest allows for features with different levels of importance to be included in the model. In Isolation Forest, all features are equally weighted, this results in poor performance if features in the model have varying importance for identifying anomalies.

Algorithm 5 : *Feature Bias iTree(X, e, l, w)*

Inputs: X - input data, e - current tree height, l - height limit, $w = (w_1, w_2, \dots w_N)$ - feature weights
Output: an iTree

```
1:  if $e \geq l$ or $|X| \leq 1$ then
2:      return exNode{Size ← |X|}
3:  else
4:      let $Q$ be a list of attributes in $X$
6:      select an attribute $q \in Q$ such that
        P($q = q_i$) = $w_i$ where $q_i$ is the $i_{th}$ feature of Q
7:      randomly select a split point $p$ from max and min
        values of attribute $q$ in $X$
8:      $X_l$ ← filter($X$, $q < p$)
9:      $X_r$ ← filter($X$, $q \geq p$)
10:     return inNode{Left ← iTree($X_l$, $e + 1$, $l$, $w$),
11:                   Right ← iTree($X_r$, $e + 1$, $l$, $w$),
12:                   SplitAtt ← $q$,
13:                   SplitValue ← $p$}
14: end if
```

Consider the case of a dataset with two features, X and Y. Suppose we know that inputs that are outliers in feature X are more likely to be anomalies. In conventional implementations of Isolation Forest, we have no way to represent this knowledge within the model. To use this knowledge would require pre-processing of the dataset. However, in our model, we can represent this knowledge by setting $w = (.75, .25)$ (Fig. 4). Furthermore, as w becomes a model hyperparameter we can use hyperparameter search to find the ideal values for w. we generate the data for this experiment using the same methodology as in section IV.

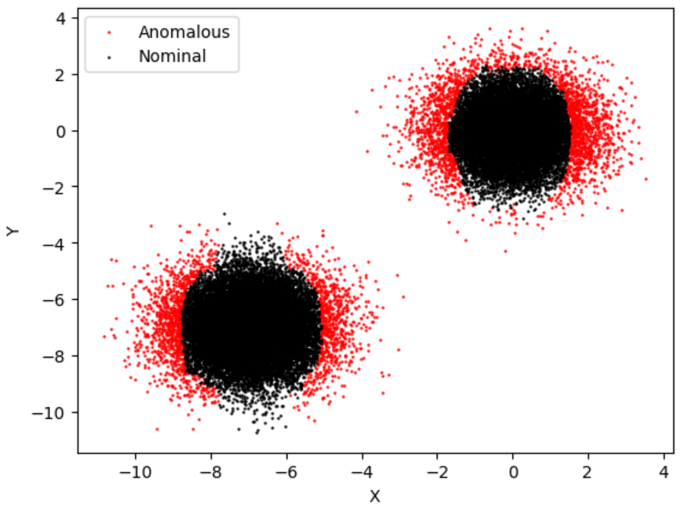

Fig. 4. Feature Bias method with $w = (.75, .25)$ and anomaly threshold = 0.51

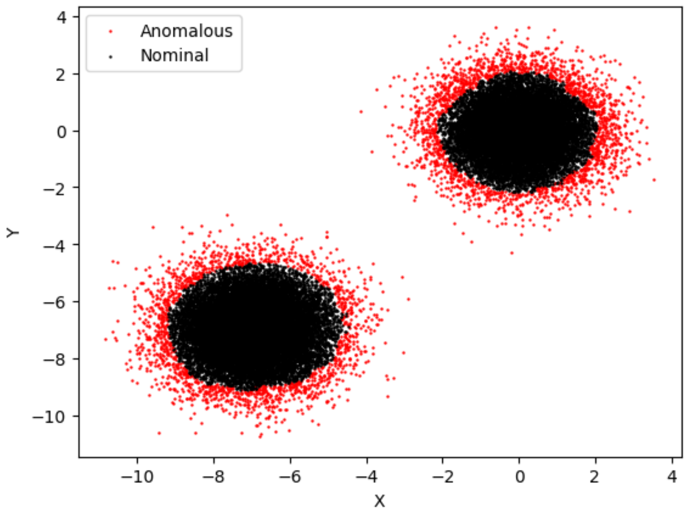

Fig. 5. Scikit-Learn's implementation of Isolation Forest with contamination = .1

As can be seen from these visualizations, our model is more sensitive to outliers in the X feature direction than the Scikit-Learn's model, as intended. As Isolation Forest is a very fast model to train, hyperparameter search on w is significantly less computationally intensive than in other machine learning methods. This method also works well on different distributions and higher dimensional data, as we show later in this paper.

VI. Analysis of the Properties of Algorithms 3 and 4

In this section, we show how both of our introduced algorithms satisfy properties 1 and 2 as defined in section III. In the following proofs, it is important to note that we assume that the dataset X is a set of vectors drawn from a continuous distribution. That is, for any two $x_i, x_j \in X$ $(i \neq j)$ the probability that $x_i = x_j$ along a feature is 0. This follows from the fact that in a continuous distribution, the probability of a given point being generated is 0. If you are applying Isolation Forest to a dataset that is sampled from a discrete domain, the following results will not hold.

Lemma 1. *Property 1 holds for Algorithm 4*

Proof. We assume that *iTree* is called with valid parameters. That is, $|X| > 0$, $e \leq l$, and $0 \leq \alpha \leq 1$.

If line 1 is true then the proof is trivial, as X_L and X_R are never generated. So, assume line 1 is false.

It follows that there exists $x_1, x_2 \in X$ such that x_1 has the minimum value of q in X and x_2 has the maximum value of q in X. As x_1 and x_2 are drawn from a continuous distribution, it follows that $x_1 \neq x_2$ in feature q.

It follows that $min < max$ from lines 6 and 7, so we define v such that $min + v = max$ $(v > 0)$.

Consider the lower bound of the sampling interval for p:

$$\frac{max + min}{2} - \frac{\alpha(max - min)}{2}$$
$$= \frac{2\,min + v}{2} - \frac{\alpha v}{2}$$
$$= min + \frac{v}{2}(1 - \alpha) \geq min$$
$$\textit{(as } 0 \leq \alpha \leq 1 \textit{ and } v > 0\textit{)}$$

Consider the upper bound of the sampling interval for p:

$$\frac{max + min}{2} + \frac{\alpha(max - min)}{2}$$
$$= \frac{2max - v}{2} - \frac{\alpha v}{2}$$
$$= max - \frac{v}{2}(1 - \alpha) \leq max$$
$$\textit{(as } 0 \leq \alpha \leq 1 \textit{ and } v > 0\textit{)}$$

$\Rightarrow p \in (min, max)$ as $P(p = \text{min}) = P(p = \text{max}) = 0$ as p is a real number sampled uniformly from the sampling interval.

$\Rightarrow x_1 \in X_L$ and $x_2 \in X_R$ as $x_1 = min$ in feature q and $x_2 = max$ in feature q and $min < p < max$.

$\Rightarrow |X_L| > 0$ and $|X_R| > 0$.

Lemma 2. *Property 1 holds for Algorithm 5*

Proof. We assume that *iTree* is called with valid parameters that is, $|X| > 0$, $e \leq l$, and $w_i > 0$ for all i in w.

If line 1 is true then the proof is trivial, as X_L and X_R are never generated. So, assume line 1 is false.

It follows that there exists $x_1, x_2 \in X$ such that x_1 has the minimum value of q in X and x_2 has the maximum value of q in X. As x_1 and x_2 are drawn from a continuous distribution, it follows that $x_1 \neq x_2$ in feature q.

It follows that $min < max$ from lines 6 and 7.

$\Rightarrow p \in (min, max)$ as $P(p = min) = P(p = max) = 0$ as p is a uniformly sampled real number from the interval $[min, max]$.

$\Rightarrow x_1 \in X_L$ and $x_2 \in X_R$ as $x_1 = min < p < max = x_2$ in feature q.

$\Rightarrow |X_L| > 0$ and $|X_R| > 0$

Lemma 3. *Property 2 holds for Algorithm 4*

Proof. let $x_1 \neq x_2 \in X$ and assume sufficiently large l such that the algorithm only terminates when all points are isolated in a leaf.

As $x_1 \neq x_2$, it follows that X_L and X_R are generated on lines 8 and 9. From lemma 1 $|X_L| > 0$ and $|X_R| > 0$.

Case 1: $x_1 \in X_L$ *and* $x_2 \in X_R$
In this case, x_1 belongs to a leaf in the subtree rooted inNode's left child and x_2 belongs to a leaf in the subtree rooted inNode's right child. As these trees have no overlap, it follows that x_1 and x_2 are contained in different leaves.

Case 2: $x_2 \in X_L$ *and* $x_1 \in X_R$
In this case, x_2 belongs to a leaf in the subtree rooted inNode's left child and x_1 belongs to a leaf in the subtree rooted inNode's right child. As these trees have no overlap, it follows that x_1 and x_2 are contained in different leaves.

Case 3: $x_1, x_2 \in X_i$ *(i,j* $\in \{L, R\}$*,* $i \neq j$*)*
As set X has finite size and $|X_j| > 0$ it follows that $|X_i| < |X|$. Set $X = X_i$, we can continually apply our initial argument because l is sufficiently large such that the algorithm only terminates if every data point is isolated at a leaf (by the recursion on line 10). Eventually, $|X| = 2$ or one of cases 1 or 2 have occurred.

If $|X| = 2$, then $X = \{x_1, x_2\}$ and generating X_L and X_R must separate x_1 and x_2 into different subtrees by lemma 1.

Lemma 4. *Property 2 holds for Algorithm 5*

Proof. let $x_1 \neq x_2 \in X$

As x_1 and x_2 are drawn from a continuous distribution, it follows that $x_1 \neq x_2$ in feature q selected on line 6. Assume without loss of generality that $x_1 < x_2$ in feature q.

As the split point p is chosen randomly uniformly there exists an *iTree* such that $x_1 < p < x_2$

$\Rightarrow x_1 \in X_L$ and $x_1 \in X_R$

$\Rightarrow x_1, x_2$ belong to different leaves in the ITree by the recursion on line 10.

Lemmas 3 and 4 prove that our algorithms can generate *iTrees* that differentiate between any two differing points given sufficiently large maximum tree height. However, these results do not guarantee that any ensemble of *ITree* estimators (an *iForest*) can differentiate between two differing points. In fact, it is quite likely that two points that are very close together will obtain the same anomaly score when evaluated using a finite size *iForest*.

VII. Analysis of Feature Bias on the HTTP Dataset

In this section, we apply both the original Isolation Forest partition scheme and the feature bias partition scheme on the HTTP KDD CUP 1999 dataset. The HTTP KDD CUP 1999 is a reduced dataset containing 567479 data points and 2211 (0.4%) outliers that was derived from the original KDD CUP 1999 simulated military environment network intrusion dataset [15]. This reduced dataset contains three basic attributes, two of which will be analyzed to detect anomalies: source bytes (src_bytes) and destination bytes (dst_bytes). Source bytes refer to the number of bytes transmitted from the web client to the web server, whereas destination bytes are the number of bytes transmitted from the web server to the web client. These basic attributes can indicate what the typical request and response sizes are for a web server. Depending on what types of operations the server supports, different clusters can exist. For example, one cluster with low values for the source and destination bytes may indicate a small GET request, whereas a different cluster with high values may indicate more computationally demanding operations (such as a download request). Anomalies that occur outside of operation clusters will have different source and destination bytes than any operation that is known to be supported by the server, which could indicate a potential problem.

An analysis of HTTP networking data can reveal a variety of use cases for detecting anomalies. These anomalies may be caused by security threats, system errors, or oversights in system design, and their detection is essential for ensuring the secure and efficient operation of networks. For example, a software bug that leads to the submission of malformed requests in specific edge cases can be difficult to identify and resolve, as it may not be immediately apparent that there is a networking issue. However, if the anomalous request is detected, it can be thoroughly investigated to determine the root cause and implement a solution. Security threats can come in the form of repeated network requests, unusually large requests which are not typically supported by the web server, and more. Lastly, anomalies may be found after implementing a system which can indicate problems in the overall architecture and design, such as redundant or repeated network requests.

HTTP networking data is complex, with a wide range of information transmitted for each request, including request type, headers, and body. When analyzing anomalies in this data, it is important to consider a wide range of features and not just focus on individual features. Hence it can be useful to apply biases to different features by applying the feature bias partition scheme. Therefore, we will compare our results with the original Isolation Forest algorithm, and the known outliers of the data set. Our analysis shows that the original Isolation Forest may not detect clusters of outliers, whereas our implementation can correctly identify the groupings of outliers by applying specific biases to the feature that the outliers are most prominent in.

We consider the comparison of the actual known outliers in the HTTP KDD CUP 1999 dataset (Fig. 6), compared to the outliers detected by Scikit-Learn's implementation of Isolation Forest (Fig. 7). To use the Sci-kit Learn Isolation Forest implementation to detect the dataset's outliers, the contamination parameter will be set to 0.004 = 0.4% (Fig. 7). This value represents the percentage of outliers in the dataset and controls how many outliers will be identified by Scikit-Learn's implementation of Isolation Forest.

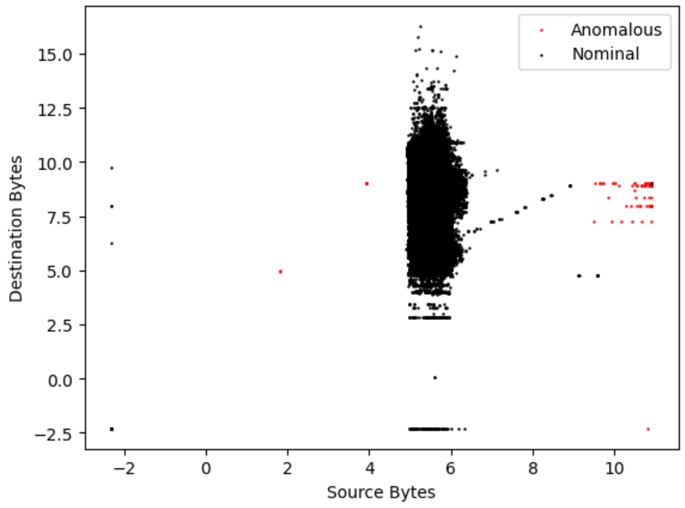

Fig. 6. Ground truth for the HTTP KDD CUP 1999 dataset

As seen in Figure 6, a large nominal cluster that contains 99% of the overall data exists in the source byte destination range [5, 6.3]. We note that there are two nominal data points far away outside of this cluster. The first is (10.906691489914584, 9.025708147644988) which is located inside of the cluster of anomalies near X=10, and which has a total of 2121 instances in the HTTP KDD CUP 1999 dataset. The second point is (-2.3025850929940455, -2.3025850929940455) which has a total of 303 instances, located as the most bottom left point.

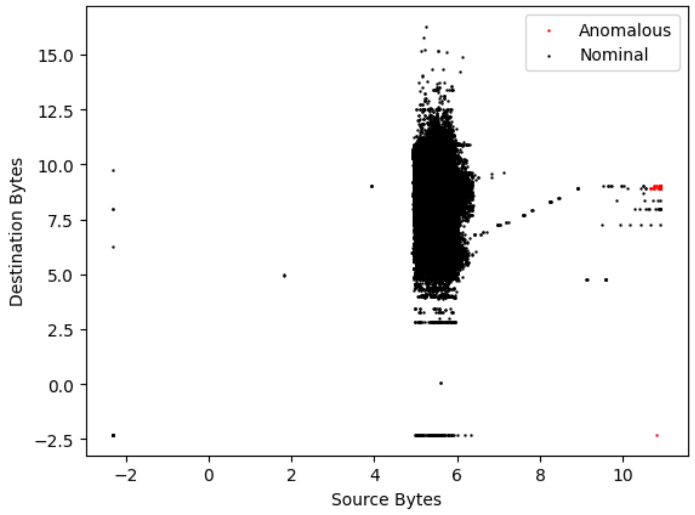

Fig. 7. Scikit-Learn's implementation of Isolation Forest with contamination = .004

Analyzing the result indicates a large cluster of anomalies that are undetected by the original implementation of Isolation Forest which are surrounded by a dense cluster of identical nominal data points. Scikit-Learn's implementation of Isolation Forest detects approximately 2207 outliers, of which the most anomalous data point (10.906691489914584, 9.025708147644988) was incorrectly predicted to be anomaly a total of 2121 times; this also accounts for every nominal instance of that data point in the HTTP KDD CUP 1999 dataset. Additionally, this data point is the most repeated row in the HTTP KDD CUP 1999 dataset (0.4% of the dataset), followed by the data point (-2.3025850929940455, -2.3025850929940455) which occurs much less frequently for a total of 303 times. As every instance of this data point was predicted to be an anomaly, this leaves approximately 100 other data points that are permitted to be predicted as anomalous, limited by the supplied contamination parameter. As the dense nominal cluster accounts for a very small percentage of the total data compared to the entire data set, it is reasonable that this data point was marked anomalous. However, this indicates that Isolation Forest is susceptible to detecting a large amount of identical nominal data as anomalous and fails to predict actual anomalous points. Increasing the contamination score would allow Isolation Forest to detect these anomalies, as the total amount of anomalies allowed are limited to approximately 2200 (around 4% of 567479). However, other large nominal groupings would incorrectly be flagged as anomalous as well and would then require an additional contamination score increase to overcome. Also, this increases the likelihood that the destination bytes feature will cause more anomalous values to be marked, instead of the source destination bytes which contains the real anomalies. Instead, we will analyze and show that the difference in anomalies detected by applying the feature bias method using the same value for alpha and using a bias on the source destination attribute can contain more anomalies being detected close to dense clusters of nominal data.

Applying the feature bias partitioning scheme increases the total amount of outliers which are detected, by a total of approximately 2700. By using the same alpha value of 0.61 applying a feature bias on the source bytes of 67%, and a bias on the destination bytes of 33%, we can use the feature bias partitioning scheme to identify the large source destination anomalies more accurately that exist near the dense cluster of nominal data (Fig. 8). This partitioning scheme however still detects the dense cluster of nominal data at (10.906691489914584, 9.025708147644988), but it successfully predicts the cluster of anomalies surrounding the nominal cluster. As 98% of all data points occur within the large cluster ranging from approximately [5.1, 6.3], and as a wide range of destination bytes are accepted as nominal in this range, applying a bias on the source destination can assist with the detection of anomalies that outside of the normally accepted range even when there exist smaller clusters of nominal data near the anomalous data. These operations may indicate far less used supported network operations which requires a large difference in the source destination bytes. One example use case of this can be uploading files.

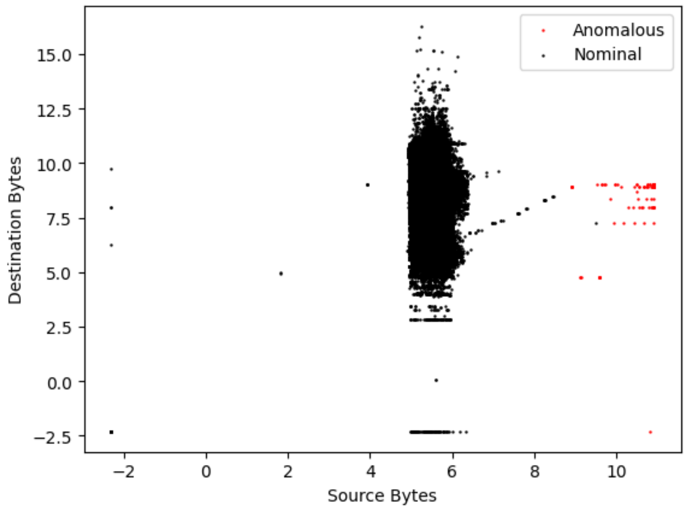

Fig. 8. Feature Bias method with $w = (.67,\ .33)$ and anomaly threshold = 0.66

VIII. Conclusion

In this paper, we introduced properties for generalizing modifications of Isolation Forest. We justified why each of these properties are required for creating Isolation Forest variations that produce meaningful anomaly scores and by extension, effective anomaly detection models. We applied these properties to create generalized versions of Isolation Forest. The first generalization on Isolation Forest we introduced was the middle method. We found that this method could be used to create a model that was useful for detecting data that was anomalous in multiple features. We also introduced the feature bias method. By weighting which features are sampled, we demonstrated how to control the importance of features in the resulting model. This method is very powerful for encoding information about the underlying anomalies in the Isolation Forest model. Through leveraging information about the nature of anomalies in a dataset, we can create improved Isolation Forest models. In section VI, we proved that our introduced methods conform to the properties we described. Finally, we applied the feature bias method to the HTTP dataset. We compared the performance of the feature bias method to Scikit Learn's implementation of Isolation Forest on using the HTTP dataset. This comparison showed that our algorithm performed better than Scikit Learn's implementation when compared to the ground truth for the dataset. Overall, our findings suggest that by leveraging domain-specific knowledge about the nature of anomalies, the Isolation Forest algorithm can be more effectively customized for specific application domains. Our work is limited by our model validation methods. Most outlier datasets do not have a ground truth. Therefore, when we are developing new methodologies, we rely on visualization to validate our solutions. This is inherently limited to low dimensions, as visualizing higher dimensional solutions is very difficult. Future work could be done to expand our validation techniques.

Developing a numeric way to score models would allow for more automated hyperparameter search, which would further increase model performance.

References

[1] K. Mehrotra, C. Mohan and H. Huang. “Anomaly Detection Principles and Algorithms.” 1st ed. 2017., Springer International Publishing, 2017, https://doi.org/10.1007/978-3-319-67526-8.

[2] F. Liu, K. Ting and Z. Zhou. “Isolation Forest.” 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 413–22, https://doi.org/10.1109/ICDM.2008.17.

[3] F. Liu, K. Ting and Z. Zhou. “Isolation-Based Anomaly Detection.” ACM Transactions on Knowledge Discovery from Data, vol. 6, no. 1, 2012, pp. 1–39, https://doi.org/10.1145/2133360.2133363.

[4] Sinaĭ, I͡Akov Grigorʹevich. Probability Theory : an Introductory Course. Springer-Verlag, 1992.

[5] Stewart, J. Calculus: Early Transcentals, 8th Ed.

[6] S. Hariri, M. Kind, R. Brunner. “Extended Isolation Forest.” IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, 2021, pp. 1479–89, https://doi.org/10.1109/TKDE.2019.2947676.

[7] Tokovarov, Mikhail, and Paweł Karczmarek. “A Probabilistic Generalization of Isolation Forest.” Information Sciences, vol. 584, 2022, pp. 433–49, https://doi.org/10.1016/j.ins.2021.10.075.

[8] P. Karczmarek, A. Kiersztyn, W. Pedrycz and E. Al. “K-Means-Based Isolation Forest.” Knowledge-Based Systems, vol. 195, 2020, p. 105659–, https://doi.org/10.1016/j.knosys.2020.105659.

[9] K. Paweł, A. Kiersztyn, W. Pedrycz and D. Czerwiński. “Fuzzy C-Means-Based Isolation Forest.” Applied Soft Computing, vol. 106, 2021, p. 107354–, https://doi.org/10.1016/j.asoc.2021.107354.

[10] L. Galka, P. Karczmarek and M. Tokovarov, “Isolation Forest Based on Minimal Spanning Tree, “IEEE access, pp. Vol.10, p.74175-74186, 2022.

[11] Laskar, Md Tahmid Rahman, et al. “Extending Isolation Forest for Anomaly Detection in Big Data via K-Means.” ACM Transactions on Cyber-Physical Systems, vol. 5, no. 4, 2021, pp. 1–26, https://doi.org/10.1145/3460976.

[12] H. Ma, B. Ghojogh, M. N. Samad, D. Zheng and M. Crowley, "Isolation Mondrian Forest for Batch and Online Anomaly Detection," 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 3051-3058, doi: https://doi.org/10.1109/SMC42975.2020.9283073.

[13] J. Lesouple, C. Baudoin, M. Spigai and J. Tourneret, “Generalized isolation forest for anomaly detection.” Pattern Recognition Letters, Volume 149, 2021, Pages 109-119, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2021.05.022.

[14] D. Zambon, L. Livi and C. Alippi, "Graph iForest: Isolation of anomalous and outlier graphs," 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1-8, doi: https://doi.org/10.1109/IJCNN55064.2022.9892295.

[15] D. Dua and C. Graff, "Machine Learning Repository [http://archive.ics.uci.edu/ml]," University of California, School of Information and Computer Science, Irvine, CA, 2019.