Isolation Mondrian Forest for Batch and Online Anomaly Detection

Haoran Ma[1,2,*], Benyamin Ghojogh[1,*], Maria N. Samad[1,*], Dongyu Zheng[1,3], Mark Crowley[1]

[1]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
[2]SoundHound, Toronto, ON, Canada
[3]Facebook, Seattle, Washington, USA
Emails: h65ma@uwaterloo.ca, bghojogh@uwaterloo.ca, mnsamad@uwaterloo.ca,
dongyu.zheng@edu.uwaterloo.ca, mcrowley@uwaterloo.ca

***Abstract*—We propose a new method, named isolation Mondrian forest (iMondrian forest), for batch and online anomaly detection. The proposed method is a novel hybrid of isolation forest and Mondrian forest which are existing methods for batch anomaly detection and online random forest, respectively. iMondrian forest takes the idea of isolation, using the depth of a node in a tree, and implements it in the Mondrian forest structure. The result is a new data structure which can accept streaming data in an online manner while being used for anomaly detection. Our experiments show that iMondrian forest mostly performs better than isolation forest in batch settings and has better or comparable performance against other batch and online anomaly detection methods.**

***Index Terms*—Anomaly detection, Mondrian forest, isolation forest, random forest, iMondrian forest.**

I. Introduction

Anomaly detection [1] refers to the task of detecting the outliers in a dataset where the anomalies are different from the regular pattern of data. Anomaly detection can be performed on either offline or online data where the data are processed as a batch or stream, respectively. Numerous methods have been proposed for batch and online anomaly detection for use in a multitude of applications such as fraud detection, disease diagnosis, and intrusion detection.

One batch anomaly detection method is Local Outlier Factor (LOF) [2] which defines a measure for local density of every data point according to its neighbors and then compares the local density of every point with its neighbors to find the anomalies. One-class SVM [3] is another method which estimates a function to be 1 and -1 in the regions with high and low density of data, respectively. It can also use kernels to map the data to a higher dimensional feature space for a possible better performance. Isolation forest (iForest) [4] is an isolation-based method [5] which isolates anomalies rather than separating normal points. Its main idea is that anomalies are separated shallower in the tree by a random forest so the depth of a node can determine the anomaly score of a point.

One method for online anomaly detection is incremental LOF [6], which updates the local density and other characteristics of the LOF algorithm for the new data points in the stream. It also updates the density for the existing points which are affected by the new data in their k-nearest neighbors. There is another method which uses kernel density estimation for online anomaly detection which assigns the anomaly label to a point if it deviates significantly from the estimated density [7]; note that, this method does not calculate scores for the points. Oversampling Principal Component Analysis (osPCA) is an online anomaly detection method which oversamples a point and calculates the principal direction both with and without the oversampled point. If the principal direction deviates significantly, the point is considered to be anomalous. There exist two versions of osPCA: osPCA1 with power method [8] and osPCA2 with least squares approximation [9].

Decision forests [10] are ensembles of decision trees which partition the input space for different tasks such as classification and regression. Random forest [11] is a forest of binary trees which randomly sample from data points and features for the different trees. As the trees in random forest are not very correlated, the variance of estimation is reduced because of bootstrap aggregating or bagging [12]. Extremely randomized trees [13] select both the split dimension and split value randomly. Some forests, such as Hoeffding trees [14], are proposed for online processing of data [15], [16]. The Hoeffding tree learns a regression tree model but stores no data points along the way. New points help build confidence in the split at each node, and once the confidence is high enough, the node is split. Later arriving points are used to start learning the new leaves which could in turn become new internal nodes. Mondrian forest (MForest) is another online forest method for classification [17] and regression [18]. It is based on the Mondrian processes [19] which are a family of distributions over tree structures. The trees in MForest can grow incrementally without complex tree rotation and correction which previous streaming tree methods required. The new nodes in MForest can also be added as internal nodes. While Hoeffding trees perform online learning and scale well for regression and classification, their particular incremental approach to tree building and lack of restructuring makes them less suitable than MForest to utilize the isolation concept [5] for anomaly detection. In this paper we propose *isolation Mondrian forests* (*iMondrian forest* or *iMForest*), a

*The first three authors contributed equally to this work.

novel hybrid of iForest and MForest providing the best of both worlds: an anomaly detector which can perform unsupervised learning of anomalous and normal points for both batch and online processing.

Assume we have a batch of data denoted by $\mathcal{X} := \{\boldsymbol{x}_i\}_{i=1}^n$ where n is the sample size and d is the dimensionality of data, i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$. We may also have m new data points denoted by $\mathcal{X}^{(n)} := \{\boldsymbol{x}_i^{(n)}\}_{i=1}^m$. Our goal is to detect the anomalies in $\mathcal{X}$ and $\mathcal{X}^{(n)}$ using both batch and online processing of data.

II. Background

In this section, we review isolation forest [4] and Mondrian forest [17] in more detail.

A. Isolation Forest

An *isolation forest* (*iForest*) [4] is an ensemble of isolation trees. An *isolation tree* is an extremely randomized tree [13] where the tree is a proper binary tree and its splitting dimension q and splitting value p are randomly selected at every node. The tree grows until every leaf includes exactly one data point, i.e., $|\mathcal{X}| = 1$ in the leaf node. Let $h(\boldsymbol{x})$ denote the path length for a data point $\boldsymbol{x}$ in the tree where the path length is defined as the number of edges $\boldsymbol{x}$ traverses from the root to the leaf it belongs to. The average height of an isolation tree is $\log(n)$. As the structure of the isolation tree is equivalent to the binary search tree, the estimation of average path length in isolation trees is [20]:

$$c(n) := 2\,h(n-1) - \big(2\,(n-1)/n\big), \tag{1}$$

where $h(i)$ is the i-th harmonic number, defined as:

$$h(i) := \ln(i) + 0.5772156649, \tag{2}$$

where the added constant is the Euler's constant. The anomaly score of a point $\boldsymbol{x}$ is:

$$s(\boldsymbol{x}) := 2^{-\mathbb{E}(l(\boldsymbol{x}))/c(n)}, \tag{3}$$

where $\mathbb{E}(l(\boldsymbol{x}))$ is the expected path length for the data point $\boldsymbol{x}$ among the trees of the forest:

$$\mathbb{E}(l(\boldsymbol{x})) := \frac{1}{|\mathcal{F}|}\sum_{t=1}^{|\mathcal{F}|} l_t(\boldsymbol{x}), \tag{4}$$

where $l_t(\boldsymbol{x})$ is the path length of $\boldsymbol{x}$ in the t-th tree and $|\mathcal{F}|$ is the population of trees in the forest. The intuition of anomaly score in iForests is that the anomalies tend to be isolated sooner, i.e., shallower in the tree, while the normal points require more splits to become isolated, i.e., deeper the tree. The anomaly score is in the range $s(\boldsymbol{x}) \in [0, 1]$ where $s(\boldsymbol{x}) = 0$ and $s(\boldsymbol{x}) = 1$ corresponded to normal and anomaly points, respectively. The $s = 0.5$ may be a proper threshold for anomaly detection in iForests. Note that the authors of the original iForest use a subset of the data with subsampling size specified by $\psi = 256$ [4].

Procedure: BatchTraining($\mathcal{X}$, $|\mathcal{F}|$)
Input: $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$, $|\mathcal{F}|$: number of trees
for *tree* t *from* 1 *to* $|\mathcal{F}|$ **do**
  $\mathcal{F} \leftarrow \mathcal{F} \cup$ iMondrianTree(root, $\mathcal{X}, 0$)
Return Forest $\mathcal{F}$

Algorithm 1: Batch training in iMondrian forest.

B. Mondrian Forest

A *Mondrian forest* [17] is an ensemble of Mondrian trees which are based on the Mondrian process [19]. *Mondrian processes* are families of random hierarchical binary partitions and probability distributions over tree data structures. While Mondrian processes are infinite structures, Mondrian trees are restrictions of Mondrian processes on a finite set of points. Every node r in the Mondrian tree has a *split time* τ_r which increases with the depth of the node. The split time is zero at the root and infinite at the leaves of the tree.

Let $\hat{\mathcal{B}}_r := (\hat{\ell}_{r1}, \hat{u}_{r1}] \times \cdots \times (\hat{\ell}_{rd}, \hat{u}_{rd}]$ for the r-th node, where $\hat{\ell}_{rj}$ and $\hat{u}_{rj}$ are the lower and upper bounds of hyper-rectangular block $\hat{\mathcal{B}}_r$ along dimension j. The Mondrian tree considers the smallest block containing the data points in a node; therefore, it defines $\mathcal{B}_r := (\ell_{r1}, u_{r1}] \times \cdots \times (\ell_{rd}, u_{rd}]$ where ℓ_{rj} and u_{rj} are the lower and upper bounds of the smallest hyper-rectangular block $\mathcal{B}_r$ along dimension j. For a node indexed by r, let $\boldsymbol{\ell}_{\mathcal{X}_b} = [\ell_{r1}, \ldots, \ell_{rd}]^\top$ and $\boldsymbol{u}_{\mathcal{X}_b} = [u_{r1}, \ldots, u_{rd}]^\top$; thus, $\boldsymbol{\ell}_{\mathcal{X}_b} := \min(\{\boldsymbol{x}_i^{(b)} \mid \forall i\})$ and $\boldsymbol{u}_{\mathcal{X}_b} := \max(\{\boldsymbol{x}_i^{(b)} \mid \forall i\})$ where $\mathcal{X}_b = \{\boldsymbol{x}_i^{(b)}\} = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{B}_r\}$. For the r-th node, the split time of a node is determined as $\tau_{\text{parent}(j)} + e$ where e is a random variable from an exponential distribution with a rate which is a function of $\boldsymbol{\ell}_{\mathcal{X}_b}$ and $\boldsymbol{u}_{\mathcal{X}_b}$. Depending on whether the split time of the node is smaller or greater than the split time of its parent, it is put before or after the parent node in the tree.

Mondrian trees can be updated with new data making them suitable for online streaming domains. When a new data point arrives, it is checked whether it belongs to an existing block or not. If not, the lower and upper errors (deviations) from the block are calculated. Again, a random variable is sampled from an exponential distribution with a rate which is a function of the lower and upper errors. As before, the split time of the new node is calculated and depending on it, its location in the tree is determined. In this way, new nodes can be added in the middle of a tree and not just grown at the end of tree like in regular online random forests. Note that the Mondrian forest is designed for classification [17] and regression [18] so its authors propose an analysis for smooth posterior updates in the blocks. However, the Mondrian forest can be used for unsupervised purposes where the posterior analysis and the pausing process can be bypassed. This is useful for this work because anomaly detection is usually considered as an unsupervised task.

Procedure: iMondrianTree(r, $\mathcal{X}, \tau_{\text{parent}}$)
Input: r: node pointer, $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$, τ_{parent}: split time of the parent node
$\mathcal{X}_b = \{\boldsymbol{x}_i^{(b)}\} = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{B}_r\}$
$\boldsymbol{\ell}_{\mathcal{X}_b} \leftarrow \min(\{\boldsymbol{x}_i^{(b)} \mid \forall i\})$
$\boldsymbol{u}_{\mathcal{X}_b} \leftarrow \max(\{\boldsymbol{x}_i^{(b)} \mid \forall i\})$
if $|\mathcal{X}_b| > 1$ **then**
  $e \sim \text{Exp}\big(\lambda = \sum_{j=1}^d (\boldsymbol{u}_{\mathcal{X}_b}(j) - \boldsymbol{\ell}_{\mathcal{X}_b}(j))\big)$
  $\tau \leftarrow \tau_{\text{parent}} + e$
  $q \leftarrow$ sample from $\{1, \ldots, d\}$ with distribution $\propto (\boldsymbol{u}_{\mathcal{X}_b}(j) - \boldsymbol{\ell}_{\mathcal{X}_b}(j))$ for the j-th dimension
  $p \sim U(\boldsymbol{\ell}_{\mathcal{X}_b}(q), \boldsymbol{u}_{\mathcal{X}_b}(q))$
  $\mathcal{X}_{\text{left}} \leftarrow \{\boldsymbol{x} \in \mathcal{X}_b \mid \boldsymbol{x}(q) < p\}$
  $\mathcal{X}_{\text{right}} \leftarrow \{\boldsymbol{x} \in \mathcal{X}_b \mid \boldsymbol{x}(q) \geq p\}$
  Left $\leftarrow$ iMondrianTree(leftChild(r), $\mathcal{X}_{\text{left}}, \tau$)
  Right $\leftarrow$ iMondrianTree(rightChild(r), $\mathcal{X}_{\text{right}}, \tau$)
  Return internalNode{leftChild: Left, rightChild: Right, splitDim: q, splitVal: p, time: τ, dim_{min}: $\boldsymbol{\ell}_{\mathcal{X}_b}$, dim_{max}: $\boldsymbol{u}_{\mathcal{X}_b}$, population: $|\mathcal{X}_b|$}
else
  Return leafNode{time: ∞, dim_{min}: $\boldsymbol{\ell}_{\mathcal{X}_b}$, dim_{max}: $\boldsymbol{u}_{\mathcal{X}_b}$, population: 1}

Algorithm 2: Constructing iMondrian tree.

III. iMondrian Forest

The iMondrian forest can be used for both batch and online anomaly detection. In the following, we cover both cases.

A. Batch Processing

1) Training: The iMondrian forest is an ensemble of iMondrian trees. Algorithm 1 shows this ensemble where $\mathcal{X} := \{\boldsymbol{x}_i\}_{i=1}^n$ is the batch of data and $|\mathcal{F}|$ is the number of trees in the forest. Inspired by [4], the data in a batch can be subsampled with subsampling size $\psi = 256$ for growing the tree. If subsampling is used, $\mathcal{X}$ denotes the sample of data and $n = \psi$. The iMondrian tree is grown recursively as detailed in Algorithm 2. As with Mondrian trees, bounds of hyper-rectangular blocks are defined $\mathcal{B}_r := (\ell_{r1}, u_{r1}] \times \cdots \times (\ell_{rd}, u_{rd}]$ along each of d dimensions for the r-th node. Let $\mathcal{X}_b := \{\boldsymbol{x}_i^{(b)}\} := \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{B}_r\}$ be the subset of data which exist in the smallest block enclosing the node. For a node, the lower and upper bounds of $\mathcal{B}_r$ along the features are denoted by $\boldsymbol{\ell}_{\mathcal{X}_b}$ and $\boldsymbol{u}_{\mathcal{X}_b}$, respectively.

In order to split a block, we sample a random variable e from an exponential distribution with the rate $\lambda = \sum_{j=1}^d (\boldsymbol{u}_{\mathcal{X}_b}(j) - \boldsymbol{\ell}_{\mathcal{X}_b}(j))$ which is the linear dimension of $\mathcal{B}_r$. We set the split time of a node to the split time of its parent plus e. We sample the dimension of the split, q, from a discrete distribution proportional to $(\boldsymbol{u}_{\mathcal{X}_b}(j) - \boldsymbol{\ell}_{\mathcal{X}_b}(j))$. We sample the value of the split, p, from a continuous uniform distribution $U(\boldsymbol{\ell}_{\mathcal{X}_b}(q), \boldsymbol{u}_{\mathcal{X}_b}(q))$. The tree is grown until every node contains a single data point, i.e., $|\mathcal{X}| = 1$.

2) Evaluation: After growing the iMondrian trees in the forest, we calculate the path length of every tree for a data point $\boldsymbol{x}$ as in Algorithm 3. The path length for the t-th tree, $l_t(\boldsymbol{x})$, is the number of edges traversed by the point from the root to the node containing point $\boldsymbol{x}$. We calculate the expected path length in the iMondrian forest using Eq. (4) and the anomaly score for point $\boldsymbol{x}$ using Eq. (3). For determining whether a point in the dataset is normal or an anomaly, we can either use the threshold $s = 0.5$ as in [4] or K-means clustering. In the threshold approach, the point is determined as anomaly if $s(\boldsymbol{x}) > 0.5$. In the K-means approach, we assign the scores of training data into two clusters and take the points in the cluster with greater mean as the anomaly points. The theoretical reason for threshold 0.5 is that the expected path length for the data point (Eq. (4)) is the estimation of the

Procedure: PathLength($\boldsymbol{x}$, t, l)
Input: $\boldsymbol{x}$: data point, t: iMondrian tree, l: current path length (initialized to 0)
$q \leftarrow t$.splitDim
$p \leftarrow t$.splitVal
if $\boldsymbol{x}(q) < p$ **then**
  Return PathLength($\boldsymbol{x}$, t.leftChild, $l + 1$)
else
  Return PathLength($\boldsymbol{x}$, t.rightChild, $l + 1$)

Algorithm 3: Calculation of path length.

Procedure: ExtendIMondrianForest($\mathcal{X}^{(n)}$, $\mathcal{F}$)
Input: $\mathcal{X}^{(n)} = \{\boldsymbol{x}_i^{(n)}\}_{i=1}^m$: new data, $\mathcal{F}$: Forest
for $\boldsymbol{x}_i^{(n)} \in \mathcal{X}^{(n)}$ **do**
  for *tree* $t \in \mathcal{F}$ **do**
    ExtendIMondrianTree(t.root, $\boldsymbol{x}_i^{(n)}, 0$)

Algorithm 4: Extension of iMondrian forest.

Procedure: ExtendIMondrianTree($r, \boldsymbol{x}^{(n)}, \tau_{\text{parent}}$)
Input: r: node pointer, new data point: $\boldsymbol{x}^{(n)}$, τ_{parent}: split time of the parent node
$\boldsymbol{e}_\ell \leftarrow \max(r.\text{dim}_{\text{min}} - \boldsymbol{x}^{(n)}, \boldsymbol{0})$
$\boldsymbol{e}_u \leftarrow \max(\boldsymbol{x}^{(n)} - r.\text{dim}_{\text{max}}, \boldsymbol{0})$
$e \sim \text{Exp}\big(\lambda = \sum_{j=1}^{d} (\boldsymbol{e}_\ell(j) + \boldsymbol{e}_u(j))\big)$
if $\tau_{parent} + e < r.\tau$ **then**
  $q \leftarrow$ sample from $\{1, \ldots, d\}$ with distribution $\propto (\boldsymbol{e}_\ell(j) + \boldsymbol{e}_u(j))$ for the j-th dimension
  if $\boldsymbol{x}^{(n)}(q) > r.dim_{max}(q)$ **then**
    $p \sim U\big(r.\text{dim}_{\text{max}}(q), \boldsymbol{x}^{(n)}(q)\big)$
  else if $\boldsymbol{x}^{(n)}(q) < r.dim_{min}(q)$ **then**
    $p \sim U\big(\boldsymbol{x}^{(n)}(q), r.\text{dim}_{\text{min}}(q)\big)$
  newNode $\leftarrow$ internalNode{splitDim: q, splitVal: p, time: $\tau_{\text{parent}} + e$, dim_{min}: $\min(r.\text{dim}_{\text{min}}, \boldsymbol{x}^{(n)})$, dim_{max}: $\max(r.\text{dim}_{\text{max}}, \boldsymbol{x}^{(n)})$, population: r.population + 1}
  newNode.parent $\leftarrow r$.parent
  if $\boldsymbol{x}^{(n)}(q) > p$ **then**
    newNode.leftChild $\leftarrow r$
    newNode.rightChild $\leftarrow$ iMondrianTree (rightSibling(r), $\boldsymbol{x}^{(n)}$, newNode.time)
  else
    newNode.leftChild $\leftarrow$ iMondrianTree (leftSibling(r), $\boldsymbol{x}^{(n)}$, newNode.time)
    newNode.rightChild $\leftarrow r$
else
  $r.\text{dim}_{\text{min}} \leftarrow \min(r.\text{dim}_{\text{min}}, \boldsymbol{x}^{(n)})$
  $r.\text{dim}_{\text{max}} \leftarrow \max(r.\text{dim}_{\text{max}}, \boldsymbol{x}^{(n)})$
  if $\boldsymbol{x}(r.splitDim) \leq r.splitVal$ **then**
    ExtendIMondrianTree(r.leftChild, $\boldsymbol{x}^{(n)}, r.\tau$)
  else
    ExtendIMondrianTree(r.rightChild, $\boldsymbol{x}^{(n)}$, $r.\tau$)

Algorithm 5: Extension of iMondrian tree.

average path length (Eq. (1)) when $s = 0.5$ (see [4, p. 415], same holds for us). The empirical reason is that the results of $s = 0.5$ and K-means are almost the same (see Fig 1).

In batch processing, for an out-of-sample data point, or novelty detection [21], we feed the data point to the trees of iMondrian forest and calculate the score using Eq. (3). Then, we can use the threshold $s = 0.5$ again or assign the point to the cluster whose mean is closer to the score of the point. Our experiments showed that both the threshold and clustering approaches have almost equally good performance for batch processing.

B. Online Processing

1) Training: A major advantage of iMondrian forests is their ability to be updated online for new data. Let $\mathcal{X}^{(n)} := \{\boldsymbol{x}_i^{(n)}\}_{i=1}^{m}$ denote the m new data points. We process data points one-by-one to extend each tree in the forest (see Algorithm 4). Algorithm 5 describes how we extend each iMondrian tree for $\boldsymbol{x}^n$. The tree is extended recursively starting from the root. The lower and upper errors of deviation of a point from the smallest block contained by the node r are calculated as $\mathbb{R}^d \ni \boldsymbol{e}_\ell := \max(r.\text{dim}_{\text{min}} - \boldsymbol{x}^{(n)}, \boldsymbol{0})$ and $\mathbb{R}^d \ni \boldsymbol{e}_u := \max(\boldsymbol{x}^{(n)} - r.\text{dim}_{\text{max}}, \boldsymbol{0})$, respectively, where dim_{min} and dim_{max} are the upper and lower bounds of the block along different dimensions. We sample a random variable e from an exponential distribution with the rate $\lambda = \sum_{j=1}^{d} (\boldsymbol{e}_\ell(j) + \boldsymbol{e}_u(j))$.

In the case where the split time of the node r is greater than the split time of its parent plus e, a new node is created above the node r. Note that we started from the root and are moving downwards so the new node is added before the current node for which a condition holds. In this case, we randomly pick a split dimension q from the distribution proportional to $(\boldsymbol{e}_\ell(j) + \boldsymbol{e}_u(j))$. If the value on dimension q of the data point is *greater* than the upper bound of the current block, then the split value p is sampled from the uniform distribution $U\big(r.\text{dim}_{\text{max}}(q), \boldsymbol{x}^{(n)}(q)\big)$. If the value is *lower*, then p is sampled from $U\big(\boldsymbol{x}^{(n)}(q), r.\text{dim}_{\text{min}}(q)\big)$. Depending on the split value and the feature of data point, we create an iMondrian tree as the left or right sibling of the node r.

In the case where the split time of the node r is less than the split time of its parent plus e, we simply descend down the tree and call the extending function recursively for the left or right of the node r depending on the split dimension and split values of the children.

2) Evaluation: After the extension of the trees of iMondrian forest, we can process data points through the forest to calculate their anomaly scores using Eq. (3). This can be done for all the new points and any other out-of-sample points. Whenever the trees have been updated we should also ideally process previous batches of data through the forest again to recalculate their anomaly scores. This is expected since more data will lead to an improved model and a better structure for detection of false negative or positive points. However, for performance reasons, in practice this recalculation of scores could be done for just a window of the latest points. For online processing, our experiments showed that the threshold $s = 0.5$ is not necessarily the best threshold and K-means clustering works more better. Hence, we use K-means to cluster all the into two clusters and set the cluster with greater mean as anomalous. The out-of-sample points are assigned to the cluster whose mean is closer to their score.

IV. Complexity Analysis

Proposition 1. *The training and evaluation (score calculation) phases of batch processing in iMondrian forest each take $\mathcal{O}(|\mathcal{F}|\, d\, n \log n)$ and $\mathcal{O}(|\mathcal{F}|\, n \log n)$ time, respectively, assuming that the trees are balanced.*

Proof. Assuming that the tree is balanced, in training, the i-th point traverses $\log n$ edges to reach its leaf. For n points, we have $\mathcal{O}(\sum_{i=1}^{n} \log n) = \mathcal{O}(\log(n^n)) = \mathcal{O}(n \log n)$ [22]. Moreover, computation of $\boldsymbol{\ell}_{\mathcal{X}_b}$ and $\boldsymbol{u}_{\mathcal{X}_b}$ take $\mathcal{O}(d\, n)$ time at every depth of tree, resulting in $\mathcal{O}(d\, n \log n)$ in a tree. The complexity of a forest with $|\mathcal{F}|$ trees is, therefore, $\mathcal{O}(|\mathcal{F}|\, d\, n \log n)$. In evaluation, every point traverses at most $\mathcal{O}(\log n)$ edges. For the n points and the whole forest, it becomes $\mathcal{O}(|\mathcal{F}|\, n \log n)$. Note that if we use subsampling with size ψ, the complexity of training and evaluation phases become $\mathcal{O}(|\mathcal{F}|\, d\, \psi \log \psi)$ and $\mathcal{O}(|\mathcal{F}|\, n \log \psi)$, respectively. Also, note that since the trees are independent of one another, trees can be computed in parallel. □

Proposition 2. *After construction of forest by the initial batch, the training and evaluation (score calculation) phases of online processing in iMondrian forest each take $\mathcal{O}(|\mathcal{F}|\, d\, (n+m) \log(n+m))$ and $\mathcal{O}(|\mathcal{F}|\, (n+m) \log(n+m))$ time, respectively, assuming that the trees are balanced.*

Proof. In training, the i-th new point traverses at most $\log(n+i)$ edges in the worst case because every new point adds either an internal or leaf node to the tree. The time for the m new points is $\mathcal{O}(\log(n+1)+\cdots+\log(n+m)) = \mathcal{O}(\log((n+m)!) - \log(n!)) \approx \mathcal{O}((n+m)\log(n+m)) - \mathcal{O}(n \log n) = \mathcal{O}((n+m)\log(n+m))$ where approximation is because of Stirling's approximation for the factorial function. Also, calculation of $\boldsymbol{e}_\ell$ and $\boldsymbol{e}_u$ take $\mathcal{O}(d)$ time for each new point at every node. The time for the whole forest is $\mathcal{O}(|\mathcal{F}|\, d\, (n+m) \log(n+m))$; although, the trees can be processed in parallel. In evaluation, the tree includes $(n+m)$ leaves so every point traverses at most $\mathcal{O}(\log(n+m))$ edges. Having all the trees and the $n+m$ points, including the re-evaluation of previous batch, takes $\mathcal{O}(|\mathcal{F}|\, (n+m) \log(n+m))$ time. □

V. Experiments

A. Synthetic Data

We created four two-dimensional synthetic datasets (a)-(d) as can be seen in Fig. 1. The datasets include a variety of edge-case distributions of random anomalous points. Dataset (a) has 255 and the rest of datasets have 100 inliers while the number of outliers are 45.

1) Batch Experiments: The results of batch processing in iMondrian forest on the synthetic datasets are illustrated in Fig. 1. The anomaly scores are shown for the input space of data. As expected, the scores are higher for anomalous points of space, which are far away from the core of distribution. Fig. 1 shows the results of both K-means clustering and thresholding (with threshold $s = 0.5$) which perform almost equally well. We also show the result of iForest (with threshold $s = 0.5$) for the sake of comparison. iMondrian forest clearly performed much better than iForest due to having much fewer false negatives. It is because iMondrian trees take into account the smallest blocks containing the points within a node while iForest considers the whole block.

2) Online Experiments: We divided every dataset, using stratified sampling, into five subsets with equal amounts of outliers. We used these subsets to simulate streaming data by adding each subset to the existing data in a succession of five steps. Fig. 2 shows the results of online processing using iMondrian forests on the synthetic datasets. K-means clustering was used in all of these experiments. In set (a), we see that in the second step, some inliers are falsely recognized as anomalous; however, by receiving more data in the next steps, the structures of iMondrian trees have been modified correctly and those points are recognized correctly as inliers. For set (c), merely some core points of the larger blob are detected as normal. This is because in the initial steps, there happen to be far fewer points from the smaller blob so the algorithm has found that region to be anomalous; however, if that blob had become much denser in the further steps, they would be detected as normal. Overall, we see that for the different datasets, the performance of online iMondrian forest is acceptable.

B. Real Data

We selected eight varied datasets with different characteristics from the outlier detection datasets [23] and one very large dataset, CICIDS 2017 [24], [25]. In CICIDS data, we only used the data of Wednesday and excluded its one categorical feature. Table I reports the characteristics of these datasets. The datasets have different sample size, dimensionality, and percentage of outliers.

1) Batch Experiments: We compared iMondrian forest with iForest, LOF (with $k = 10$), and one-class SVM (with RBF kernel). The experiments were performed with 10-fold cross validation except for the wine dataset where we used two folds due to small sample size. The average Area Under ROC Curve (AUC) [26], which is a common measure in anomaly detection literature [4], [5], and time of algorithms over the

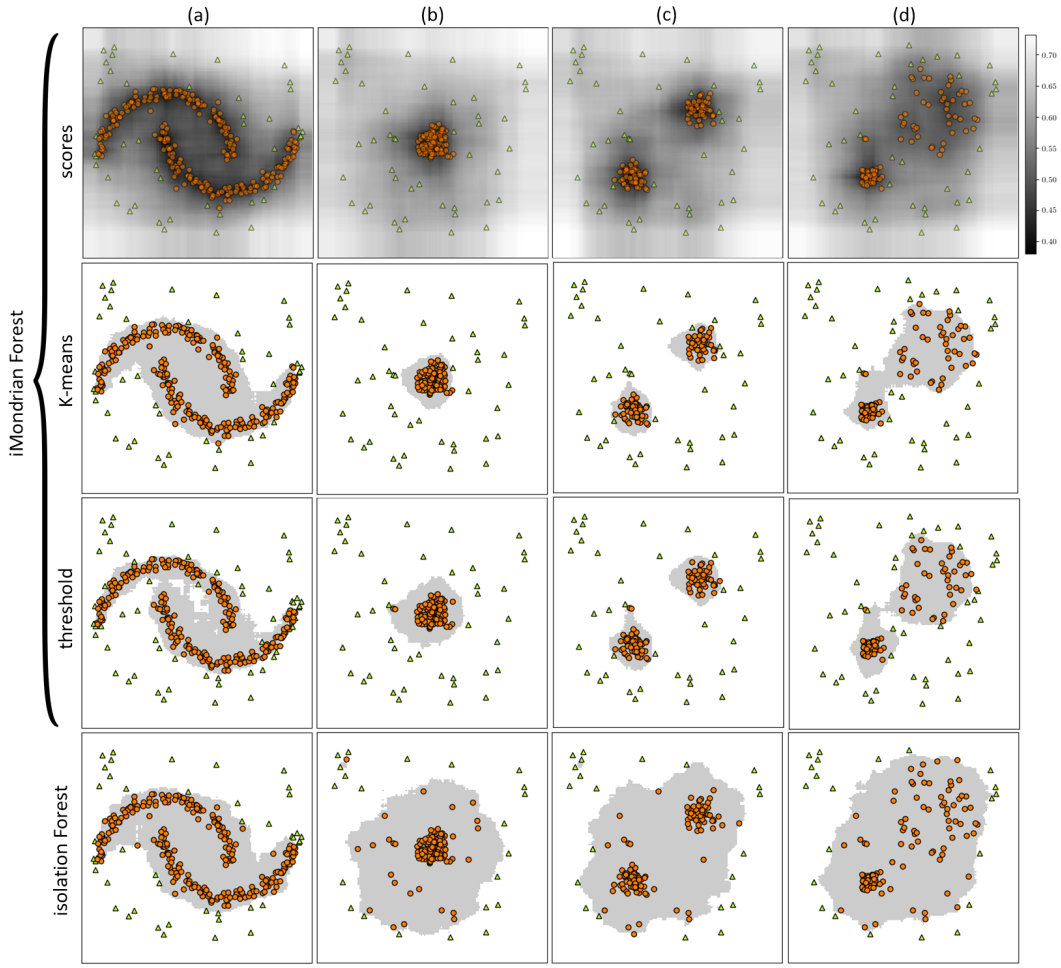

Fig. 1. Comparison of batch anomaly detection in iMondrian and iForests for the synthetic datasets (a)-(d). The orange circles and green triangles correspond to the detected normal and anomalous points, respectively, while shaded gray regions show the partition of space detected as normal.

TABLE I
CHARACTERISTICS OF UTILIZED DATASETS FOR EXPERIMENTS.

	WBC	Pima	Thyroid	Satellite	Optdigits	Ionosphere	Wine	SMTP	CICIDS
#Instances	278	768	3772	6435	5216	351	129	95156	691406
#Features	30	8	6	36	64	33	13	3	77
% anomalies	37%	35%	2.5%	32%	3%	36%	7.7%	0.03%	36%

ten folds are reported in Table II. The results are reported for both training and test subsets of data. In most datasets, we outperform iForest, LOF, and SVM. In three datasets Pima, thyroid, and and SMTP, iForest is slightly better; although, the difference is not significant. In time, iForest is mostly better than iMondrian but its accuracy is often less.

2) Online Experiments: For the online experiments, we divided datasets into five stages using stratified sampling and introduce the streaming data to the algorithms where each new point is accumulated to previous data. The AUC of a stage is for scores up to that stage. WBC was not used here because there was such a small relative portion of outliers it made the stratified sampling not possible. We compared iMondrian forest with incremental LOF (with $k = 10$), osPCA1, and osPCA2, reported in Table III. The results of CICIDS on osPCA methods are not reported as they did not perform in a reasonable time on these datasets. The AUC of iMondrian forest reported for every stage is the rate for recalculated scores of the available data. In the first stage of osPCA1 and osPCA2, we used decremental PCA approach with oversampling [9]. In different datasets, iMondrian forest has stable performance in different stages which shows its stability over the streaming data. In most cases, we outperform all the baseline methods. In terms of time, we outperform osPCA1 and osPCA2.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed iMondrian forest for batch and online anomaly detection. It is a novel hybrid of isolation

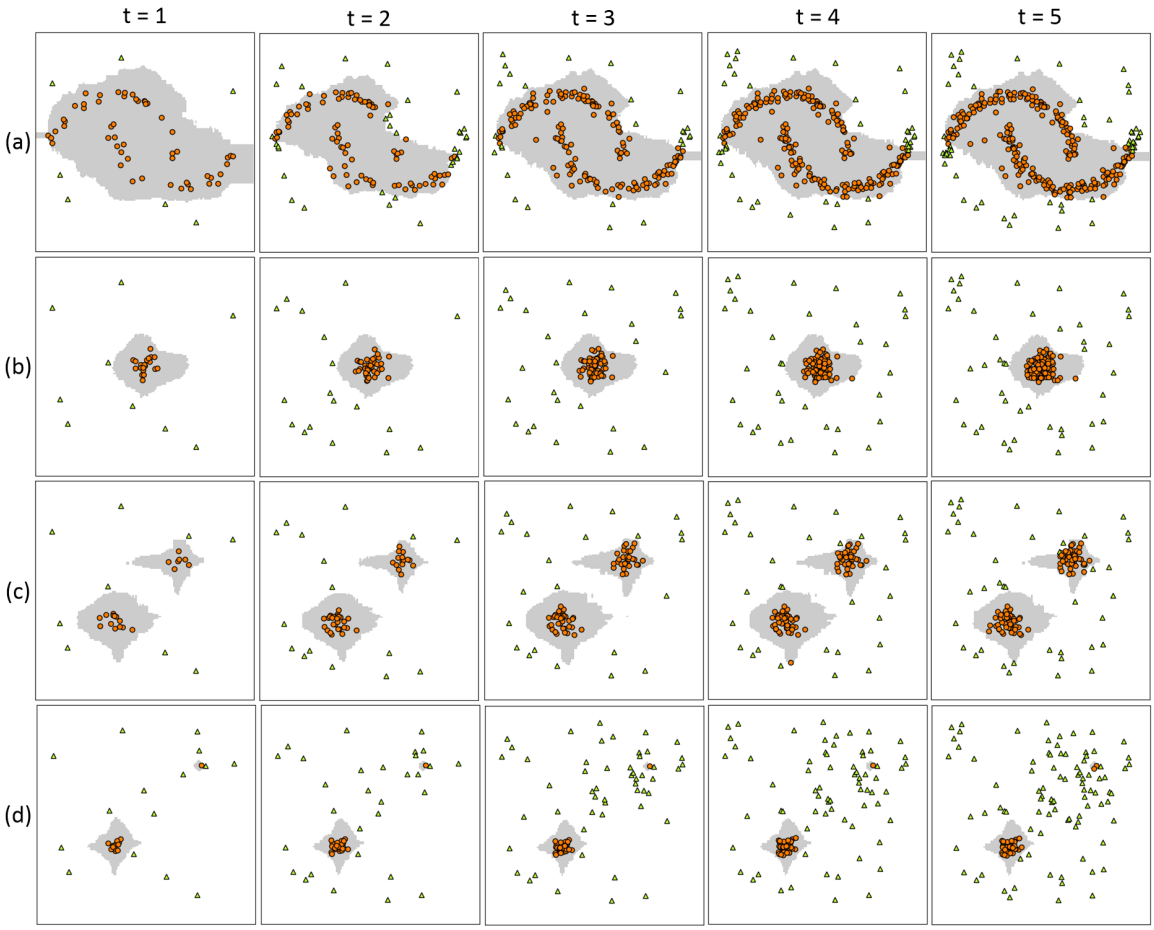

Fig. 2. Online anomaly detection in iMondrian forest for the synthetic datasets (a)-(d). The orange circles and green triangles correspond to the detected normal and anomalous points, respectively, while shaded gray regions show the partition of space detected as normal. Time steps are denoted by t in this figure.

TABLE II
COMPARISON OF BATCH ANOMALY DETECTION METHODS. RATES ARE AUC PERCENTAGE AND TIMES ARE IN SECONDS AVERAGED OVER THE FOLDS. THE UPWARD ARROWS IN AUC RATES MEAN IMFOREST OUTPERFORMS THE OTHER METHODS.

			WBC	Pima	Thyroid	Satellite	Optdigits	Ionosphere	Wine	SMTP
iMForest	**Train:**	**Time:**	2.40 ± 0.01	2.54 ± 0.04	4.96 ± 0.15	16.33 ± 0.18	5.28 ± 0.03	2.27 ± 0.02	0.48 ± 0.00	68.64 ± 1.11
		AUC:	86.35 ± 1.31	63.63 ± 1.02	95.36 ± 0.41	73.93 ± 1.19	72.90 ± 3.49	86.07 ± 0.95	99.01 ± 0.16	86.76 ± 1.47
	Test:	**Time:**	0.04 ± 0.00	0.07 ± 0.01	0.33 ± 0.02	1.58 ± 0.02	0.35 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	7.42 ± 0.11
		AUC:	86.25 ± 5.02	63.74 ± 9.39	95.37 ± 1.62	73.67 ± 2.33	73.00 ± 7.64	83.99 ± 6.32	99.71 ± 0.28	85.12 ± 14.25
iForest	**Train:**	**Time:**	0.14 ± 0.00	0.14 ± 0.00	0.25 ± 0.01	0.46 ± 0.01	0.81 ± 0.01	0.12 ± 0.00	0.08 ± 0.00	4.41 ± 0.05
		AUC:	78.75 ± 1.62 ↑	67.49 ± 1.36	97.89 ± 0.22	70.13 ± 2.13 ↑	68.38 ± 4.64 ↑	84.74 ± 0.94 ↑	79.56 ± 10.59 ↑	90.74 ± 1.37
	Test:	**Time:**	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.19 ± 0.00
		AUC:	78.81 ± 6.20 ↑	68.00 ± 5.14	97.87 ± 0.84	70.13 ± 3.46 ↑	68.36 ± 8.11 ↑	84.32 ± 6.19	76.09 ± 10.11 ↑	89.44 ± 10.02
LOF	**Train:**	**Time:**	0.01 ± 0.00	0.01 ± 0.00	0.04 ± 0.00	0.65 ± 0.00	1.84 ± 0.05	0.01 ± 0.00	0.01 ± 0.00	1.05 ± 0.06
		AUC:	61.12 ± 1.56 ↑	49.91 ± 1.41 ↑	70.26 ± 2.10 ↑	52.65 ± 0.33 ↑	60.84 ± 1.67 ↑	89.59 ± 0.81	98.70 ± 1.29 ↑	53.51 ± 7.25 ↑
	Test:	**Time:**	0.01 ± 0.00	0.01 ± 0.00	0.05 ± 0.00	0.77 ± 0.01	2.13 ± 0.08	0.01 ± 0.00	0.01 ± 0.00	1.14 ± 0.04
		AUC:	61.94 ± 7.56 ↑	51.36 ± 7.50 ↑	66.17 ± 13.06 ↑	53.26 ± 2.32 ↑	61.12 ± 11.65 ↑	89.87 ± 7.23	92.58 ± 5.69 ↑	56.23 ± 25.90 ↑
SVM	**Train:**	**Time:**	0.03 ± 0.00	0.04 ± 0.00	0.27 ± 0.00	3.78 ± 0.00	3.10 ± 0.01	0.01 ± 0.00	0.01 ± 0.00	240.93 ± 3.81
		AUC:	49.40 ± 3.16 ↑	51.93 ± 0.02 ↑	84.36 ± 0.53 ↑	48.54 ± 0.57 ↑	50.52 ± 3.81 ↑	76.23 ± 0.86 ↑	68.59 ± 4.25 ↑	84.14 ± 1.75 ↑
	Test:	**Time:**	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.18 ± 0.00	0.15 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	4.89 ± 0.09
		AUC:	94.21 ± 1.35	60.01 ± 8.37 ↑	84.49 ± 4.12 ↑	64.04 ± 1.65 ↑	37.49 ± 7.41 ↑	76.61 ± 7.90 ↑	91.13 ± 3.83 ↑	83.06 ± 17.75 ↑

and Mondrian forests which are existing methods for batch anomaly detection and online random forest, respectively. The proposed method makes use of the best of two worlds of isolation-based anomaly detection and online ensemble learning. As a future direction, we seek to investigate whether it is worthy to develop the idea of isolation-based anomaly detection in other online tree structures such as binary space partitioning forest [27] which is based on the binary partitioning process [28]. Another future work is to investigate other clustering methods instead of K-means for clustering scores.

TABLE III

COMPARISON OF ONLINE ANOMALY DETECTION METHODS. RATES ARE AUC PERCENTAGE AND TIMES ARE IN SECONDS. UPWARD ARROWS MEAN BETTER PERFORMANCE OF IMFOREST.

	Stages	Pima	Thyroid	Satellite	Optdigits	Ionosphere	Wine	SMTP	CICIDS
iMForest	**Time:**	1.25	6.59	13.24	9.85	0.56	0.21	185.33	2.6E3
	AUC:	70.19	95.41	71.94	67.50	86.62	95.65	95.48	71.02
	Time:	1.48	8.40	15.26	11.88	0.65	0.22	364.03	1.0E4
	AUC:	68.07	94.27	73.73	68.00	85.07	98.91	95.01	70.95
	Time:	1.59	9.20	16.65	12.93	0.69	0.23	319.45	5.9E3
	AUC:	65.45	94.65	74.15	66.70	84.27	98.79	96.58	70.76
	Time:	1.72	10.32	18.87	14.45	0.73	0.24	349.33	7.3E3
	AUC:	64.51	94.58	74.00	66.40	83.10	98.64	93.84	70.80
	Time:	1.88	11.29	20.67	15.29	0.79	0.29	393.93	8.6E3
	AUC:	65.50	94.64	73.40	67.10	82.80	97.60	92.87	70.83
Incremental LOF	**Time:**	0.001	0.006	0.04	0.06	0.001	0.0009	0.71	4.1E2
	AUC:	58.81 ↑	85.61 ↑	54.23 ↑	56.87 ↑	93.06	95.65	94.90 ↑	46.58 ↑
	Time:	0.001	0.02	0.14	0.26	0.001	≈ 0	1.87	1.5E3
	AUC:	55.13 ↑	70.62 ↑	53.76 ↑	60.78 ↑	89.57	98.91	95.44	46.06 ↑
	Time:	0.001	0.04	0.29	0.58	0.003	≈ 0	3.93	3.3E3
	AUC:	53.31 ↑	72.34 ↑	52.33 ↑	62.98 ↑	88.81	91.06 ↑	58.59 ↑	45.99 ↑
	Time:	0.003	0.07	0.52	1.06	0.003	≈ 0	5.39	4.3E3
	AUC:	49.10 ↑	69.61 ↑	51.64 ↑	64.22 ↑	88.93	81.92 ↑	52.79 ↑	46.00 ↑
	Time:	0.005	0.11	0.79	1.68	0.004	≈ 0	8.02	7.9E3
	AUC:	48.40 ↑	68.10 ↑	51.69 ↑	62.75 ↑	90.13	91.51 ↑	49.60 ↑	45.93 ↑

	Stages	Pima	Thyroid	Satellite	Optdigits	Ionosphere	Wine	SMTP
osPCA1	**Time:**	1.50	9.02	17.72	15.67	0.95	0.25	252.87
	AUC:	80.37	40.42 ↑	26.48 ↑	54.66 ↑	69.95 ↑	86.95 ↑	10.52 ↑
	Time:	1.58	9.24	16.17	16.49	1.20	0.24	252.88
	AUC:	75.10	45.96 ↑	42.43 ↑	57.19 ↑	61.24 ↑	60.86 ↑	17.25 ↑
	Time:	1.85	9.62	17.01	18.57	0.90	0.24	282.63
	AUC:	73.26	53.49 ↑	45.65 ↑	54.24 ↑	53.67 ↑	62.31 ↑	11.41 ↑
	Time:	1.85	9.33	18.49	19.32	0.87	0.25	302.31
	AUC:	72.00	52.91 ↑	47.32 ↑	51.65 ↑	50.92 ↑	67.11 ↑	18.37 ↑
	Time:	1.95	9.54	19.59	20.16	0.89	0.28	321.40
	AUC:	71.34	55.80 ↑	48.08 ↑	51.09 ↑	49.74 ↑	72.35 ↑	24.72 ↑
osPCA2	**Time:**	0.15	3.51	14.58	11.32	0.12	0.01	2101.3
	AUC:	50.94 ↑	49.93 ↑	49.94 ↑	49.95 ↑	48.88 ↑	47.82 ↑	49.99 ↑
	Time:	2.04	17.35	42.41	33.80	0.95	0.25	5346.3
	AUC:	56.10 ↑	57.00 ↑	54.03 ↑	46.14 ↑	57.02 ↑	58.69 ↑	58.03 ↑
	Time:	2.32	25.88	68.78	54.53	1.06	0.26	9979
	AUC:	59.42 ↑	64.59 ↑	56.33 ↑	40.87 ↑	61.58 ↑	67.63 ↑	68.00 ↑
	Time:	2.81	33.53	69.17	73.20	1.07	0.34	14416
	AUC:	60.02 ↑	64.78 ↑	57.23 ↑	38.97 ↑	62.84 ↑	72.14 ↑	69.13 ↑
	Time:	3.21	42.30	122.71	93.13	1.18	0.40	18678
	AUC:	61.05 ↑	67.19 ↑	57.36 ↑	36.60 ↑	64.51 ↑	75.04 ↑	68.82 ↑

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[3] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *NeurIPS conference*, 2000, pp. 582–588.

[4] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[5] ——, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, p. 3, 2012.

[6] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *2007 IEEE symposium on CIDM*. IEEE, 2007, pp. 504–515.

[7] T. Ahmed, "Online anomaly detection using KDE," in *2009 IEEE conference on global telecommunications*. IEEE, 2009, pp. 1–8.

[8] Y.-R. Yeh, Z.-Y. Lee, and Y.-J. Lee, "Anomaly detection via over-sampling principal component analysis," in *New Advances in Intelligent Decision Technologies*. Springer, 2009, pp. 449–458.

[9] Y.-J. Lee, Y.-R. Yeh, and Y.-C. F. Wang, "Anomaly detection via online oversampling principal component analysis," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 7, pp. 1460–1470, 2013.

[10] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends® in Computer Graphics and Vision*, vol. 7, no. 2–3, pp. 81–227, 2012.

[11] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial," *arXiv preprint arXiv:1905.12787*, 2019.

[13] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[14] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD*, vol. 2, 2000, pp. 71–80.

[15] H. Abdulsalam, D. B. Skillicorn, and P. Martin, "Streaming random forests," in *11th IDEAS 2007*. IEEE, 2007, pp. 225–232.

[16] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *2009 IEEE ICCV workshops*. IEEE, 2009, pp. 1393–1400.

[17] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian forests: Efficient online random forests," in *NeurIPS conference*, 2014, pp. 3140–3148.

[18] ——, "Mondrian forests for large-scale regression when uncertainty matters," in *Artificial Intelligence and Statistics*, 2016, pp. 1478–1487.

[19] D. M. Roy and Y. W. Teh, "The Mondrian process," in *NeurIPS conference*, 2008, pp. 1377–1384.

[20] B. R. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. John Wiley & Sons Incorporated, 2000.

[21] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[22] G. Louppe, "Understanding random forests: From theory to practice," Ph.D. dissertation, University of Liège, Faculty of Applied Sciences, 2014.

[23] S. Rayana, "Outlier detection data sets," http://odds.cs.stonybrook.edu/, 2019.

[24] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*, 2018, pp. 108–116.

[25] Canadian Institute for Cybersecurity, "Intrusion detection evaluation dataset (CICIDS2017)," https://www.unb.ca/cic/datasets/ids-2017.html, 2017.

[26] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[27] X. Fan, B. Li, and S. A. Sisson, "Binary space partitioning forests," in *22nd AISTATS conference*, vol. 89, 2019, pp. 1–10.

[28] ——, "The binary space partitioning-tree process," in *21st AISTATS conference*, vol. 84, 2018, pp. 1–9.