RS-Forest: A Rapid Density Estimator for Streaming Anomaly Detection

Ke Wu*, Kun Zhang*, Wei Fan†, Andrea Edwards* and Philip S. Yu‡
*Department of Computer Science, Xavier University of Louisiana, {kwu, kzhang, aedwards}@xula.edu
†Huawei Noah Ark's Lab, david.fanwei@huawei.com
‡Department of Computer Science, University of Illinois at Chicago, psyu@cs.uic.edu

***Abstract*—Anomaly detection in streaming data is of high interest in numerous application domains. In this paper, we propose a novel one-class semi-supervised algorithm to detect anomalies in streaming data. Underlying the algorithm is a fast and accurate density estimator implemented by multiple fully randomized space trees (RS-Trees), named RS-Forest. The piecewise constant density estimate of each RS-tree is defined on the tree node into which an instance falls. Each incoming instance in a data stream is scored by the density estimates averaged over all trees in the forest. Two strategies, statistical attribute range estimation of high probability guarantee and dual node profiles for rapid model update, are seamlessly integrated into RS-Forest to systematically address the ever-evolving nature of data streams. We derive the theoretical upper bound for the proposed algorithm and analyze its asymptotic properties via bias-variance decomposition. Empirical comparisons to the state-of-the-art methods on multiple benchmark datasets demonstrate that the proposed method features high detection rate, fast response, and insensitivity to most of the parameter settings. Algorithm implementations and datasets are available upon request.**

I. Introduction

Anomalies or outliers are rare events or items that are inconsistent with or deviate from those that are normal or expected. These abnormal items, if not identified promptly, could lead to devastating consequences in many practical applications including military surveillance, network security management, industrial system monitoring and control, etc. With the advances in hardware technologies, recent years have seen a dramatic increase in our ability to collect data continuously in those application domains. Most of the gathered data are no longer finite and stationary. Instead, they are unbounded sequences of large-volume, high-speed real-time data, referred to as data streams.

To date, anomaly detection has been the subject of numerous researches in the data mining community [1]. However, the inherent characteristics of data streams pose unparalleled challenges to a majority of the existing anomaly detectors. First, data are streaming in at unprecedented speed and hence must be processed in a timely manner. This requires that the rate of updating a detection model should be higher than the data rate, and the obtained detector must be able to adapt to the high speed nature of streaming information. Second, conventional anomaly detection algorithms need data to be resident in memory for model construction. This array of methods will be nullified by the voluminous unbounded data due to memory exhaustion. Third, in streaming data, normal and abnormal events keep evolving with the drifting concepts. Such incessant changes often outdate the detection models learned from old data. Therefore, the detector needs to quickly adjust to the evolution of normal behavior over time. Lastly, in practice, anomaly instances are rare or even not available in streaming data. Anomaly detection systems should be able to detect suspicious behaviors even if they were trained only on the normal events.

In response to these challenges, we propose a novel one-class semi-supervised algorithm for detecting anomalies in streaming data. Underlying this method is a fast and accurate density estimator driven by multiple RS-Trees, named RS-Forest. In RS-Forest, each tree can be randomly built in advance without data. Specifically, before the tree construction, a statistical mechanism is first employed to estimate the potential evolution of feature ranges throughout the to-be-mined data stream. Any value in the estimated range of a randomly picked attribute can be used to split the tree. The trees constructed in this way can not only accommodate the ever-evolving nature of streaming data, but also maximize the diversity of the resulting ensemble, leading to more accurate density estimation.

When applied to data streams, RS-forest operates in a fashion of single window. This window keeps the newly arrived instances waiting for detection. There are two major processes in streaming RS-forest. One is prediction or scoring, and the other is model update. Streaming RS-Forest deems that anomalies occur in sparse or low density regions, and anomalous instances are indicated by low density values. The anomaly score in streaming RS-forest is defined through the piecewise local density of the tree node into which an instance falls. Each incoming instance is then ranked by the average score gathered over all trees in the forest. Whenever the window is full, a model update will be triggered. To speed up model updates, RS-forest employs a dual node profile technique. This technique leverages the fixed tree structure so that capturing the node size profiles from newly arrived instances and making predictions for the same instances can be synchronized. The captured node size profiles are then used to score the next round of data arriving in the window. These two processes continue as long as data is streaming in. Compared to existing benchmark methods, streaming RS-forest features high anomaly detection rate, less runtime and rigorous theoretical foundation, as demonstrated in the rest of the paper.

The main contributions of this work are summarized as follows:

- We define and introduce a fast and accurate density estimation algorithm, RS-Forest;
- We extend the proposed algorithm to tackle the anomaly detection problem in the scenario of streaming data;
- Theoretical analysis provides the performance upper bound of the proposed algorithm and establishes its

1550-4786/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDM.2014.45

IEEE computer society

TABLE I: Major notations

Notation	Description
$\mathcal{X} = R^d$	Feature space with d dimensions
$\mathbf{x} \in \mathcal{X}$	An instance in feature space $\mathcal{X}$
$f(\mathbf{x})$	True density function
$\hat{f}(\mathbf{x}; T)$	Density estimated by an RS-Tree T
$\hat{f}(\mathbf{x}; F)$	Density estimated by an RS-Forest F
$\ell(\mathbf{x}, T)$	An RS-tree's termination node into which $\mathbf{x}$ falls
$\lvert \cdot \rvert$	Node size or node profile
$\mathcal{V}(\cdot)$	Volume of the region that a node represents
M	Number of the RS-trees in an RS-Forest
N	Number of instances, which is used to estimate density
H	Tree depth limit
l_j	Estimated length of attribute j
$V = \prod_{j=1}^{d} l_j$	Volume of the estimated feature space
δ_j^i	Length of attribute j that forms the region represented by node i
δ_{ij}	Length of attribute j that forms the region represented by a termination node $\ell(\mathbf{x}; T_i)$
$V_i = \prod_{j=1}^{d} \delta_j^i$	Volume of the region represented by node i
h_j^i	The number of splits performed on attribute j along the path from the root to node i
$r_{jk}^i \in (0, 1)$	The selected random number at the k-th split of attribute j along the path from root to node i
$r_k^i \in (0, 1)$	The random number selected at an internal node along the path from root to node i

asymptotic properties via Bias-Variance decomposition;

- Empirical studies on multiple benchmark datasets demonstrate that the proposed technique outperforms the competing methods on most of the datasets in terms of AUC score and runtime.

II. Method

This section provides the road map of the proposed algorithm. We first define a density estimator using multiple random space trees. Then we present the implementation details and algorithmic flow to extend it for anomaly detection in streaming data. Table I summarizes the major notations used in the paper.

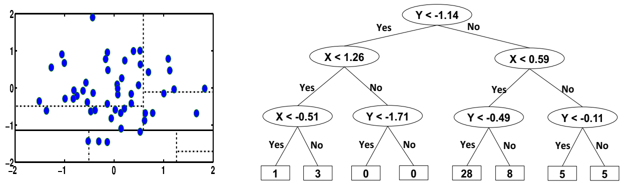
A. Basic definitions

Definition 2.1: A Random-Space Tree (RS-Tree) is a full binary tree, which is constructed by randomly selecting an attribute and a cut point on the chosen attribute.

An RS-Tree of depth $H (H \geq 0)$ has a total of $2^{H+1} - 1$ nodes, where leaf nodes are at the same depth H. The RS-tree stems from random decision trees [2] and the tree structure can be built without data. When constructing an RS-Tree, the algorithm recursively selects an attribute from the feature set and then picks a random dividing point of the chosen attribute to split the tree until the predefined tree depth H is reached. Figure 1b demonstrates an RS-Tree on a two-dimensional data (Figure 1a). It is worth noting that an RS-Tree considered in this work operates only in the continuous feature space.

Definition 2.2: A node profile in an RS-Tree is defined as the number of instances falling into the region that a node represents given a sample of data.

Then, we assume that an RS-Tree can perform density estimation over a sample of N instances from some unknown distribution. The estimation criterion is defined below.



(a) (b)

Fig. 1: (a) An example of two-dimensional data with random space partitions. (b) The RS-Tree corresponding to those partitions.

Definition 2.3: [3] The piecewise constant density estimate based on an RS-Tree T is defined as

$$\hat{f}(x; T) = \frac{|\ell(\mathbf{x}, T)|}{N\mathcal{V}(\ell(\mathbf{x}, T))} \tag{1}$$

where $\ell(\mathbf{x}, T)$ is the termination node, $| \cdot |$ is the node size or node profile, and $\mathcal{V}(\cdot)$ is the volume of the hyper-rectangle that a node represents. In an RS-tree, a termination node is the node used for density estimation. It can be a leaf node, or the first node that satisfies (i.e., $\leq$) the node size limit.

Furthermore, motivated by the triumph of ensembles on some fundamental tasks such as classification and regression, we propose to perform more accurate density estimation using multiple RS-trees, termed RS-Forest.

Definition 2.4: The density estimate based on an RS-Forest F is defined as

$$\hat{f}(x; F) = \frac{1}{M} \sum_{t=1}^{M} \hat{f}(x; T_t) \tag{2}$$

where $\hat{f}(x; T_t)$ denotes the density estimated by the t-th tree and M denotes the number of trees.

To illustrate the effectiveness of RS-Forest on density estimation, we apply it to a highly skewed distribution in one dimension. This distribution is defined as

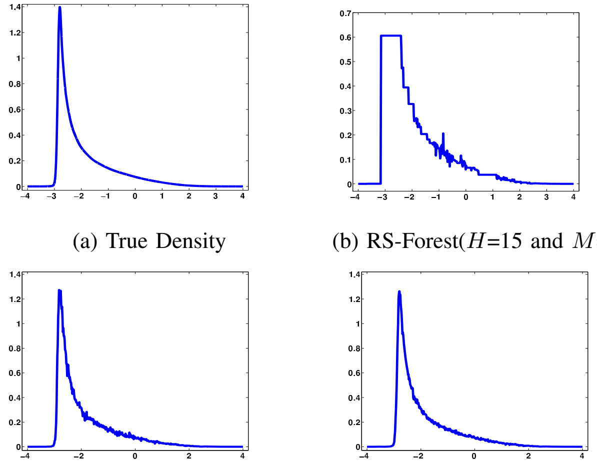
$$X \sim \sum_{i=0}^{7} \frac{1}{8} \mathcal{N}\left(3\left(\left(\frac{2}{3}\right)^i - 1\right), \left(\frac{2}{3}\right)^{2i}\right). \tag{3}$$

Figure 2 shows the estimated density functions by an RS-Forest with different number of trees on a sample of size 100,000. It is evident that, as more RS-Trees are involved, the densities estimated by RS-Forest more resemble the true density with a smoother fitting. Moreover, we compare RS-Forest with Kernel Density Estimation (KDE) on estimation accuracy and training and testing time. As illustrated in Figures 2 and 3, RS-Forest can perform a faster and more accurate estimation compared to classical KDE [1] implemented by a tree-based speedup.

B. RS-Trees

Tree node initialization. An RS-Tree with depth H consists of $2^{H+1} - 1$ nodes. In the implementation, each node contains the following elements: (1) variables m_l and m_r, which are used to alternately record the node profile in

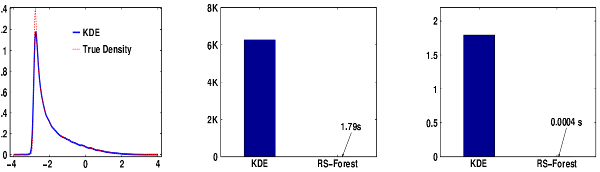
[1] KDE was performed using scikit-learn package (Http://scikit-learn.org/stable/). It is a tree-based implementation to speedup runtime. Training was conducted on the same sample data shown in Figure 2 and testing was done over 500 unseen points. The bandwidth was set to be 0.06, which was determined by a 20-fold cross validation.



(a) True Density (b) RS-Forest(H=15 and M=1)

(c) RS-Forest(H=15 and M=5) (d) RS-Forest(H=15 and M=30)

Fig. 2: Density estimation: RS-Forest



(a) Density (b) Training Time (c) Testing Time

Fig. 3: Density Estimation: KDE vs. RS-Forest

the prediction model or the node profile captured from the streaming data in the window; (2) three node pointers, each of which points to the left child, right child and parent of the current node, respectively; (3) a variable v that stores the log-scaled ratio of the current-node volume to the volume of the entire feature space.

Attribute range estimation. Determining the correct range of each attribute is a crucial step to build the structure of an RS-Tree. A tight range, such as the one directly borrowed from the sample data, can not cater for the potential changes throughout the streaming data. A wide range, on the other hand, could bring in undesired noise and futile space partitions. In this regard, we employ a statistical strategy to estimate a proper range for each attribute. More specifically, using Chen *et al.*'s algorithm [4] and a sample data, we first compute the 90% highest posterior density (HPD) confidence interval $[LM_i, UM_i]$ for the mean of attribute i. Then we enlarge this confidence interval roughly by the rule of three sigma [2], that is, we set the lower bound as $LB_i = LM_i - 3\sigma$ and the upper bound as $UB_i = UM_i + 3\sigma$, where σ is the standard deviation of attribute i. We analyze the reliability of the range estimation in Section III-A. The lower and upper bound values (i.e., LBs and UBs) generated in this way are used as inputs of Algorithm 1 to create the structure of a single RS-Tree. Hereafter, without specification, the range or length of an attribute is referred to the estimated range or length here.

Node volume calculation. Computing a node volume is essential for density estimation by an RS-Tree. Once an RS-Tree is built, the volume of each node is fixed. As shown in Figure 1a, a space Ω can be partitioned by an RS-Tree. Each node represents a hyper-rectangle formed by 2^d inequalities $\mathbf{e^T x} < c$, where $\mathbf{e}$ is a unit vector with only one element being 1 and others being 0, $\mathbf{x} \in \mathbb{R}^\mathbf{d}$ and $c \in \mathbb{R}$. To compute a node volume, we first introduce two lemmas.

Algorithm 1 BuildRS-TreeStructure(LBs, UBs, e, H)

```
Input: $LBs$ ($UBs$): list of lower (upper) bound values of attributes, $e$: current tree depth, and $H$: tree depth limit.
Output: a tree
if $e \geq H$ then
    return $leaf(m_l \leftarrow 0,\ m_r \leftarrow 0\ )$ ;
else
    initialize a non-leaf node $root$;
    randomly select an attribute $q \in Q$;
    randomly select a number $r$ between 0 and 1;
    cut-point $p$ = $LBs(q) + r \cdot (UBs(q) - LBs(q))$;
    $t \leftarrow UBs(q)$; $UBs(q) \leftarrow p$;
    $left \leftarrow$ BuildTreeStructure $(LBs, UBs, e+1, H)$;
    $left.v \leftarrow r$; $left.parent \leftarrow root$;
    $UBs(q) \leftarrow t$ ; $LBs(q) \leftarrow p$;
    $right \leftarrow$ BuildTreeStructure$(LBs, UBs, e+1, H)$;
    $right.v \leftarrow 1 - r$; $right.parent \leftarrow root$;
    return $root(leftChild \leftarrow left, rightChild \leftarrow right,\ splitAtt \leftarrow q, cutPoint \leftarrow p, m_l \leftarrow 0, m_r \leftarrow 0)$ ;
end if
```

Lemma 2.1: Let δ^i_j be the length of attribute j that forms the hyper-rectangle represented by node i, then we have $\delta^i_j = l_j \prod_{k=1}^{h^i_j} r^i_{jk}$, where l_j denotes the length of attribute j, $r^i_{jk} \in (0, 1)$ is the selected random number at the k-th split of attribute j along the path from the root to node i, h^i_j denotes the total number of splits performed on attribute j along the path from root to node i, and $\prod_{k=1}^{0} r^i_{jk} = 1$.

Proof: Given an interval $[a, b]$, a random number c in the interval can be generated by $a+s\cdot(b-a)$, where s is a random number in $[0, 1]$. The length of the interval [a, c] is $s \cdot l$ and the length of the interval [c, b] is $s' \cdot l$, where $l = b - a$ and $s' = 1 - s$. s' is also a random number in $[0, 1]$. For convenience sake, we use δ^i_{jk} to denote the length of attribute j after performing k splits on this attribute. At the first split of attribute j, the length of the interval to partition is the original one. Thus, we have $\delta^i_{j1} = l_j \cdot r^i_{j1}$. Assume that $s_1 \in [0, 1]$ is the random number picked at the parent node. When the current node is the left child, $r^i_{j1} = s_1$; Otherwise, $r^i_{j1} = 1-s_1$. At the second split of the same attribute, the length of the splitting interval is δ^i_{j1}. Hence we have $\delta^i_{j2} = \delta^i_{j1} \cdot r^i_{j2} = l_j \cdot r^i_{j1} \cdot r^i_{j2}$. Similarly, $r^i_{j2} = s_2$ for the left child, and $r^i_{j2} = 1 - s_2$ for the right child, where $s_2 \in [0, 1]$ is the randomly selected value at the parent node. Based on inductive reasoning, we have $\delta^i_{jh^i_j} = l_j \prod_{k=1}^{h^i_j} r^i_{jk} = \delta^i_j$. ■

Lemma 2.2: Let V_i be the volume of the hyper-rectangular region represented by node i, then we have $V_i = V \cdot \prod_{k=1}^{h_i} r^i_k$, where $r^i_k \in [0, 1]$ is the random number selected at an internal node along the path from root to node i, $V = \prod_{j=1}^{d} l_j$, $h_i = \sum_{j=1}^{d} h^i_j$ and $\prod_{k=1}^{0} r^i_k = 1$.

The proof is omitted due to page constraints. The above lemma

[2] http://en.wikipedia.org/wiki/68-95-99.7_rule

suggests that the volume of node i can be computed via the randomly selected numbers at the internal nodes along the path from the root to node i. Therefore, to compute a node volume, we just need to record those selected random numbers as we build an RS-Tree. See Algorithm 2 for the implementation details.

Algorithm 2 ComputeNodeVolRatio(T)

```
initialize a queue $node\_queue$;
$nodeQueue$.Enqueue($T$);
while ! $nodeQueue$.empty() do
    $curNode \leftarrow nodeQueue$.Dequeue();
    $parentNode \leftarrow curNode.parent$;
    if $parentNode$ is NULL then
        $curNode.v \leftarrow 0$;
    else
        $curNode.v \leftarrow parentNode.v+$ log($curNode.v$);
    end if
    if $curNode$ is an inner node then
        $nodeQueue$.Enqueue($curNode.leftChild$);
        $nodeQueue$.Enqueue($curNode.rightChild$);
    end if
end while
```

C. Streaming RS-Forest

The proposed method employs the density estimated by an RS-Forest to score an incoming instance and is coupled with a periodic fast model update for drifting concepts. In the implementation, the algorithm segments the streaming data into windows. Each window can be of equal or varying size depending on application requirements. The algorithm always operates in one window, called *the latest window*. At the initial stage of the anomaly detection process, the system builds tree (model) structures while recording the node profiles from the sample normal data. Then the system scores the incoming instances in the latest window as well as synchronously records the node profiles captured from those instances. As soon as the window is full, a model update is triggered and a fast model update is achieved by a technique using dual node profiles. Once the model update finishes, the old node profile is erased and the latest window is emptied for the newly arrived data. This process continues as long as data are streaming in.

Anomaly score. We score an incoming instance, $\mathbf{x}$, based on the density estimated by RS-Forest. The lower the density value of an instance, the higher the degree to which that instance is considered as an anomaly. Combining Definitions 2.3 and 2.4, we define the anomaly score of $\mathbf{x}$ as follows:

$$\frac{1}{M}\sum_{t=1}^{M}\frac{|\ell(\mathbf{x},T_t)|}{N\mathcal{V}(\ell(\mathbf{x},T_t))} \qquad (4)$$

Using Lemma 2.2, we can rewrite Eq. 4 as below:

$$\frac{1}{MV}\sum_{t=1}^{M} Score(\mathbf{x},T_t) \qquad (5)$$

where $Score(\mathbf{x},T_t) = \exp(\log(|\ell(\mathbf{x},T_t)|) - \ell(\mathbf{x},T_t).v - \log N)$ and $V = \prod_{j=1}^{d} l_j$. In practice, we only need to use $\sum_{t=1}^{M} Score(\mathbf{x},T_t)$ to rank anomalies as M and V are constants. Values are log-scaled to avoid underflow in computation.

Model Implementation. Algorithm 3 presents the implementation details of a streaming RS-Forest. Line 1 initializes an RS-Forest. In the initialization (Algorithm 4), obtainRange(X) is used to obtain the estimated range for each attribution q_i via a sample X. Refer to section II-B for details. X could be a sample of normal instances or the first ψ normal instances of the stream. The algorithm initializes the node profiles while building the structure of each RS-Tree. Line 2 in Algorithm 3 uses X to update the node profile m_l of each tree in the forest. In the streaming RS-Forest model, the procedures of prediction (or scoring) and model update occur alternately. In the entire execution process, the structure of each tree remains unchanged and the node profiles obtained from the scored data will be used for the model update in the next round. Therefore, there are some common operations shared by these two procedures. In this regard, we employ a dual node profile technique to save the same operations that have been done in the prediction phrase and will then be performed in the model update stage. Specifically, two node profiles, i.e., m_l and m_r, are alternately used to store the node profile for the current model (to score incoming data) and the node profile captured from the incoming data in the latest window (to update the model in the next round). After ψ data points are processed, the role of m_l and m_r changes and such a change is signaled by a variable LR.

As shown above, model update is one of the key procedures in a streaming RS-Forest. Its implementation is summarized in Algorithm 5. In general, there are two major functional blocks. One is for the anomalous instances (lines 3-9) and the other is for the normal instances (lines 10-19). In the former case of a anomalous instance, we update each RS-tree by reducing all node profiles along the path from the termination node to the root by one. In the latter case of a normal instance, if the termination node is not a leaf and the node size limit is not met, we update each RS-tree by increasing all node profiles along the path from the termination node to a leaf (or the first node satisfying the node size limit) by one. Here $NextChild(child, \mathbf{x})$ denotes the next child node that $\mathbf{x}$ falls into. It is worth noting that, once each instance is scored, streaming RS-Forest will receive the true label of the instance, which is the key defining characteristic of online learning [3].

Time and Space Complexities. As shown in algorithm 3, there are three primary operations in the main loop. They are the prediction or scoring (line 9), model update (line 15) and node profile reset (line 18). In prediction using the current model, each instance traverses from the root to the termination node in each tree. When updating the model, we take two different procedures with respect to each anomalous or normal instance. Because anomalous instances are rare, the time spent on the scoring and model update should be equal to or less than MH for each instance. Thus, the worse running time is $O(nMH)$ for n streaming data points. Resetting node profile is performed within at most $M \cdot 2^{(H+1)}$ whenever the model update is triggered. Its worse running time is $O(\frac{n}{\psi}M \cdot 2^{(H+1)})$. Considering that M, H and ψ are fixed during the running time, we can regard the worse time complexity of streaming RS-Forest as $O(n)$.

In terms of space complexity, the forest itself occupies

[3] http://en.wikipedia.org/wiki/Online_algorithm

Algorithm 3 StreamingRS-Forest(X, M, H, ψ, ζ)

```
Input: $M$: number of RS-trees, $H$: tree depth limit, $\psi$: window size, and $\zeta$: node size limit.
Output: s: anomaly score for each incoming instance $\mathbf{x}$
$F$ = BuildForest($X, M, H$);
update node profile $m_l$ of each RS-Tree in $F$ using $X$;
$B \leftarrow [\,]$;
$LR \leftarrow False$;
while data stream continues do
    Receive a new data point $\mathbf{x}$: $b.X = \mathbf{x}, b.nodeList = [\,]$;
    $s \leftarrow 0$;
    for each tree $T$ in $F$ do                    ▷ prediction phrase
        $s \leftarrow s + Score(\mathbf{x}, T)$;
        $b.nodeList$.Enqueue($\ell(\mathbf{x}, T)$);
    end for
    $B.Enqueue(b)$ ;
    Report the anomaly score $s$ of data point $\mathbf{x}$;
    if $|B| == \psi$ then                          ▷ model update phrase
        UpdateModel($F, B, LR, \zeta$);
        $LR \leftarrow\, !LR$;
        $B$.clear();
        $LR\,?\ node.m_l \leftarrow 0 :\ node.m_r \leftarrow 0$ for non-zero nodes of each tree $T$ in $F$;
    end if
end while
```

Algorithm 4 BuildForest(X, M, H)

```
($LBs, UBs$)= obtainRange($X$);
Initialize: $F = [\,]$;
for $t = 1$ to $M$ do
    $T$ = BuildTreeStructure($LBs, UBs, 0, H$);
    ComputeNodeVol($T$);
    $F = F \cup T$;
end for
return $F$;
```

$O(M2^H)$. The data buffer B in the latest window takes at most $O(\psi M)$. Since both ψ and M are small constants, the space occupation can be ignored. Therefore, the space complexity is $O(2^H)$. In the execution, H is fixed, so memory usage of RS-Forest is constant for streaming data.

III. Analysis

A. Reliability of Range Estimation

The range of each attribute is one of the key inputs to construct an RS-tree structure. Refer to II-B for details. An RS-tree, built in the d dimensional space bounded by the attribute ranges, needs to accommodate the potential feature drifts throughout the to-be-learned data stream. Therefore, we introduce the following theorem for the reliability analysis of the employed range estimation strategy.

Theorem 3.1: [5] When the distribution is unknown

$$p(\mathrm{L} < Z < \mathrm{U}) \geq \frac{4\left((\mu - \mathrm{L})(\mathrm{U} - \mu) - \sigma^2\right)}{(\mathrm{U} - \mathrm{L})^2} \tag{6}$$

where Z is a random variable, L and U are constants, $\mathrm{L} < \mu < \mathrm{U}$, $(\mu - \mathrm{L})(\mathrm{U} - \mu) > \sigma^2$, μ is the mean and σ^2 is the variance.

Algorithm 5 UpdateModel(F, B, LR, ζ)

```
Input: $F$: RS-Forest, $B$: buffer for each instance, $LR$: indicator for which profile will be updated, and $\zeta$: node size limit.
Output: None
split $B$ into lists $N$ and $A$ based on label info.;    ▷ $N(A)$ contains info. of normal (anomalous) instances in the latest window.
for i = 1 to $M$ do
    for each $\mathbf{x}$, $nodeList$ in $A$ do
        $ancestor = nodeList[i]$;
        while ($ancestor$ not exists ) do
            $LR\,?\ ancestor.m_l$ -- : $ancestor.m_r$ --;
            $ancestor \leftarrow ancestor.parent$;
        end while
    end for
    for each $\mathbf{x}$, $nodeList$ in $N$ do
        $LR\,?\ n = nodeList[i].m_l :\ n = nodeList[i].m_r$;
        if $n > \zeta$ && $nodeList[i]$ not leaf then
            $child = NextChild(child, \mathbf{x})$;
            while $child$ not NULL do
                $LR\,?\ child.m_l$++ : $child.m_r$++;
                $child = NextChild(child, \mathbf{x})$;
            end while
        end if
    end for
end for
```

The above theorem provides a probability guarantee for the range, which satisfies $(\mu - \mathrm{L})(\mathrm{U} - \mu) > \sigma^2$, around the mean of a variable. Here the variable can follow any distribution. In our algorithm, we define a range of attribute i to be $(LM_i - 3\sigma, UM_i + 3\sigma)$. For a normal distribution, 90% HPD confidence interval of the mean is approximately $(\mu - 1.645\sigma, \mu + 1.645\sigma)$. In our case, $\mathrm{L} = \mu - 4.645\sigma$ and $\mathrm{U} = \mu + 4.645\sigma$. Based on the three-sigma rule, $p(\mu - 4.645\sigma < Z < \mu + 4.645\sigma) \approx 1$. According to Equation 6, we have $p(\mu - 4.645\sigma < Z < \mu + 4.645\sigma) \geq 0.954$. As a result, we can empirically use the introduced method to estimate the range of an attribute as it offers a high probability guarantee.

B. Upper Bound of Density Estimation

To measure the performance of RS-Forest as a density estimator, we use Mean Squared Error (MSE) to quantify the deviation between the estimation $\hat{f}(\mathbf{x})$ and the true probability density $f(\mathbf{x})$.

Theorem 3.2: Let $\hat{f}(\mathbf{x}; F) = \mathbb{E}_T \hat{f}(\mathbf{x}; T)$ be the density estimated by RS-Forest F, then we have

$$\mathbb{E}_{\mathbf{x}}\left[\hat{f}(\mathbf{x}; F) - f(\mathbf{x})\right]^2 \leq \mathbb{E}_T \mathbb{E}_{\mathbf{x}}\left[\hat{f}(\mathbf{x}; T) - f(\mathbf{x})\right]^2 \tag{7}$$

Proof:

$$\begin{aligned} & \mathbb{E}_T \mathbb{E}_{\mathbf{x}}\left[\hat{f}(\mathbf{x}; T) - f(\mathbf{x})\right]^2 \\ = \; & \mathbb{E}_T \mathbb{E}_{\mathbf{x}}\left[\hat{f}^2(\mathbf{x}; T) - 2\hat{f}(\mathbf{x}; T) f(\mathbf{x}) + f^2(\mathbf{x})\right] \end{aligned} \tag{8}$$

Using the inequality $\mathbb{E}(W)^2 \geq (\mathbb{E}W)^2$, we obtain

$$\mathbb{E}_T \mathbb{E}_{\mathbf{x}} \hat{f}^2(\mathbf{x}; T) \geq \mathbb{E}_{\mathbf{x}} \Big[\mathbb{E}_T \hat{f}(\mathbf{x}; T) \Big]^2 \tag{9}$$

Plugging Eq. 9 into Eq. 8 and rearranging the terms in Eq. 8, we can derive Eq. 7.

■

Theorem 3.2 provides a performance upper bound of using an RS-Forest to estimate the density. It reveals that the MSE of the density estimated by RS-Forest is not greater than the average MSEs of density estimated by all member RS-Trees.

C. Asymptotic Analysis of Density Estimation

To get further insights of the density estimated by an RS-Forest, we also conduct the asymptotic analysis using MSE as defined below:

$$R = \mathbb{E}[(\hat{f}(\mathbf{x}; F) - f(\mathbf{x}))^2] \tag{10}$$

This function can be decomposed into two terms, i.e., the squared bias and the variance of the estimator. Thus, we rewrite Equation 10 as:

$$\begin{aligned} R &= \text{Bias}^2(\hat{f}, f) + \text{Var}(\hat{f}) \\ &= \Big[\mathbb{E}[\hat{f}(\mathbf{x}; F)] - f(\mathbf{x}) \Big]^2 + \mathbb{E}\left[\Big[\hat{f}(\mathbf{x}; F) - \mathbb{E}[\hat{f}(\mathbf{x}; F)] \Big]^2 \right] \end{aligned} \tag{11}$$

Before formally analyzing these two terms, we first introduce two lemmas. The following lemma approximates the probability of an instance falling into a node based on the node volume and the length of each attribute that forms the hyper-rectangle represented by a node.

Lemma 3.1:

$$\int_{\mathbf{x} \in \mathcal{N}} f(\mathbf{x}) d\mathbf{x} \approx f(\mathbf{a}) \mathcal{V}(\mathcal{N}) + \frac{1}{24} \sum_{j=1}^{d} \frac{\partial^2 f(\mathbf{a})}{\partial x_j^2} {\delta_j}^2 \mathcal{V}(\mathcal{N}) \tag{12}$$

where $\mathcal{N}$ denotes a node itself as well as the region it represents , δ_j is the length of attribute j that forms the region represented by node $\mathcal{N}$, and $\mathbf{a}$ is the midpoint of the region defined by node $\mathcal{N}$.

Proof:

After applying the second-order Taylor approximation of $f(\mathbf{x})$ around $\mathbf{a}$ for $\mathcal{N}$, we have

$$f(\mathbf{x}) \approx f(\mathbf{a}) + (\mathbf{x} - \mathbf{a})^T J(\mathbf{a}) + \frac{1}{2} (\mathbf{x} - \mathbf{a})^T H(\mathbf{a}) (\mathbf{x} - \mathbf{a}) \tag{13}$$

where $J(\mathbf{a})$ is the gradient of f and $H(\mathbf{a})$ is the Hessian matrix, both of which are evaluated at $\mathbf{x} = \mathbf{a} = (a_1, a_2, \ldots, a_d)$. Hence,

$$\int_{\mathbf{x} \in \mathcal{N}} f(\mathbf{x}) d\mathbf{x} = \int_{a_d - \delta_d/2}^{a_d + \delta_d/2} \cdots \int_{a_1 - \delta_1/2}^{a_1 + \delta_1/2} f(\mathbf{x}) dx_1 \ldots x_d \tag{14}$$

By Plugging Eq. 13 into Eq. 14 and then applying U-substitution technique, we can derive Equation 12 since the integral of an odd function over any symmetric space is zero.

■

The lemma below relates the expectation of density estimated by RS-Forest, $\hat{f}(x; F)$, to the true density.

Lemma 3.2:

$$\mathbb{E}[\hat{f}(\mathbf{x}; F)] = \frac{1}{M} \sum_{i=1}^{M} \frac{\int_{u \in \ell(\mathbf{x}, T_i)} f(u) du}{\mathcal{V}(\ell(\mathbf{x}, T_i))} \tag{15}$$

The proof is omitted due to page constraints. Based on these two lemmas, we can derive the following theorem regarding the bias of RS-Forest as a density estimator.

Theorem 3.3:

$$\text{Bias}^2(\hat{f}, f) \leq \frac{1}{M^2} \left\{ \sum_{i=1}^{M} \sum_{j=1}^{d} \left(C_{ij} {\delta_{ij}}^2 + C'_{ij} \delta_{ij} \right) \right\}^2 \tag{16}$$

where $\mathbf{a}_i$ is the midpoint of the region defined by node $\ell(\mathbf{x}, T_i)$, δ_{ij} is the length of attribute j that forms the region represented by node $\ell(\mathbf{x}, T_i)$, $C_{ij} = \frac{1}{24} \left| \frac{\partial^2 f(\mathbf{a}_i)}{\partial x_j^2} \right|$ and $C'_{ij} = \frac{1}{2} \left| \frac{\partial f(\mathbf{a}_i)}{\partial x_j} \right| + \frac{1}{8} \left| \frac{\partial^2 f(\mathbf{a}_i)}{\partial x_j \, \partial x_k} \right| \sum_{k=1}^{d} \delta_{ik}$.

Proof: Applying Lemma 3.2, we have

$$\text{Bias}^2(\hat{f}, f) = \left[\frac{1}{M} \sum_{i=1}^{M} \frac{\int_{u \in \ell(\mathbf{x}, T_i)} f(u) du}{\mathcal{V}(\ell(\mathbf{x}, T_i))} - f(x) \right]^2 \tag{17}$$

According to Lemma 3.1, the above equation can be rewritten as

$$\begin{aligned} \text{Bias}^2(\hat{f}, f) \approx & \left[\frac{1}{M} \sum_{i=1}^{M} \left[\frac{1}{24} \sum_{j=1}^{d} \frac{\partial^2 f(\mathbf{a}_i)}{\partial x_j^2} {\delta_{ij}}^2 - (\mathbf{x} - \mathbf{a}_i)^T J(\mathbf{a}_i) \right. \right. \\ & \left. \left. - \frac{1}{2} (\mathbf{x} - \mathbf{a}_i)^T H(\mathbf{a}_i) (\mathbf{x} - \mathbf{a}_i) \right] \right]^2 \end{aligned} \tag{18}$$

Also, $\mathbf{a}_i$ is the midpoint of the region defined by node $\ell(\mathbf{x}, T_i)$.

$$\begin{aligned} \text{Bias}^2(\hat{f}, f) \leq & \frac{1}{M^2} \left\{ \sum_{i=1}^{M} \left[\frac{1}{24} \sum_{j=1}^{d} \left| \frac{\partial^2 f(\mathbf{a}_i)}{\partial x_j^2} \right| {\delta_{ij}}^2 + \frac{1}{2} \sum_{j=1}^{d} \left| \frac{\partial f(\mathbf{a}_i)}{\partial x_j} \right| \delta_{ij} \right. \right. \\ & \left. \left. + \frac{1}{8} \sum_{j=1}^{d} \sum_{k=1}^{d} \left| \frac{\partial^2 f(\mathbf{a}_i)}{\partial x_j \, \partial x_k} \right| \delta_{ij} \delta_{ik} \right] \right\}^2 \end{aligned} \tag{19}$$

After some algebra, we can immediately derive Equation 16 from the above inequality.

■

This theorem indicates that the number of RS-trees and the length of each attribute that forms the region represented by a termination node $\ell(\mathbf{x}, T_i)$ are key factors in determining bias. The bias of an RS-Forest can be reduced by more RS-Trees. To further explore the role of the attribute length, we introduce the next lemma.

Lemma 3.3: Let δ_j^i be the length of attribute j that forms the region represented by node i in an RS-Tree, we have $\delta_j^i \leq$

$l_j \cdot \left(\frac{\sum_{k=1}^{h_j^i} r_{jk}^i}{h_j^i}\right)^{h_j^i}$, where $r_{jk}^i \in (0,1)$ is a random variable following a uniform distribution.

The proof will be given in the longer version of the paper. Since the random variable r_{jk}^i follows the uniform distribution, we have $\mathbb{E}[\sum_{k=1}^{h_j^i} r_{jk}^i] = \frac{h_j^i}{2}$. For simplicity of derivation, we conjecture that $\delta_j^i \leq l_j \cdot 2^{-h_j^i}$, for a large h_j^i according to the law of large numbers. Since the splitting attribute at a node is randomly chosen, the probability of selecting an attribute is approximately the same for a large depth limit H. As a result, we have $\delta_j^i \leq l_j \cdot 2^{-H/d}$ given a large H.

Next, we use the approximated result to derive an asymptotic property from Theorem 3.3. Since $\delta_j^i \leq l_j \cdot 2^{-H/d}$, we have $C_{ij}{\delta_{ij}}^2 + C'_{ij}\delta_{ij} \leq C_{ij}l_j^2 2^{-2H/d} + C'_{ij}l_j 2^{-H/d}$. Note that l_j and d are constants, and δ_{ij} is different from δ_j^i which denotes a more general node. Therefore, for a large H and a constant M, we have Bias$^2 \approx O(4^{-H/d})$. This shows that when H increases, the squared bias decreases.

In the following, we introduce the theorem regarding the variance of RS-Forest as a density estimator.

Theorem 3.4:

$$\text{Var}(\hat{f}) \leq \frac{1}{4MN}\sum_{i=1}^{M}\left(\frac{1}{\mathcal{V}(\ell(\mathbf{x},T_i))}\right)^2 \qquad (20)$$

Proof: Using Cauchy's inequality, we can conclude that

$$\begin{aligned} \text{Var}(\hat{f}) &= \mathbb{E}[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]]^2 \\ &= \frac{1}{M^2}\mathbb{E}\left[\sum_{i=1}^{M}\left(\hat{f}_i(x) - \mathbb{E}\hat{f}_i(x)\right)\right]^2 \\ &\leq \frac{1}{M}\sum_{i=1}^{M}\text{Var}(\hat{f}_i(x)) \end{aligned} \qquad (21)$$

To simplify the notation, let $v_i = |\ell(\mathbf{x},T_i)|$. v_i follows a Binomial distribution. Hence Var$[v_i] = Np(v_i)(1-p(v_i))$.

According to Definition 2.3, we have

$$\begin{aligned} \text{Var}(\hat{f}_i(x)) &= \frac{\text{Var}(v_i)}{N^2\left[\mathcal{V}(\ell(\mathbf{x},T_i))\right]^2} \\ &= \frac{p(v_i)(1-p(v_i))}{N[\mathcal{V}(\ell(\mathbf{x},T_i))]^2} \end{aligned} \qquad (22)$$

Plugging Eq. 22 into Eq. 21 we have

$$\begin{aligned} \text{Var}(\hat{f}) &\leq \frac{1}{MN}\sum_{i=1}^{M}\frac{p(v_i)(1-p(v_i))}{\mathcal{V}(\ell(\mathbf{x},T_i))^2} \\ &\leq \frac{1}{4MN}\sum_{i=1}^{M}\left(\frac{1}{\mathcal{V}(\ell(\mathbf{x},T_i))}\right)^2 \end{aligned}$$

■

This theorem shows that the variance of RS-Forest depends on three factors, i.e., the number of RS-trees, the data sample size, and the sum of $\left(\frac{1}{\mathcal{V}(\ell(\mathbf{x},T_i))}\right)^2$, which is the squared reciprocal of the termination node's volume. When the number of RS-trees and / or the size of data sample increase, the variance decreases. Moreover, intuitively, when H increases, the value of $\mathcal{V}(\ell(\mathbf{x},T_i))$ becomes small and thus the variance increases.

IV. Experiments

Using standard evaluation metrics, we empirically compare streaming RS-forest with four baseline methods on thirteen synthetic and real-world benchmark datasets. For simplicity's sake, streaming RS-forest is abbreviated as RS-forest in this section.

A. Datasets

Table II summarizes the characteristics of six synthetic and seven real-world datasets used in our experiments. The procedures of each dataset preparation are presented below.

Synthetic Data. ***Mulcross*** [6] features a multi-variate normal distribution with two dense anomaly clusters hard to detect. We used the same parameter settings as [7] to generate this set. ***Synthetic Stream*** The concept in this data stream is defined as $g(\mathbf{x}) = \sum_{i=1}^{d} a_i \cdot x_i \cdot x_{d-i+1} - a_0$, where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $p(x_i) \sim \mathcal{N}(\mu_i, \delta_i)$. Similar to [8], we created three datasets, i.e., Syn_feature, Syn_cond and Syn_dual. Each of them respectively simulates a different type of concept drift, that is, feature change $p(\mathbf{x})$, conditional change $p(y|\mathbf{x})$ and dual change $p(\mathbf{x}, y)$. ***HyperP1 and HyperP2*** were generated from a multi-class synthetic data stream[4], which contains gradually drifting concepts. In HyperP1, class 3 is treated as the anomalous class. While for HyperP2, instances from class 4 are considered as anomalies.

Real-world Data. We adopted ***Smtp, Http and Smtp_http*** from KDDCUP 99 [9]. We created Smtp_http by combining Smtp with Http so that there would be a sudden concept drift when the communication protocol changes from Smtp to Http. We obtained ***Covertype, Shuttle and Adult*** from UCI [10]. All three sets are commonly employed to evaluate data stream algorithms. In particular, to produce a data stream with sufficient concept drifts, we created 14 data chunks from Adult according to the unique values of the occupation attribute before merging them into a single data stream. By doing so, the created stream can represent multiple distinct concepts. The NOAA ***Weather*** dataset spans 51 years [11]. To denote scarce anomalies, we under-sampled the instances in no_rain class from 31% to 4.83%.

B. Experimental Setup

We compared the performance of RS-Forest with several baseline methods, namely HSTa [9], LOADED [12], Hoeffding Trees (HT) [13], and online coordinate boosting with HT (BoostHT) [14]. **HSTa**[5] is a recently proposed one-class anomaly detector for evolving data streams. For HSTa, we tried different model parameters(i.e., window size =[64, 128, 250, 512], ensemble size = [25, 30], and tree depth = 15) and reported its best result for each dataset. **LOADED** [12] is

[4] http://www.cse.fau.edu/˜xqzhu/stream.html

[5] https://sites.google.com/site/analyticsofthings/recent-work-fast-anomaly-detection-for-streaming-data

TABLE II: Benchmark datasets

data sets	#inst.	#feat.	Anomaly
Mulcross	262,144	4	2 dense clusters(10%)
Syn_feature	51,000	10	class <0 (1.0%)
Syn_cond	51,000	10	class <0 (1.0%)
Syn_dual	51,000	10	class <0 (1.0%)
HyperP1	100,000	10	class C3 (10.81%)
HyperP2	100,000	10	class C4 (17.71%)
Http	567,497	3	attack (0.4%)
Smtp	95,156	3	attack (0.03%)
Smtp_http	662,653	3	attack (0.35%)
Covertype	286,048	10	outlier (0.9%)
Shuttle	49,097	9	classes 2,3,5-7 (7%)
Adult	35,760	6	class >50k(3.21%)
Weather	13,094	8	class no_rain(4.83%)

a link-based unsupervised anomaly detector aiming to handle mixed attributes. The original anomaly score of LOADED is defined for handling both categorical and continuous attributes. For fair comparison purposes, we dropped their score mechanism for categorical attributes (i.e., condition check C1) and updated the correlation matrix using only normal instances. As recommended by [12], we set the parameter τ to be 1.96. **HT** and **BoostHT** are two state-of-the-art classification algorithms for data streams. In the experiments, we used their Java implementations from MOA[6]. HT is executed with the default parameters and the smoothing parameter of BoostHT is set as 0.5. We implemented **RS-Forest** in C++. Without specification, the default parameters of RS-Forest are set as follows. Window size=512, ensemble size = 30, and tree depth = 15. We used AUC, the area under an ROC curve, to evaluate the performances of all algorithms. 30 independent runs were conducted by each method on a dataset before the average result was reported. We performed all the experiments on a 2.6 GHz Intel Core i7 MacBook Pro with 8GB main memory.

C. Experimental Results

Comparative Results. Table III presents the average AUCs of all compared algorithms on each benchmark dataset. It is evident that RS-Forest statistically significantly outperforms other methods on most of the datasets. Based on the paired t-test with a 95% confidence interval, the pairwise win-loss-tie statistics between RS-Forest and HSTa, LOADED, HT, BoostHT are 10-0-3, 10-3-0, 13-0-0, 11-1-1, respectively. We attribute this to our better strategies of detecting anomalies in the context of streaming data. HSTa is no doubt a competitive baseline, hitting the second place in terms of the averaged AUC over all data. BoostHT works well on most of the real-world data. But its AUC scores on some synthetic sets including Syn_cond and Syn_dual are even worse than those of HT. This performance degradation may indicate that over-fitting could easily happen to BoostHT when the conditional class distribution changes. Being the single-model algorithms, LOADED and HT are at the bottom with respect to the AUC ranking. Different from BoostHT, LOADED performs surprisingly well only on some synthetic sets generated from Gaussian distribution, such as Syn_feature, Syn_cond and Syn_dual. This could be due to the strict model assumption that underlies LOADED.

Moreover, as shown in Figure 4, we further examine the dynamic performances of the three ensembles with the progression of a data stream. Without much variation, the over-time AUC curves of RS-Forest consistently lead the corresponding curves of HSTa and BoostHT on Mulcross and HyperP1 datasets as each stream proceeds. Performance fluctuation does happen on Covertype. In particular, as the anomalous instances occur in the middle stage of the Covertype progression (highlighted by the dashed lines), the curves of RS-Forest and HSTa tend to go down while the curve of BoostHT moves up abruptly. HSTa obviously demonstrates a larger downwards trend than RS-Forest. A close look via the snapshots in Figure 5 suggests that intensively occurring anomalies, though rarely happen in reality, could deteriorate the performance of HSTa. RS-Forest can better adapt to this situation and its overall curve still dominates the curves of the other two.

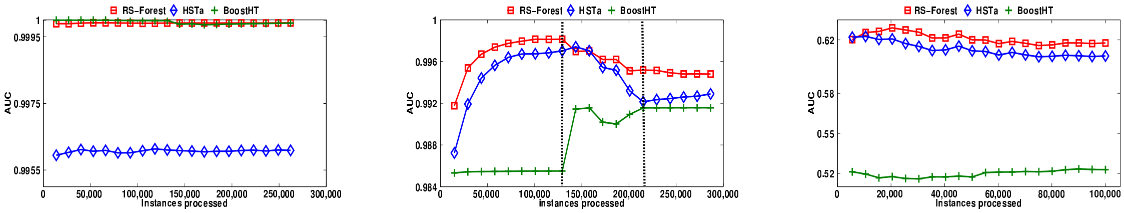
data sets	RS-Forest	HSTa	LOADED	HT	BoostHT
Mulcross	1.000	0.998 ●	0.999 ●	0.902 ●	1.000
Syn_feature	0.897	0.885 ●	0.912 ○	0.515 ●	0.536 ●
Syn_cond	0.899	0.886 ●	0.914 ○	0.589 ●	0.535 ●
Syn_dual	0.893	0.888 ●	0.917 ○	0.639 ●	0.533 ●
HyperP1	0.618	0.610 ●	0.597 ●	0.519 ●	0.523 ●
HyperP2	0.760	0.752	0.622 ●	0.629 ●	0.772 ○
Http	0.999	0.996 ●	0.500 ●	0.909 ●	0.997 ●
Smtp	0.880	0.875	0.500 ●	0.696 ●	0.833 ●
Smtp_http	0.998	0.996 ●	0.500 ●	0.589 ●	0.994 ●
Covertype	0.995	0.993 ●	0.500 ●	0.655 ●	0.992 ●
Shuttle	0.998	0.999	0.501 ●	0.770 ●	0.987 ●
Adult	0.645	0.639 ●	0.500 ●	0.573 ●	0.597 ●
Weather	0.593	0.582 ●	0.500 ●	0.489 ●	0.512 ●
Average	**0.860**	0.854	0.651	0.652	0.755

TABLE III: Performance comparison of different methods on all benchmark datasets. AUC score is measured. The last row reports the average AUC over all datasets for each method. ●/○ indicates that RS-Forest performs significantly better/worse than the corresponding method regarding a paired t-test with a 95% confidence interval.

Runtime Comparison. Figure 6 summarizes the average time of 30 independent runs for each method on all datasets. LOADED and HT cost moderate time since they are single-model learners. The BoostHT ensemble consumes the most amount of time as its member tree (HT) always relies on computing Hoeffding bound to determine the best attribute to split the tree. HSTa performs faster than BoostHT but its average time consumption is still approximately five times that of RS-Forest. RS-Forest uses a data structure similar to HSTa. However, unlike HSTa, RS-Forest further reduces the model update time by leveraging the common operations shared by the prediction and model update processes. Therefore, the runtime of RS-Forest is the least among all the studied methods, even the ones of single model.

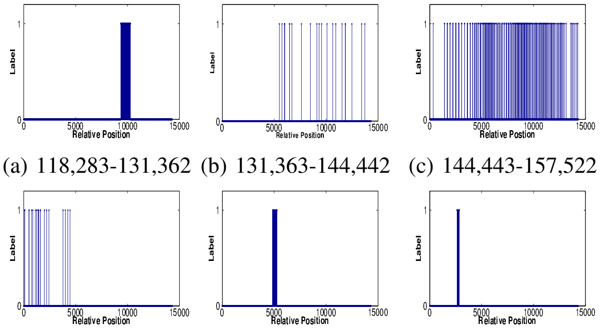
Sensitivity Study. To run RS-Forest, one needs to specify several parameters. They are the node size limit, the window size and the number of trees. Taking the Covertype dataset as an example, we further explore the effects of different parameter settings on the performance of RS-Forest. As shown in Figure 7, the effects of other parameters on RS-Forest's performance diminish as more RS-Trees are introduced into the forest, which well corroborates with the theoretical analysis presented in section III-C. When the number of RS-Trees increases, both bias and variance of RS-Forest can be reduced, leading to a more accurate density estimation. Moreover, we studied the relationship between different window sizes and varying node size limits. We can see that the node size limit

[6] http://moa.cms.waikato.ac.nz



(a) Mulcross dataset (b) Covertype dataset (c) HyperP1 dataset

Fig. 4: Dynamic performance comparison with the progression of a data stream.



(a) 118,283-131,362 (b) 131,363-144,442 (c) 144,443-157,522

(d) 157,523-170,602 (e) 170,603-183,682 (f) 183,683-196,762

Fig. 5: Illustration of anomaly distribution in consecutive data segments over Covertype dataset. Label 0: normal instances. Label 1: anomaly instances.

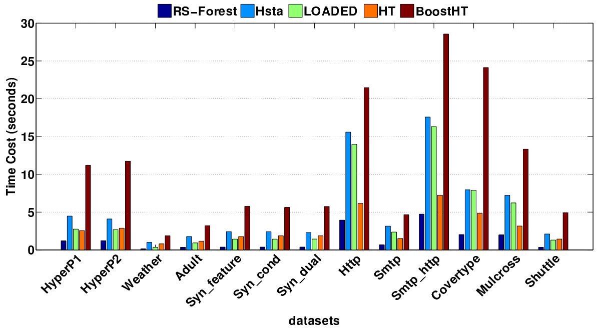
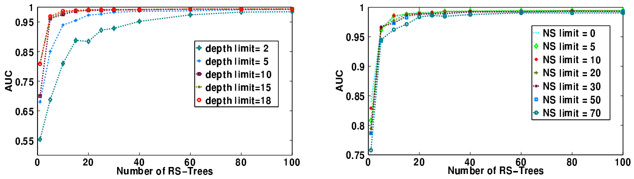


Fig. 6: Runtime comparison on all benchmark datasets.

won't influence the performance of RS-Forest much once the window size is fixed. Also, we notice that, due to the underlying concept drift, a large window size does not always result in a performance gain. When the window size is larger than 512, the performance of RS-Forest even declines. This suggests that choosing an appropriate window size would be crucial in the context of drifting concepts. We recommend 512 as the default value according to our observations also on other datasets.

V. Related Work

In the past few decades, many algorithms have been proposed for anomaly or outliner detection. Typical examples include model-based methods [15][16][7], clustering-based methods [17], distance-based methods [18] and density-based methods [19]. However, most of the techniques, designed for static finite stored data, require the entire data to reside in the memory and exhibit a high computational cost. Therefore, they are not readily to be applied to the ever-evolving streaming data, which features large volume, high velocity and drifting concepts.



(a) Effect of depth limit (b) Effect of node size limit

Fig. 7: Effects of varying depth limit and node size limit

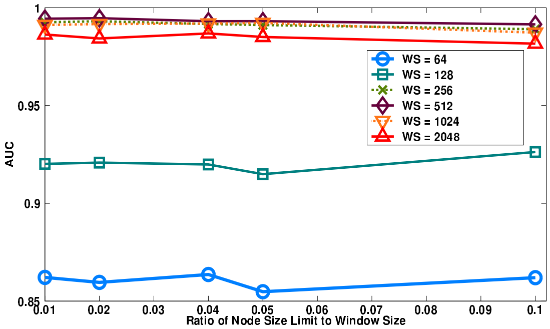


Fig. 8: Effects of varying window size

Recently, much effort [20][21][22] has been devoted to improving the detection efficacy of the distance-based methods in the context of the sliding window model for data streams. These studies primarily deal with continuous queries, which are queries evaluated continuously as data instances arrive incessantly. In this work, we focus on handling one-time queries evaluated over a point-in-time. Refer to [23] regarding the query model in data streams. Another related work is LOADED proposed by Ghoting *et al.* [12]. It is a one-pass unsupervised anomaly detector that aims to handle the streaming data with mixed attributes. A unified link-based approach is employed to capture the dependencies between the continuous and categorical attributes.

Compared to the aforementioned unsupervised methods, fewer supervised or semi-supervised data stream algorithms have been proposed for anomaly detection. Seminal data stream classifiers, such as Hoeffding Trees (HT)[13] and online coordinate boosting with Hoeffding Trees (BoostHT) [14], can be adapted to this purpose. Nevertheless, such algorithms require both normal and anomalous instances for training, and in practice, anomalous data are typically rare or even not available.

Tan *et al.*'s work [9] is most closely related to ours. They developed an ensemble of streaming Half-Space-Trees (HSTa) to detect anomalies in data streams. Each HS-Tree is created by bisecting a randomly picked continuous attribute from an estimated feature space using a heuristic technique. An anomaly score defined on a mass statistic is then aggregated over all trees to rank newly arrived instances. Unlike HSTa, our algorithm uses multiple fully randomized space trees to tackle the streaming detection problem from the density estimation aspect, which has a solid theoretical foundation as shown in this work. Moreover, our algorithm is more efficient as it leverages the common operations shared by the prediction and model update processes.

In addition, the proposed RS-Forest density estimator is somewhat analogous to other two methods, density estimation trees (DET) [3] and forest density estimation [24]. However, DET performs the optimal tree construction based on a predefined target function and forest density estimation is an undirected graph model forest using data splitting. Therefore their tree construction relies on training data and thus has a high computational cost. Unlike these two methods, RS-Forest constructs trees without data and hence is efficient and suitable for streaming data mining.

VI. Conclusion

In this paper, we have introduced a novel anomaly detection algorithm for streaming data. A fast and accurate density estimator, RS-Forest, has been proposed and then seamlessly extended to the data stream scenario. The proposed algorithm meets the key expectations set by ever-evolving data streams on anomaly detectors: (1) it is a one-pass algorithm, and has constant space complexity and linear time complexity; (2) it works well with drifting concepts, as a highly diverse ensemble operating in the statistically estimated feature space with high probability guarantee; (3) it has a super-fast response time due to the employed dual node profile technique; and (4) it only needs normal instances for training, and performs anomaly detection and model update in a timely and coherent manner. Furthermore, we conduct rigorous analysis, laying the solid theoretical foundation for the proposed algorithm. Experimental evaluations on multiple benchmark datasets demonstrated that, compared to the existing state-of-the-art methods, streaming RS-Forest can achieve higher AUC score, less runtime and is robust to different parameters. As part of future work, we are looking to (1) further explore the efficacy of RS-Forest as a non-parametric density estimator, (2) adapt streaming RS-Forest to the mixed feature space with missing values, and (3) reduce its space occupation while maintaining high detection rate.

Acknowledgment

This work is supported by a US Dept. of Army grant (W911NF-12-1-0066), an NIH grant (NIMHD RCMI 2G12MD007595) and a seed grant from the Louisiana Cancer Research Consortium (LCRC).

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[2] K. Zhang and W. Fan, "Forecasting skewed biased stochastic ozone days: Analyses, solutions and beyond," *Knowl. Inf. Syst*, vol. 14, no. 3, pp. 299–326, 2008.

[3] P. Ram and A. G. Gray, "Density estimation trees," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 627–635, 2011.

[4] M.-H. Chen, Q.-M. Shao, and J. G. Ibrahim, *Monte Carlo Methods in Bayesian Computation*. New York: Springer, 2000.

[5] K. Steliga and D. Szynal, "On markov-type inequalities," *International Journal of Pure and Applied Mathematics*, vol. 58, no. 2, pp. 137–152, 2010.

[6] D. M. Rocke and D. L. Woodruff, "Identification of outliers in multivariate data," *Journal of the American Statistical Association*, vol. 91, pp. 1047–1061, 1996.

[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. of the 8th IEEE International Conference on Data Mining*, pp. 413–422, IEEE Computer Society, 2008.

[8] K. Wu, A. Edwards, W. Fan, J. Gao, and K. Zhang, "Classifying imbalanced data streams via dynamic feature group weighting with importance sampling," in *SIAM International Conference on Data Mining*, pp. 722–730, 2014.

[9] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 1511–1516, 2011.

[10] K. Bache and M. Lichman, "UCI machine learning repository," 2013.

[11] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2283–2301, 2013.

[12] A. Ghoting, M. E. Otey, and S. Parthasarathy, "LOADED: link-based outlier and anomaly detection in evolving data sets," in *Proc. of the 4th IEEE International Conference on Data Mining*, pp. 387–390, 2004.

[13] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD '00*, pp. 71–80, 2000.

[14] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 139–148, 2009.

[15] P. J. Rousseeuw and K. V. Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.

[16] N. Abe, B. Zadrozny, and J. Langford, "Outlier detection by active learning," in *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 504–509, 2006.

[17] K. Sequeira and M. Zaki, "ADMIT: anomaly-based data mining for intrusions," in *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 386–395, ACM, 2002.

[18] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proc. of the 24rd International Conference on Very Large Data Bases*, pp. 392–403, 1998.

[19] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, 2000.

[20] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *CIKM '07*, pp. 811–820, 2007.

[21] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *EDBT '09*, pp. 529–540, 2009.

[22] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *ICDE '11*, pp. 135–146, 2011.

[23] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *PODS '02*, pp. 1–16, 2002.

[24] H. Liu, M. Xu, H. Gu, A. Gupta, J. D. Lafferty, and L. A. Wasserman, "Forest density estimation," *Journal of Machine Learning Research*, vol. 12, pp. 907–951, 2011.