Fast Anomaly Detection for Streaming Data*

Swee Chuan Tan
SIM University, Singapore
jamestansc@unisim.edu.sg

Kai Ming Ting and **Tony Fei Liu**
Monash University, Australia
{kaiming.ting, tony.liu}@monash.edu

Abstract

This paper introduces Streaming Half-Space-Trees (HS-Trees), a fast one-class anomaly detector for evolving data streams. It requires only normal data for training and works well when anomalous data are rare. The model features an ensemble of *random HS-Trees*, and the tree structure is constructed without any data. This makes the method highly efficient because it requires no model restructuring when adapting to evolving data streams. Our analysis shows that Streaming HS-Trees has *constant* amortised time complexity and constant memory requirement. When compared with a state-of-the-art method, our method performs favourably in terms of detection accuracy and runtime performance. Our experimental results also show that the detection performance of Streaming HS-Trees is not sensitive to its parameter settings.

1 Introduction

The problem of detecting anomalies in streaming data has the following characteristics. Firstly, the stream is infinite, so any off-line learning algorithms that attempt to store the entire stream for analysis will run out of memory space. Secondly, the stream contains mostly normal instances because anomalous data are rare and may not be available for training. In this case, any multi-class classifiers that require fully labeled data will not be suitable. Thirdly, streaming data often evolve over time. Thus, the model must adapt to different parts of the stream in order to maintain high detection accuracy.

This paper proposes an anomaly detection algorithm, Streaming Half-Space Trees (HS-Trees), that addresses the above-mentioned problem. The proposed method has several features that distinguish itself from other existing techniques. Firstly, it processes data in one pass and only requires constant amount of memory to process potentially endless streaming data or massive datasets. Thus it is different from existing off-line anomaly detectors (e.g., ORCA [Bay and Schwabacher, 2003], LOF [Breunig *et al.*, 2000] and SVM [Scholkopf *et al.*, 2002]) which are designed to mine static and finite datasets.

Secondly, Streaming HS-Trees is a one-class anomaly detector which is useful when a stream contains a significant amount of normal data.

Thirdly, it performs fast model updates in order to maintain high detection accuracy when dealing with time-varying data distribution. Its model update is simple and fast because it requires no modifications of the tree structure when processing streaming data.

Streaming HS-Trees employs *mass* [Ting *et al.*, 2010] as a measure to rank anomalies. The mass profile can be constructed with small samples, allowing the anomaly detector to learn quickly and adapt to changes in data streams in a timely manner.

Unlike other decision trees (e.g., random forests [Breiman, 2001]), Streaming HS-Trees does not induce its tree structure from actual training examples. Instead, the tree structure is constructed using the data space dimensions alone. The trees can be built quickly because it requires no attribute or split-point evaluations; and the model can be deployed before the streaming data arrive. A direct consequence of this feature is that Streaming HS-Trees has a *constant* amortised time complexity and a constant memory requirement. This is unlike algorithms that induce decision tree and alter its tree structure dynamically as streaming data arrive (e.g., Hoeffding Tree [Domingos and Hulten, 2000]).

Our experimental study shows that an ensemble of Streaming HS-Trees leads to a robust and accurate anomaly detector that is not too sensitive to different parameter settings.

2 Related work

In the literature, there are already a number of studies devoted to anomaly detection in *static* datasets. Typical examples include the statistical methods [Barnett and Lewis, 1994], classification-based methods [Abe *et al.*, 2006], clustering-based methods [He *et al.*, 2003], distance-based methods [Bay and Schwabacher, 2003], One-Class Support Vector Machine (SVM) [Scholkopf *et al.*, 2002] and Isolation Forest [Liu *et al.*, 2008]. These off-line learning methods are not designed to process streaming data because they require loading of the entire dataset into the main memory for mining.

*This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Recent work on anomaly detection for streaming data include the domain of monitoring sensor networks [Subramaniam *et al.*, 2006] and for abnormal event detection [Davy *et al.*, 2005], but there is currently little work considering anomaly detection in *evolving* data streams.

One interesting related work is LOADED by Otey *et al.* [2006], a link-based unsupervised anomaly detector that works well on datasets with mixed (continuous and categorical) attributes. However, LOADED does not work well on datasets with purely continuous attributes [Otey *et al.*, 2006]. Unlike LOADED, Streaming HS-Trees is a semi-supervised one-class learner [Chandola *et al.*, 2009] that works well for data with continuous attributes.

A recent system that deals with non-stationary data distributions is OLINDDA (OnLIne Novelty and Drift Detection Algorithm) [Spinosa *et al.*, 2009]. OLINDDA uses standard clustering algorithm to groups examples into clusters (or concepts). Through monitoring the clusters, it detects new emerging concepts rather than anomalies.

Apart from unsupervised methods discussed earlier, supervised learning methods can also be used for anomaly detection in data streams. For example, Hoeffding Trees (HT) [Domingos and Hulten, 2000; Hulten *et al.*, 2001] is an incremental anytime decision tree induction algorithm for classifying high-speed data streams. HT can also be used with Online Coordinate Boosting (denoted as BoostHT) [Pelossof *et al.*, 2009]. HT and BoostHT require positive as well as negative class labels to be available for training. However, this is not a realistic assumption because anomalous data is usually rare or not available for training.

3 The Proposed Method

This section presents the proposed method and the key notations used to describe the method are listed in Table 1.

x	a streaming point
n	the number of streaming points
T	an Half Space Tree, HS-tree
$Node$	a node in an HS-Tree
k	the current depth of a node or $Node.k$
t	the number of HS-Trees in an ensemble
h	maximum depth (level) of a tree, or $maxDepth$
r	mass of a node in the reference window
l	mass of a node in the latest window
ψ	window size
s	an anomaly score

Table 1: Key notations used in this paper.

3.1 Overview

The proposed method is an ensemble of HS-Trees. Each HS-Tree consists of a set of nodes, where each node captures the number of data items (a.k.a. mass) within a particular subspace of the data stream. Mass is used to profile the degree of anomaly because it is simple and fast to compute in comparison to distance-based or density-based methods.

To facilitate learning of mass profiles in evolving data streams, the algorithm segments the stream into windows of equal size (where each window contains a fixed number of data items). The system operates with two consecutive windows, the *reference* window, followed by the *latest* window. During the initial stage of the anomaly detection process, the algorithm learns the mass profile of data in the reference window. Then, the learned profile is used to infer the anomaly scores of new data subsequently arriving in the latest window—new data that fall in high-mass subspaces is construed as normal, whereas data in low-mass or empty subspaces is interpreted as anomalous. As new data arrive at the latest window, the new mass profile is also recorded. When the latest window is full, the newly recorded profile is used to override the old profile in the reference window; thus the reference window will always store the latest profile that can be used to score the next batch of newly arriving data. Once this is done, the latest window erases its stored profile and get ready to capture profile of the next batch of newly arriving data. This process continues as long as the stream exists.

3.2 Half-Space Trees

Definition An HS-Tree of depth h is a full binary tree consisting of $2^{h+1} - 1$ nodes, in which all leaves are at the same depth, h.

When constructing a tree, the algorithm expands each node by picking a randomly selected dimension, q, in the *work space* (to be described later in this section) associated with the node. Using the mid-point of q, the algorithm bisects the work space into two half-spaces, thus creating the left child and right child of the node. Node expansion continues until the maximum depth (i.e., h or *maxDepth*) of all nodes is reached.

Each node records the mass profile of data in a work space that it represents, and has the following elements: (i) arrays min and max, which respectively store the minimum and maximum values of each dimension of the work space represented by the node; (ii) variables r and l, which record the mass profiles of data stream captured in the reference window and latest window, respectively; (iii) variable k, which records the depth of the current node; and (iv) two nodes representing the left child and right child of the current node, each associated with a half-space after the split. Figure 1 depicts an example window of (two-dimensional) data that is partitioned by a simple HS-Tree.

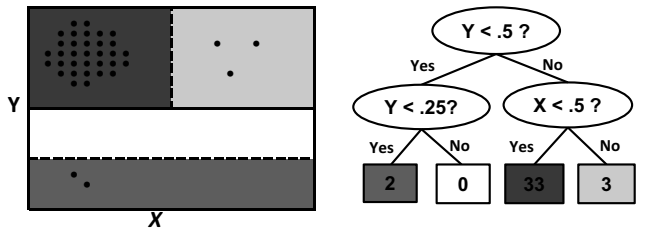


Figure 1: An example of data (in a window) partitioned by a simple HS-Tree.

Creating diverse HS-Trees is crucial to the success of our ensemble method. This is achieved by using a procedure,

Initialise Work Space, right before the construction of *each tree*. Assume that attributes' ranges are normalised to [0, 1] at the outset. Let s_q be a real number *randomly and uniformly* generated from the interval [0, 1]. A work range, $s_q \pm 2 \cdot max(s_q, 1 - s_q)$, is defined for every dimension q in the feature space D. This produces a work space that is a random perturbation of the original feature space. Since each HS-Tree is built from a different work space (defined as min and max in Algorithm 1), the result is an ensemble of diverse HS-Trees.

Algorithm 1 : BuildSingleHS-Tree(min, max, k)

Inputs: min & max - arrays of minimum and maximum values for every dimension in a Work Space,
k - current depth level
Output: an HS-Tree

1: **if** $k == maxDepth$ **then**
2: return *Node*($r \leftarrow 0, l \leftarrow 0$) {External node}
3: **else**
4: randomly select a dimension q
5: $p \leftarrow (max_q + min_q)/2$
6: {Build two nodes ($Left$ & $Right$) from a split into two equal-volume half-spaces.}
7: $temp \leftarrow max_q$; $max_q \leftarrow p$
8: $Left \leftarrow$ BuildSingleHS-Tree($min, max, k + 1$)
9: $max_q \leftarrow temp$; $min_q \leftarrow p$
10: $Right \leftarrow$ BuildSingleHS-Tree($min, max, k + 1$)
11: return *Node*($Left, Right, SplitAtt \leftarrow q$, $SplitValue \leftarrow p, r \leftarrow 0, l \leftarrow 0$)
12: **end if**

Algorithm 1 shows the procedure for building a single HS-Tree. Each internal node is formed by randomly selecting a dimension q (Line 4) to form two half-spaces; the split point is the mid-point of the current range of q. The mass variables of each node, r and l, are initialised to zero during the tree construction process.

Recording mass profile in HS-Trees. Once HS-Trees are constructed, mass profile of normal data must be recorded in the trees before they can be employed for anomaly detection. The process involves traversing every instance in a window through each HS-Tree. Algorithm 2 shows the process where instances in the reference window will update mass r; and mass l is updated using instances in the latest window. These two collections of mass values at each node, r and l, represent the data profiles in the two different windows. They are used in Streaming HS-Trees, which will be described in Section 3.3.

Algorithm 2 : UpdateMass($x, Node$, *referenceWindow*)

Inputs: x - an instance, $Node$ - a node in an HS-Tree
Output: none

1: (*referenceWindow*)? $Node.r$++ : $Node.l$++
2: **if** ($Node.k < maxDepth$) **then**
3: Let $Node'$ be the next level of $Node$ that x traverses
4: UpdateMass($x, Node'$, *referenceWindow*)
5: **end if**

Anomaly score. Mass in every partition of an HS-Tree is used to profile the characteristics of data. Let $m[i]$ be the mass in a half-space partition at depth level i of an HS-Tree. Under uniform mass distribution, mass values between any two partitions at levels i and j are related as follows: $m[i] \times 2^i = m[j] \times 2^j$. When the distribution is non-uniform, the following inequality establishes an ordering between partitions at different levels: $m[i] \times 2^i < m[j] \times 2^j$. We use this property to rank anomalies.

Let $Score(x, T)$ be a function that traverses a test instance x from the root of an HS-Tree (T) until a *terminal node*. This function then returns the anomaly score of x by evaluating $Node^*.r \times 2^{Node^*.k}$, where $Node^*.k$ being the depth level of the terminal node containing $Node^*.r$ instances. Here, a terminal node, or $Node^*$, is a node that has reached the maximum depth, or a node that contains $sizeLimit$ instances or fewer.

The final score for x is the sum of scores obtained from each HS-Tree in the ensemble:

$$\sum_{T \in HS\text{-}Trees} Score(x, T).$$

In practice, $sizeLimit$ is not a critical parameter, and a good default setting is 0.1ψ, where ψ is the window size. We want a large value for $maxDepth$ so that a large number of subspaces is used to capture the data profile in a comprehensive manner. But in practice, this setting is limited by the amount of computer memory available for tree construction. In our computer, we set $maxDepth$ to 15, which is adequate for capturing data stream profile. Streaming HS-Trees is able to learn data stream profile using small samples; hence, a small window size of $\psi = 250$ is sufficient for our experiments. The ensemble uses 25 trees as this is a moderate ensemble size (t) which can be easily incorporated in most machines.

3.3 Streaming HS-Trees

Algorithm 3 shows the operational procedure for Streaming HS-Trees. Line 1 builds an ensemble of Half-Space Trees. Line 2 uses the first ψ instances of the stream to record its initial reference mass profile in the HS-Trees. Since these instances come from the initial reference window, only mass r of each traversed node is updated. After these two steps, the model is ready to provide an anomaly score for each subsequent streaming point.

Mass r is used to compute the anomaly score for each streaming point (Line 8). The recording of mass for each subsequent streaming point in the latest window is then carried out on mass l (Line 9). At the end of each window, the model is updated. The model update procedure is simple—before the start of the next window, the model is updated to the latest mass by simply transferring the non-zero mass l to r (Line 14). This process is fast because it involves no structural change of the model. After this, each node with a non-zero mass l is reset to zero (Line 15).

Time and Space Complexities: The four key operations in the main loop of Algorithm 3 are: scoring (Line 8), updating mass (Line 9), model update (Line 14) and model resets (Line 15). For each of the first two operations, every

Algorithm 3 : Streaming HS-Trees(ψ, t)

Inputs: ψ - Window Size, t - number of HS-Trees
Output: s - anomaly score for each streaming instance x

```
Build t HS-Trees : Initialise Work Space and call Algorithm 1 for each tree
Record the first reference mass profile in HS-Trees: for each tree T, invoke UpdateMass(x, T.root, true) for each item x in the first ψ instances of the stream
Count ← 0
while data stream continues do
    Receive the next streaming point x
    s ← 0
    for each tree T in HS-Trees do
        s ← s + Score(x, T) {accumulate scores}
        UpdateMass(x, T.root, false) {update mass l in T}
    end for
    Report s as the anomaly score for x
    Count++
    if Count == ψ then
        Update model : Node.r ← Node.l for every node with non-zero mass r or l
        Reset Node.l ← 0 for every node with non-zero mass l
        Count ← 0
    end if
end while
```

instance is traversed from a tree's root to a terminating node (i.e., $O(h)$); the last two operations each accesses at most ψ nodes but occurs $\frac{n}{\psi}$ times over the entire stream. Hence the **(average-case) amortised time complexity** for n streaming points is $O(t(h+1))$; the worst-case is $O(t(h+\psi))$, which occurs when model update and reset are performed between streaming data. These time complexities are constant when the maximum depth level (h), ensemble size (t) and the window size (ψ) are fixed.

In Streaming HS-Trees, each arriving instance is first processed and then discarded, before the next is processed. This forms a one-pass algorithm that uses a finite memory to process infinite data streams. The **space complexity** for HS-Trees is $O(t2^h)$ which is also a constant with fixed t and h.

4 Experimental Setup

Data: Columns 2 to 4 of Table 2 summarise the six large datasets used in this study. SMTP and HTTP (from KDD Cup 99) are streaming data involving network intrusions. HTTP is characterised by sudden surges of anomalies in some streaming segments. SMTP does not have surges of anomalies, but possibly exhibits some distribution changes within the streaming sequence.

In practice, it is hard to quantify whether a distribution change has indeed occurred within a stream. For this reason, we derive a dataset, SMTP+HTTP, containing the SMTP data instances follow by the HTTP data instances. We expect a distribution change to occur when the communication protocol is switched from SMTP to HTTP.

COVERTYPE is a UCI dataset [Asuncion and Newman, 2007] commonly used in data stream research. We split the anomaly class into several small groups and placed them in different segments of the dataset in order to simulate short bursts of anomalies in different streaming segments.

SHUTTLE (from UCI) and MULCROSS [Rocke and Woodruff, 1996] are datasets with little or no distribution change. However, MULCROSS contains dense clusters of anomalies that are harder to detect than scattered anomalies.

Experimental Settings: The parameter settings for Streaming HS-Trees have been discussed earlier in Section 3. In addition, all the methods are implemented in Java and all experiments were conducted on a 3GHz Pentium CPU with 1GB RAM.

Once the anomaly scores for all instances (of a segment in the data stream or of the entire dataset) were obtained, the instances were ranked based on their anomaly scores. From this ranking and the ground truth, we then computed the AUC (Area Under receiver operating characteristic Curve) [Hand and Till, 2001] to measure the performance of all anomaly detectors reported in this paper.

In all experiments, we conducted 30 independent runs of each algorithm on each dataset, and then computed the average results. A t-test at 5% level of significance was used to compare performance levels of the algorithms.

5 Experimental Results

We report the results of the experiments in this section. First, we assess the effectiveness of model adaptation to varying data distribution. This is done by comparing Streaming HS-Trees that performs regular updates of its model (denote this as HSTa), versus Streaming HS-Trees without model update (we denote this model as HSTn, which only learns from the first ψ instances of the stream).

Model Adaptation Performance: Unlike HSTa, Table 2 (Columns 5 and 6) shows that HSTn only works well in datasets with no change in distribution (i.e., SHUTTLE and MULCROSS). It performs poorly when a distribution change occurs within the data (e.g., SMTP, SMTP+HTTP, and COVERTYPE). These results are consistent with Figure 2, where HSTn degrades when there are changes in certain streaming segments of SMTP, SMTP+HTTP, and COVERTYPE. Using two artificial datasets, we also confirm that HSTn does not work well when there is a drift in normal data, whereas HSTa works well when there is a drift in either anomalies or normal data. Details of the artificial data are omitted here due to space constraints.

Comparison with Hoeffding Trees: Here, we compare HSTa with Hoeffding Trees (HT) as well as HT with Online Coordinate Boosting (BoostHT). For HT and BoostHT, we use the probability of predicting a negative class (i.e., a normal point) as the anomaly score—a true anomaly generally gets a low prediction probability, while a normal point gets a high probability. This serves as a ranking measure for anomaly detection. We employ the Java implementations of HT and BoostHT developed by Bifet *et al.* [2009].

In terms of the overall AUC scores (c.f. Columns 6 to 8 of Table 2), we expect Hoeffding Tree (HT) to produce the

Dataset	Data Size	Dimensionality	Anomaly	AUC				Runtime	
				HSTn	HSTa	HT	BoostHT	HSTa	HT
HTTP	567497	3	attack (0.4%)	.982	.996	.994	**.998**	48	227
SMTP	95156	3	attack (0.03%)	.740	.875	.858	.692	10	39
SMTP+HTTP	662653	3	attack (0.35%)	.387	.996	.991	.993	57	272
COVERTYPE	286048	10	outlier (0.9%)	.854	.991	.998	.968	25	124
MULCROSS	262144	4	2 dense clusters (10%)	.998	.998	**1.00**	**1.00**	26	114
SHUTTLE	49097	9	class 2,3,5-7 (7%)	.999	.999	.991	.984	6	21

Table 2: Average AUC scores for Streaming HS-Trees with no model updates (HSTn) and with regular model updates (HSTa), Hoeffding Tree (HT), and HT with boosting (BoostHT). Using HSTa as a reference, scores lower than HSTa are underlined, and scores higher than HSTa are printed in boldface. Runtime is measured in seconds. The figures in brackets are the proportions of anomalies.

best overall results because it is an oracle-informed *multi-class classifier*—it is given the advantage of using the actual positive and negative class labels for training, and this is done *immediately* after each new instance is scored. In contrast, Streaming HS-Trees with regular model updates (HSTa) uses only normal data for training.

Because HT is an optimistic baseline, any one-class anomaly detector for evolving data streams that performs comparably to HT shall be deemed as a competitive method. Interestingly, Table 2 shows that HSTa actually gives higher AUC scores on four (i.e., HTTP, SMTP, SMTP+HTTP, and SHUTTLE) out of six datasets tested, as compared to HT. This observation is further examined in Figure 2, which shows that HSTa surprisingly outperforms HT in HTTP (segment 2), SMTP (segments 1 and 5), SMTP+HTTP (segment 1) and SHUTTLE (segment 1). HT is unable to cope with distribution changes in these segments, causing its detection performance to degrade.

Table 2 shows that BoostHT does not improve the performance of HT significantly. In fact, BoostHT performs poorly on SMTP, which is the most imbalanced dataset used in this study. This could be due to overfitting of the boosted model.

We also stress test the methods using datasets in which only 20% of the data are labelled. We find that HSTa outperforms HT on smaller datasets (namely SHUTTLE and SMTP) due to its ability to learn with fewer instances. Details of this experiment will be given in a future publication.

Runtime Performance: Hoeffding Tree has the ability to adapt its tree structure to streaming data—it uses Hoeffding Bound to decide the best splitting attribute when incrementally inducing a decision tree from a data stream. However, this flexibility comes with a price—the need to modify the tree causes HT's runtime to be four to six times slower than Streaming HS-Trees, as shown in the last two columns of Table 2.

Unlike Hoeffding Trees, Streaming HS-Trees does not modify or extend the tree structure during the streaming process, after it was first built. Even the tree construction procedure for Streaming HS-Trees is very efficient because the process requires no evaluation criteria for dimension or split point selections.

Effects of Parameters: Using four of the datasets for illustration, Figure 3(a) shows that the AUC scores of HSTa reach high values with 10 trees, and the scores improve steadily as the ensemble size increases. Using the SHUTTLE dataset as an example, Figures 3 (b) to (d) show that the effects of different parameter settings diminish as the ensemble size grows. This is due to the power of ensemble learning—while individual base learners (i.e., HS-Trees) may be weakened by non-optimal parameter settings for a problem at hand, the combination of these weak learners still produces reasonably good results.

6 Concluding Remarks

The proposed anomaly detection algorithm, Streaming HS-Trees, satisfies the key requirements for mining evolving data streams: (i) it is a one-pass algorithm with $O(1)$ amortised time complexity and $O(1)$ space complexity, which is capable of processing infinite data streams; (ii) it performs anomaly detection and stream adaptation in a seamless manner.

Our empirical studies show that Streaming HS-Trees with the regular model update scheme is robust in evolving data streams. In terms of detection accuracy, Streaming HS-Trees is comparable to the oracle-informed Hoeffding Tree (an optimistic baseline). In terms of runtime, Streaming HS-Trees outperforms Hoeffding Tree. Our results also show that the performance of Streaming HS-Trees is robust against different parameter settings.

References

N. Abe, B. Zadrozny, and J. Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD*, 2006.

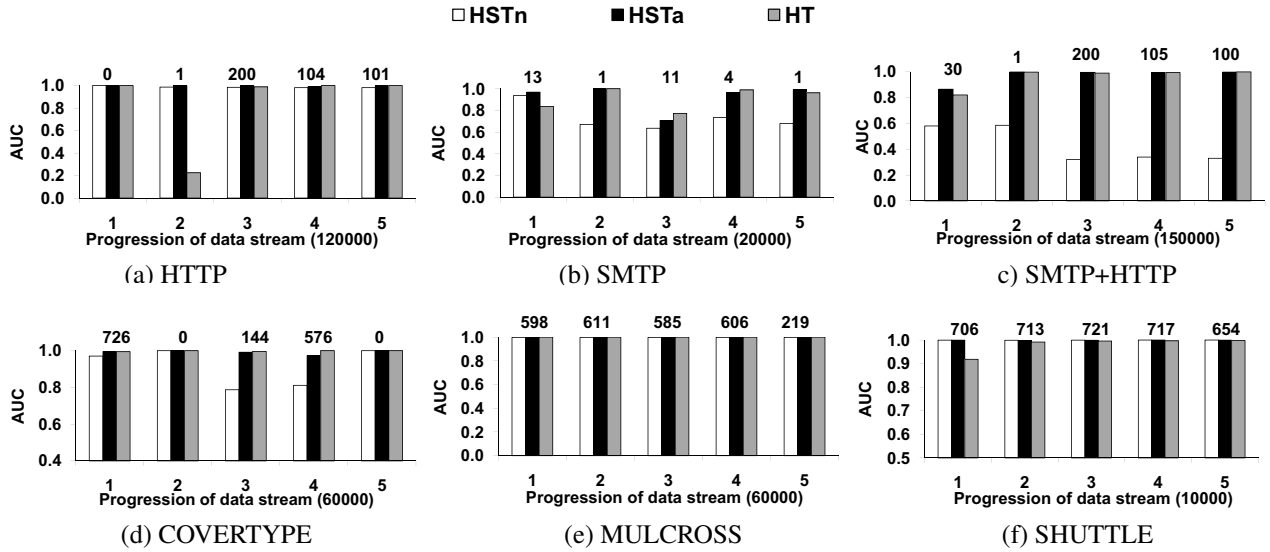
A. Asuncion and D. Newman. Uci machine learning repository. 2007.

V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley, 1994.

S.D. Bay and M. Schwabacher. Mining distance-based anomalies in near linear time with randomization and a simple pruning rule. In *Proceedings of the 9th ACM SIGKDD*, 2003.

A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD*, 2009.

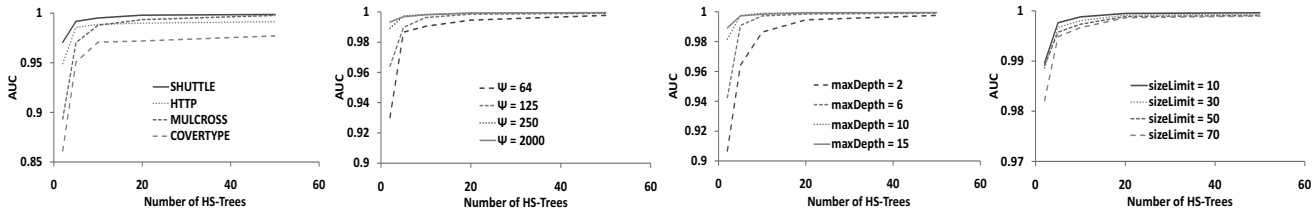
L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.



(a) HTTP (b) SMTP c) SMTP+HTTP

(d) COVERTYPE (e) MULCROSS (f) SHUTTLE

Figure 2: AUC scores in five segments. The number on top of each segment is the number of anomalies in that segment. The number in bracket is the number of items in each segment.



(a) Effect of ensemble size (t) (b) Effect of window size (ψ) (c) Effect of $maxDepth$ (d) Effect of $s0izeLimit$

Figure 3: As the ensemble size grows, the effects of different parameter settings on the detection performance tend to diminish.

M.M. Breunig, H-P. Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 6th ACM SIGKDD*, 2000.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:1–58, 2009.

M. Davy, F. Desobry, A. Gretton, and C. Doncarli. An online support vector machine for abnormal events detection. *Signal Processing*, 86:2009–2025, 2005.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD*, 2000.

D.J. Hand and R.J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45:171–186, 2001.

Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recognition Letter*, 24:1641–1650, 2003.

G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD*, 2001.

F.T. Liu, K.M. Ting, and Z.H. Zhou. Isolation forests. In *Proceedings of ICDM 2008*, 2008.

M.E. Otey, A. Ghoting, and S. Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. *Data Mining and Knowledge Discovery*, 12:203–228, 2006.

R. Pelossof, M. Jones, Vovsha, and C. I. Rudin. Online coordinate boosting. In *Proceedings of the 3rd IEEE On-line Learning for Computer Vision Workshop*, 2009.

D.M. Rocke and D.L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91:1047–1061, 1996.

B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. *Advances in Neural Information Processing Systems*, 12:582–588, 2002.

E.J. Spinosa, A.P. Leon, F. Carvalho, and J. Gama. Novelty detection with application to data streams. *Intelligent Data Analysis*, 13:405–422, 2009.

S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of The 32nd VLDB*, 2006.

K.M. Ting, G.T. Zhou, F.T. Liu, and S.C. Tan. Mass estimation and its applications. In *Proceedings of 16th ACM SIGKDD*, 2010.