Anomaly Detection for Network Monitoring: A Telstra EDN DNS Case Study

by

Muhammad Aditya Hilmy

Student ID: 1262771

School of Computing and Information Systems
Faculty of Engineering and Information Technology
THE UNIVERSITY OF MELBOURNE

June 2024

Abstract

Telstra operates a large number of heterogeneous devices, which can fail at any time. They employ redundancy to improve reliability, but this also results in small failures being unnoticed. If left unchecked, such failures can cascade into an outage. Anomaly detection can be used to detect abnormal behaviours that can indicate the presence of faults. In this work, we have established the desired properties of an anomaly detector specific to the use case of network monitoring. In line with those desired properties, we chose to evaluate STReamRHF, which is an anomaly detection method based on random forest. Additionally, we proposed some modifications to the algorithm to further fit the desired properties. We evaluated the proposed anomaly detector qualitatively using synthetic and real-world data collected from Telstra EDN DNS system. We also present the results using Numenta Anomaly Benchmark (NAB). We concluded that our proposed technique is not yet optimal for network monitoring and discuss our findings and propose future research direction for Telstra to implement anomaly detection in the EDN DNS system.

Declaration of Authorship

I certify that:

- This thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- Where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the School.
- The thesis is 8333 words in length (excluding text in images, table, bibliographies and appendices).

Signed: _____

Date: _____

Acknowledgements

We thank Christopher Leckie (University of Melbourne) for inspiration in devising the normalisation step of our approach. We thank Jeremy Laidman (Telstra) for helping us set up data collection for our qualitative evaluation. We thank Christopher Leckie (University of Melbourne), Christine Rizkallah (University of Melbourne), Anthony Balia (Telstra), Jeremy Laidman (Telstra), Krupal Patel (Telstra), and Michael Eales (Telstra) for their feedback and guidance throughout the Research Project. We also thank Bruce Kempe (Telstra) for hosting our visit to the Telstra Global Operations Centre early in the project.

Contents

Abstract i

Declaration of Authorship ii

Acknowledgements iii

List of Figures vi

List of Tables vii

1 Introduction 1

2 Literature Review 4
- 2.1 Desired properties of anomaly detectors . . . 4
- 2.2 Anomaly detection in network monitoring . . . 6
- 2.3 STReamRHF anomaly detection algorithm . . . 8
  - 2.3.1 Random Histogram Forest . . . 8
  - 2.3.2 STReamRHF . . . 10
- 2.4 Handling seasonality in anomaly detection . . . 11
  - 2.4.1 Summary . . . 12

3 Our Approach 13
- 3.1 Definition of anomaly . . . 13
- 3.2 Proposed anomaly detector . . . 14
  - 3.2.1 Encoding time series using shingling . . . 15
  - 3.2.2 Modification to the STReamRHF insertion algorithm . . . 15
  - 3.2.3 Normalisation of anomaly score . . . 17
  - 3.2.4 Determining anomaly labels . . . 18

4 Results and Discussion 20
- 4.1 Experimental evaluation . . . 20
  - 4.1.1 Synthetic dataset . . . 20
  - 4.1.2 Real-world data: Telstra EDN DNS monitoring . . . 23
  - 4.1.3 Benchmarking with Numenta Anomaly Benchmark . . . 26
- 4.2 Discussions . . . 28
  - 4.2.1 Univariate anomaly detection . . . 28

4.2.2 Multivariate anomaly detection 30
4.2.3 Summary 31

5 Conclusion and Future Work **32**
5.1 Conclusion 32
5.2 Future work 33

Bibliography **34**

List of Figures

3.1 Plot showing a synthetic time series and anomaly scores of the original and modified STReamRHF algorithm 16
3.2 Numer of NXDOMAIN responses with four unusually high spikes 19

4.1 Anomaly scores for STReamRHF with and without shingling on synthetic data with point anomalies 22
4.2 Anomaly scores for STReamRHF with and without shingling on synthetic data with seasonal anomalies 23
4.3 Anomaly scores for STReamRHF with and without shingling on synthetic data with correlation anomalies 24
4.4 Number of NXDOMAIN response with anomaly scores 25
4.5 Number of running processes in server A 26
4.6 Number of running processes in server B 26
4.7 Number of UDP packets in and out 27
4.8 Anomaly scores of art_daily_jumpsdown.csv with shingle size of 5 and 500 29

List of Tables

2.1 Comparison of anomaly detection methods 8

4.1 Numenta Anomaly Benchmark results 27

Chapter 1

Introduction

Telstra operates a large number of heterogeneous network devices and servers. With so many interconnected devices, failure is bound to happen. To prevent disruption, the company implements redundancy to keep the systems running in case of failures. As a result of this redundancy, small-scale failures will have almost no impact on performance.

However, it means that those failures can go undetected. If unresolved, such failures can manifest over time, degrading performance slowly until it reaches a critical point, and outages can happen. An example where this has happened is an outage at Discord[1], where unnoticed issues caused a cascading failure resulting in an outage.

To observe how Telstra monitors the system to prevent such outage and propose improvements, this study will focus on a service used internally at Telstra called the Enterprise Data Network Domain Name System (EDN DNS). Essentially, it is a DNS service that resolves internal hostnames, primarily used by devices in the Telstra corporate network.

Currently, the EDN DNS team monitors various performance metrics and will escalate an issue if they notice potential failures. With plenty of devices to monitor and high volume of metrics to process, escalating alerts manually has proven to be difficult in forecasting potential failures.

Telstra has attempted to automate this alerting by thresholding: an alert will be raised if the metric value reaches a certain lower or upper limit. Deciding this threshold,

[1] Unavailable Guilds & Connection Issues - https://discordstatus.com/incidents/qk9cdgnqnhcn

however, is not straightforward. Setting a static threshold will not work well because the values fluctuate over time, unless the threshold is set reasonably high or low. For instance, traffic volume will differ during working hours and midnight, or holidays and weekdays. Furthermore, the defined threshold may not reflect the devices' optimal operating parameters. As a consequence, false negatives or positives can occur.

To forecast potential problems, we can use past data to capture the patterns of normal behaviour, and raise an alert when the current behaviour deviates from the past. The process of finding abnormal patterns that do not fit into normal behaviour is known as anomaly detection [1].

The EDN DNS team is considering employing an anomaly detection method based on the Holt-Winters forecasting method devised by Brutlag [2]. While the method is suitable for network monitoring due to its small memory footprint and fast computation, it is sensitive to the choice of parameters, which is difficult to set beforehand [2].

Furthermore, the method cannot correlate multiple time series as it works on a single time series (i.e., univariate). A multivariate anomaly detector is more preferable since in an interconnected network of devices, an anomaly in one part of the system could indicate an anomaly in another part. By considering multiple variables, the anomaly detector could potentially detect potential faults earlier.

To that end, this paper aims to evaluate the use of STReamRHF algorithm for network monitoring purposes, specifically in the context of Telstra EDN DNS monitoring. STReamRHF is a multivariate anomaly detection algorithm based on a variation of random forest called Random Histogram Forest [3]. We found the properties of STReamRHF to align with the desired properties of anomaly detector for network monitoring, which will be discussed later.

We highlight our key contributions as follows.

1. We have established a set of desired properties that an anomaly detector for network monitoring should have.
2. We have identified a limitation in STReamRHF that makes it unable to detect extreme point anomalies, and we have proposed modification to address that.
3. We have proposed and evaluated the use of STReamRHF with the *shingling* method to detect anomalies in a streaming time series. The evaluation suggests

that our proposed method worked well in detecting point anomalies, but not optimal to detect other types of anomalies.

The rest of the paper is organised as follows. First, we will establish the requirements of anomaly detectors for network monitoring. Second, we will review related works in the literature, including anomaly detection techniques and seasonality detection in time series. Third, we will describe our approach in detecting the anomalies, which includes a descrption of our modification to the STReamRHF algorithm. Finally, we will evaluate the proposed algorithm against the requirements, discuss the results, and conclude the paper.

Chapter 2

Literature Review

2.1 Desired properties of anomaly detectors

Lavin and Ahmad have defined the properties of an ideal anomaly detector, which are: (1) detects all anomalies in the streaming data (100% recall) with no false positives (100% precision), (2) detects anomalies as soon as possible, (3) works with real time data, i.e. anomalies are detected as new data points come, and (4) is fully automated without human intervention. Based on this definition, we derived a set of desired properties of anomaly detectors for network monitoring. We also define additional desired properties based on the specific challenges faced at Telstra EDN DNS monitoring team.

P1 - *Favours false negatives over false positives.* While ideally anomaly detectors should have 100% precision and recall, this is incredibly difficult to achieve. Oftentimes, precision and recall are contradictory with one another. For example, we can achieve 100% recall by always detecting anomalies regardless of what the actual data indicates. However, this would result in a very low precision. To that end, we assert that anomaly detectors should favour false negatives over false positives for the following reasons. First, alerts raised by anomaly detectors must be actioned by human operators who will take a closer look and perform corrective actions if necessary. Having too many false alerts may overwhelm the human operators, potentially causing them to miss actual anomalies. Second, network systems usually implement redundancy to improve reliability. In which case, occasional false negatives should be benign due to this redundancy. Therefore, we think it is better to have false negatives than false positives.

P2 - *Detects anomalies as new data points come.* With network systems being mission critical, anomalies must be detected as soon as the anomalous data point is processed. In other words, the anomaly detector should not need to look ahead to determine whether a data point is anomalous.

P3 - *Detects problems as soon as possible.* Lavin and Ahmad defines an ideal detector to be able to detect anomalies as early as possible [4]. However, in this paper, we aim to detect anomalies to uncover potential problems in the system. Therefore, we think that "detecting *problems* as early as possible" is more suitable than "detecting anomalies as early as possible."

P4 - *Requires no human intervention.* Network monitoring can involve plenty of metrics with different characteristics and nominal ranges. Requiring human operators to continuously monitor and fine tune the detector parameters is a tedious task. With limited human resources, like in the case of Telstra EDN DNS, doing such work may not be possible. Hence, the anomaly detector must be able to work without human intervention.

P5 - *Adaptable to concept drift.* Characteristics of the time series data can change over time due to legitimate reasons. For example, a software update may cause the average CPU usage to drop because of increased efficiency. The anomaly detector must be able to detect such change in patterns and adapt the parameters to accommodate the 'new normal'.

P6 - *Able to handle seasonality.* Some metrics at the Telstra EDN DNS system have a seasonality; meaning the data fluctuates in a regular pattern. For example, CPU load peaks during the day when users are working and flattens during the evening. To that end, the anomaly detector must be able to recognise such fluctuations detect anomalies when the data point deviates from what is expected at a particular time.

P7 - *Works with multiple variables.* In a heterogeneous network, metrics can be correlated to one another, and a slight deviation in the correlation might indicate a bigger problem. The following example demonstrates the importance of multivariate anomaly detection. Two of the metrics being monitored by the Telstra EDN DNS team are "number of inbound UDP packets" (UDP in) and "number of outbound UDP packets" (UDP out). In a normal behaviour, there is a positive correlation between the

two metrics, i.e. many inbound packets implies many outbound packets. Nevertheless, there were instances when the value of UDP in does not correlate with UDP out, but those metrics were still within the normal range. A closer look at the metrics uncovered a hidden fault in the system. A univariate anomaly detector might not be able to uncover this issue because the individual values were still within the normal range. However, a multivariate anomaly detector might infer that the correlation had been broken and raised an anomaly.

P8 - *Explainable.* When an alert is raised, human operators must assess the criticality and devise an appropriate response. However, due to the complexity of heterogeneous network, doing so could be a challenging task. An explainable anomaly detector could help uncover the cause of anomalies and provide explanation as to why a data point is anomalous [5]. This explanation would justify the criticality of the alert and provide hints on where to investigate.

2.2 Anomaly detection in network monitoring

Anomaly detection relies on performance metrics received from a variety of devices to capture past patterns in the metrics and detect when pattern deviates from the past behaviour [1]. Its ability to detect even the slightest deviations allows it to discover potential problems before it becomes too severe. This would help the operations team to resolve issues before they cause an incident.

Methods of time series anomaly detection are divided into three types: proximity-based, prediction-based, and reconstruction-based [6]. Proximity-based methods rely on a distance measure to compute similarity between data points, where data points far from the others are considered anomalies [6]. However, this method requires prior knowledge on the number and duration of anomalies [6], which are difficult to infer in a highly dynamic network traffic characteristic. In prediction-based methods, past time series data is used to fit a model, and the model is used to predict future data points [6]. When, at a particular time, the actual data is significantly different than the predicted data, it is considered anomalous. In reconstruction-based methods, a model is trained to capture the characteristics of patterns and used to reconstruct the original data [6]. Anomaly is detected when the reconstructed data is far different than the original data.

Several works have been done to explore the use of anomaly detection for network monitoring. Brutlag proposed an anomaly detection method based on exponential smoothing the Holt-Winters forecasting algorithm (HW) [2]. The algorithm models the time series as variables and predicts the time series based on those variables. An anomaly is detected when the difference between actual and predicted values *consistently* exceed a specified threshold. Through the use of a window length, HW is able to forget old data, and thus robust to concept drift. Additionally, by modeling the trend and seasonality of the time series, HW can handle seasonal time series.

Unfortunately, the HW algorithm relies on parameters that are difficult to optimize a priori [2]. This is confirmed by Ekberg et al. when they evaluated the HW algorithm for network monitoring [7]. While the authors argued that the method is usable for network monitoring, the performance is sensitive to the choice of parameters, implying that a wrong choice of threshold values can lead to false positives or false negatives [7].

Geiger et al. proposed the use of generative adversarial networks (GAN) called TadGAN to detect time series anomalies, which uses reconstruction error to calculate anomaly [6]. The authors highlighted that this method offers a robust anomaly score calculation and can reduce false positives. However, GAN is computationally expensive. Therefore, it is challenging to deploy GAN to detect anomalies in real time.

To satisfy the explainability requirement, we turn to white box algorithms, specifically tree-based methods. Robust Random Cut Forest (RRCF) is a streaming data anomaly detector built upon the ideas of Isolation Forest [8]. RRCF approximately preserves pairwise distances and, similar to Isolation Forest, attempts to group similar instances and isolate anomalous ones [8]. Based on experimental evaluations at [8], the algorithm seems to be capable of handling seasonality. However, Nesic et al. pointed out that RRCF suffers from scalability issues. Furthermore, RRCF is not capable of handling concept drift [3].

Random Histogram Forest (RHF) is a batch anomaly detection method, building on the idea of Random Forest [9] and Isolation Forest [10] [11]. It relies on an ensemble of weak probabilistic detectors [11]. It has only two parameters that are not sensitive to choice [11], namely forest size and maximum depth. The algorithm is also white-box, since tree is intrinsically explainable [3]. Furthermore, the algorithm's complexity is linear to

the input size [11]. Although, one downside of the algorithm is that it cannot handle streaming data.

To that end, Nesic et al. proposed a modification to RHF called STReamRHF [3]. STReamRHF builds an initial RHF and incrementally modifies the tree as new data points arrive. Interestingly, because RHF has a linear complexity, the worst-case complexity of STReamRHF is only dictated by its fixed parameters. Additionally, to accommodate for concept drift, the method employs a *reference window* to forget old data points [3]. Although, it is unable to handle seasonality in time series since it has no notion of chronological order. This is also proved by our experimental evaluation.

Table 2.1 shows the comparison of the various anomaly detection methods.

Table 2.1: Comparison of anomaly detection methods

Properties	HW [2]	TadGAN [6]	RRCF [8]	RHF [11]	StreamRHF [3]
Handle streaming data	Yes	No	Yes	No	Yes
Unsupervised	Yes	No	Yes	Yes	Yes
Sensitive to parameter choice	Yes	No	Minimal	Minimal	Minimal
Handle concept drift	Yes	Yes	No	-	Yes
Handle seasonality	Yes	Yes	No	-	No
Multivariate	No	Yes	Yes	Yes	Yes
Explainable	Yes	No	Yes	Yes	Yes

We found that, compared to the other algorithms we have reviewed, STReamRHF inherently satisfies most of our desired properties, namely P2, P4, P5, P7, and P8. We will discuss how the algorithm is modified to satisfy P1, P3, and P6.

2.3 STReamRHF anomaly detection algorithm

2.3.1 Random Histogram Forest

We first need to understand how RHF works. RHF relies on an ensemble of probabilistic binary tree that partitions data according to the kurtosis (a.k.a. "tailedness") of the attributes. The higher the kurtosis of an attribute, the more likely that the data will be split based on that attribute [11]. The idea is that attributes with higher kurtosis is more likely to isolate anomalies [11]. Intuitively, observations that are grouped into nodes with fewer instances are more likely to be anomalous [3].

Building the tree For each split, an attribute a_s is randomly selected based on its kurtosis value $K(a_s)$, and a value val_a is randomly selected from the possible values of a_s. Subsequently, instances where the value of $a_s < val_a$ is passed onto the left subtree, and the rest to the right subtree. The process repeats on the child nodes until one of the following condition happens:

1. There is only one instance left
2. The kurtosis value is zero (i.e., all values of each attributes are identical)
3. Maximum depth is reached

For a dataset X with $d+1$ number of attributes, the selection of a_s is as follows [3].

1. Kurtosis values of an attribute $K(X_a), X_a \subseteq X$ are individually computed, and the sum is calculated: $K_s = \sum_{a \in [\![0,d]\!]} K(X_a)$
2. A random value r is uniformly chosen between 0 and K_s.
3. The splitting attribute a_s is chosen based on r.

$$GetAttribute(X, r) = a_s = argmin \left(k \in [\![0, d]\!] \middle| \sum_{a=0}^{k} log[K(X_a) + 1] > r \right) \quad (2.1)$$

This a_s selection method implies that an attribute with higher kurtosis is more likely to be chosen.

Anomaly scores For each Random Histogram Tree RHT_i in the forest F and an instance p, the anomaly score of p with respect to RHT_i is calculated as follows [11].

$$RHT_i(p) = log \left(\frac{n}{|S(Q_j)|} \right), \quad p \in S(Q_j) \quad (2.2)$$

where n is the number of instances in the whole tree, $S(Q)$ denotes a set of unique instances in the leaf node Q, and Q_j is a leaf node containing p. This is also known as Shannon's information [11]. To calculate the anomaly score for p, we simply sum $RHT_i(p)$ from all tress in the forest.

$$Score(p) = \sum_{i \in F}^{i} RHT_i(p) \quad (2.3)$$

2.3.2 STReamRHF

STReamRHF is a modification to RHF that allows it to handle streaming data [3]. First, the forest is built like a usual RHF using N initial instances. These instances will be assigned an anomaly score as usual.

Inserting a new instance When a new instance x is inserted, STReamRHF maintains each tree as follows.

1. Starting from the root node, for each node S, STReamRHF calculates kurtosis score $K(X_S \cup x)$, where X_S denotes a set of instances in S.
2. It uses equation 2.1 to determine the new split attribute $a_s = GetAttribute(X_S, r)$, where r is the random value associated with the node.
3. If a_s changes, STReamRHF rebuilds the whole subtree.
4. Otherwise, the value of the split attribute of x is compared to the random split value val_a of the node. The algorithm then moves to either the left or right subtree, and repeat the process until it rebuilds a subtree, or reaches a leaf node.
5. Once x is placed into a leaf node, $Score(x)$ is calculated.

Removing old instances STReamRHF handles concept drift by utilising a *reference window*. Essentially, it rebuilds the whole forest when the number of new instances has reached N. This allows STReamRHF to forget old instances and adapt to a 'new normal' when concept drift occurs.

Advantages First, it is unsupervised and thus require no labeled data to work. Second, it is computable in linear time [3] and only depend on the window size, forest size, and number of attributes. Third, the parameters do not need to be tuned manually [3], hence it can work with minimal human intervention. Fourth, it can detect anomalies from multiple time series (multivariate), therefore can find anomalies in the correlation between multiple time series. Fifth, it can adapt to concept drift. Finally, it is explainable due to trees being inherently white box [3]. These advantages make it suitable for network monitoring use cases.

Limitations Firstly, even though the memory usage is upper bounded, it still uses a considerable amount of memory. Unlike Holt-Winters algorithm, which models time series as variables and stores them, STReamRHF keeps the actual values in memory. At most, it stores $2w$ instances, where w is the window size. With plenty of metrics and

a larger window size, memory usage must be carefully considered. Secondly, it does not encode the chronological order of instances. Consequently, the algorithm cannot detect anomalies in seasonality.

2.4 Handling seasonality in anomaly detection

Since STReamRHF cannot handle seasonality in time series, we refer to the literature on some techniques that other anomaly detection algorithms employ to handle seasonality. Such techniques may be adaptable to STReamRHF to augment its capability.

One such technique is to directly model the seasonality of the time series and use the model for anomaly detection. For instance, Holt-Winters Aberrant Behaviour Detection [2] uses past data to create a model and generate a forecast of expected "normal" data. In modeling the time series, it also accounts for seasonal trend of the data. Therefore, the generated forecast will closely match the seasonality of the actual time series. When the actual data deviates from the forecast, the algorithm assigns a high anomaly likelihood. This is true for all forecast-based and reconstruction-based anomaly detection techniques that can model time series seasonality, such as [12] and [13].

Another way of handling seasonality is by removing the seasonal component of a time series. *Differencing* is a common method of subtracting the current observation with the previous observation, which could eliminate trend and seasonality [14]. This process would allow the anomaly detector to focus on unchanging properties of the time series.

Another commonly used method is to use *shingling* to represent time series [15]. Suppose we have a stream $S = [t_1, t_2, t_3, t_4, t_5]$. A shingle with size 3 will turn the first 3 values of the stream into a three-dimensional representation $[t_1, t_2, t_3]$. When t_4 is processed, the shingle is changed to $[t_2, t_3, t_4]$. The shingle window slides forward as new instances come. The idea is each shingle encodes a chronological pattern of the time series, allowing the anomaly detector to have a notion of time. During RRCF's experimental evaluation [15], shingling proved to be effective in preprocessing time series data for anomaly detection.

2.4.1 Summary

We have established eight desired properties of an anomaly detector for network monitoring purposes. We have reviewed the literature for alternative anomaly detection algorithms and we found STReamRHF to be the most compatible with our desired properties. Although, we discovered that STReamRHF is not equipped to detect seasonal anomalies. Therefore, we looked into the literature to explore methods of handling seasonality in anomaly detection.

Chapter 3

Our Approach

This section describes our approach in detecting anomaly in time series anomalies for network monitoring purposes. We start by stating our formal definition of anomaly, and proceeding to describe our proposed anomaly detection approach.

3.1 Definition of anomaly

There are several broad definitions of an anomaly from the literature [16]. Some of them are more conceptual [17, 18] while some others are more technical [19, 20]. In summary, an anomaly is an observation that differs significantly from the other observations. While there are several types of anomalies that can happen in a time series, only some of them have been observed in the Telstra EDN DNS monitoring data stream. For the purpose of this work, we focus on the types of anomalies that have been observed in the Telstra EDN DNS monitoring data.

Point anomaly Point anomaly, also known as global anomaly, is usually defined as an observation that is significantly different than the remaining data points [16]. For the purpose of this work, we define *point anomaly* as an instance that has a value significantly higher or lower than what was observed in the past within the same time series. Formally, given a time series $S = \{t \mid t \in \mathbb{R}\}$ and an observation $x \in \mathbb{R}$, x is a point anomaly if $\forall t \in S : |x - t| > k$ where k is reasonably large. Note that this definition is only applicable to a univariate time series.

Seasonal anomaly We define seasonal anomaly as observations that fall within the normal range, but they do not usually occur at that specific time. Consider the following example. Based on past observations, CPU usage normally remains stable at around 20%, but spikes to almost 80% at 12am because there is a scheduled job that runs at that time. If at 3pm the CPU usage is observed to be at 70%, this is an anomaly, even though 70% still falls within the normal operating range.

Correlation anomaly There are observations that can be normal at a specific context but anomalous at a different context [21]. In this work, we define *contextual anomaly* to be an instance with multiple attributes in which the values of the individual attributes are not anomalous, but they do not normally coincide.

Suppose there are two attributes A and B in a dataset D. Let $Val(Z)$ denote the unique values of attribute Z, $a \in Val(A)$ and $b \in Val(B)$ denote the value of attribute A and B in instance X respectively. An instance X is a contextual anomaly if the probability of a and b occurring together is significantly lower than other combinations of values, even though *individually*, the values of a and b are considered normal.

Further, we define *correlation anomaly* as an extension of contextual anomaly. Given two distinct time series that are *normally* correlated, a correlation anomaly happens when that correlation no longer holds (breaks). We can consider correlation anomaly as a contextual anomaly.

Consider A_1 and B_1 are the first point in time series A and B respectively. When A and B are correlated, A_1 and B_1 normally occur together. However, during a correlation anomaly, A_1 and B_1 will have values that do not usually occur together. It implies that the probability of A_1 and B_1 occurring together is lower than other combinations of values, which is the definition of a contextual anomaly.

3.2 Proposed anomaly detector

We propose an anomaly detector based on STReamRHF that aims to fulfill the desired properties outlined in Section 2.1. Although, there are some modifications that we incorporate which we discuss below.

3.2.1 Encoding time series using shingling

Recall that STReamRHF on its own does not satisfy the desired property of handling seasonality (P6). To address this, we need to provide a notion of chronological order to the detector. Hence, we incorporate *shingling* method to create shingles with a certain size.

Normally, shingling of size N is done using consecutive N observations. For instance, given a sequence of observations $[T_1, T_2, T_3, T_4]$ and a shingle size of 3, the first shingle will consist of $[T_1, T_2, T_3]$. However, in a data stream, the detector do not have access to future observations. Therefore, we use N past observations instead. That is, shingle at time step i will consist of observations $[T_{i-2}, T_{i-1}, T_i]$.

What happens to the first two observations that do not have past data yet? We use the latest observed value as past value. Hence, at time steps 0 and 1, the shingle will be $[T_0, T_0, T_0]$ and $[T_0, T_0, T_1]$ respectively. Using the latest observation as padding instead of a constant value ensures that the shingle is not too different than the rest.

Handling multivariate data The shingling method we have describe only works for a single time series. Thus, we need to define how to shingle the data in multivariate settings. We opt to use a simple approach to do this. With N attribute and a shingle size of S, the algorithm will create a $(N \times S)$-dimensional column to feed into the forest, where the first S columns contain the shingle of the first attribute, and so on.

3.2.2 Modification to the STReamRHF insertion algorithm

During our experimental evaluation, we observed that STReamRHF struggles to detect an extreme point anomaly in the streaming dataset. Consider the example in Figure 3.1, where the vertical red line indicates when the data starts to be streamed. The second and third plot show the anomaly scores when all data points are scored in batch during the initial RHF build, and when some of them are streamed to the original STReamRHF algorithm respectively. Observe that the extreme point anomalies at around time step 650 are detected when batched, but undetected when streamed. This suggests a limitation in the original STReamRHF algorithm that prevents it from detecting such extreme anomaly.

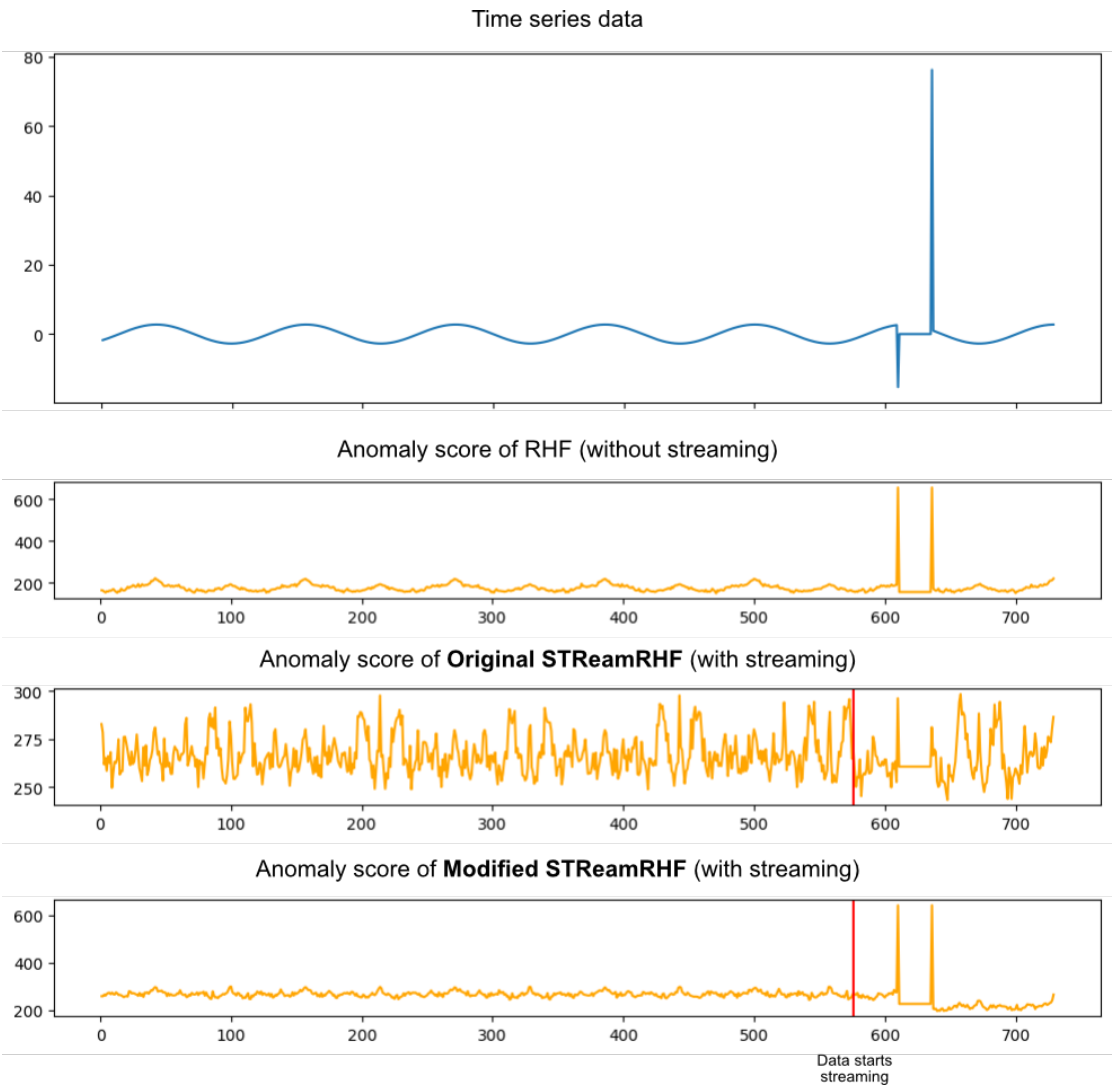

FIGURE 3.1: Plot showing a synthetic time series and anomaly scores of the original and modified STReamRHF algorithm

This limitation comes from the fact that the random split value of each node is not updated when new data is inserted. Suppose initially, dataset X contains normal values between 2.00 and 5.00. Recall that the split value is a random value between the minimum and maximum values of instances in a node [11]. As a result, the split value of each node will fall between 2.00 and 5.00. Then, suppose a new observation with a value of 87 is inserted into the tree. Since there is no node that splits at above 5.00, this new observation will likely be grouped with instances with values of around 5.00 instead of being isolated. This results in a lower anomaly score, as seen in Figure 3.1.

To address this issue, we modify the insertion algorithm to accommodate for change in the value range. If the new observation falls outside of the range of current instances, the subtree will be rebuilt. Algorithm 1 shows our proposed modification in blue. We have

experimentally evaluated this fix as seen in Figure 3.1, where the modified algorithm is shown to have detected the point anomaly in the data stream. This modification, however, results in a more frequent rebuilding of the forest, leading to a higher average computation time.

Algorithm 1 ModifiedInsert(node, i, h, z) (adapted from [3])

Input *node*: node of RHT, i: instance, h: max tree height, z: seed of array of 2^h
Output Random Histogram Tree *rht*

```
1: if node is not Leaf then
2:     kvalues = kurtosis(node.data ∪ i); ▷ same function as in the batch, but the seed of every node is initialized using values of z
3:     a_split = get-attribute(kvalues, z[node.id]);
4:     if node.attribute ≠ a_split then
5:         return RHT-build(node.data ∪ i, node.height);
6:     end if
7:     if i < min(node.data) or i > max(node.data) then
8:         return RHT-build(node.data ∪ i, node.height);    ▷ This is the modification
9:     end if
10:    if i[node.attribute] ≤ node.value then
11:        node.left = Insert(node.left, i, h, z);
12:    else
13:        node.right = Insert(node.right, i, h, z);
14:    end if
15: else
16:    if node.height = h then
17:        node.data = node.data ∪ i;
18:        return node;
19:    else
20:        return RHT-build(node.data ∪ i, node.height);
21:    end if
22: end if
```

3.2.3 Normalisation of anomaly score

STReamRHF calculates the Shannon information content as the anomaly score [11]. This means that the score has no upper bound. Furthermore, different time series could have different anomaly score threshold. For example, in a time series with low noise, 70 can be considered anomalous, while in a noisy data, 300 can be considered normal. Manually setting different thresholds for each time series is undesirable as it contradicts the desired property of requiring no human intervention (P4). Thus, we need to normalise the score such that it is interpretable as an anomaly likelihood.

We normalise the score using an exponential function to limit the range of possible values to be between 0.00 and 1.00. Equation 3.1 shows our proposed method of score normalisation. μ and σ denote the average and standard deviation of all Shannon information content values of instances in the entire tree respectively, and n is a configurable parameter which we call *normalisation coefficient*.

$$Score(p) = \frac{1}{|F|} \sum_{i \in F}^{i} 1 - e^{-RHT_i(p)/(\mu + n\sigma)} \tag{3.1}$$

There are several notable modifications to the scoring method. First, instead of summing all scores from all trees in the forest, we instead use the average score of each tree. Second, the use of $1 - e$ scales the output range to be between 0.00 and 1.00. Finally, to address the issue of different time series having different thresholds, we divide the Shannon information content with $\mu + n\sigma$. This normalisation implies that Shannon information content values higher than n standard deviations from the mean are likely to be anomalous. Therefore, we will have a standardised score for different time series. Additionally, n is a configurable parameter that specifies the sensitivity of the detection; lower value means the detector is more sensitive to deviations and vice versa.

3.2.4 Determining anomaly labels

A simple way of detecting anomalies from the score is to consider instances with $score > 0.5$ as anomalous. However, this method often yields a high number of false positive, which would contradict with the desired property of favouring false negative over false positive (P1).

In the EDN DNS dataset, we have observed cases where the value spiked to extreme levels in one time step and immediately went back down. The EDN DNS team confirmed that these are not true observations, but rather an error in the measurement, thus we cannot consider such spikes as true anomalies. Figure 3.2 illustrates such occurrence.

To make the anomaly detection more robust, we employ the moving window technique similar to what is described in [2]. If the number of observations within δ previous observations where the anomaly score is higher than 0.5 exceed a specified threshold m, we say that an anomaly is detected.

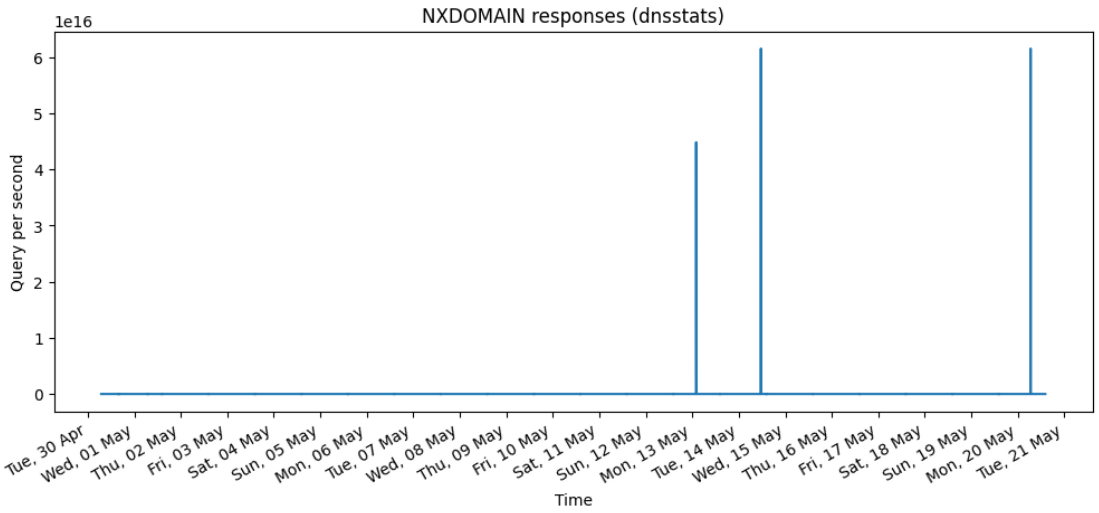

FIGURE 3.2: Numer of NXDOMAIN responses with four unusually high spikes

Formally, an anomaly is detected at time step t if and only if

$$s(t) > 0.5 \quad and \quad |S| > m; \quad S = \{s(p) > 0.5 \,|\, p \in [(t-\delta) \ldots t]\} \tag{3.2}$$

where $s(t)$ denotes the anomaly score at time step t, m denotes the threshold, and δ denotes the window size.

Chapter 4

Results and Discussion

4.1 Experimental evaluation

In this experimental evaluation, we focus on detecting anomalies that have been observed in the Telstra EDN DNS monitoring system. First, we will evaluate the anomaly detector using a synthetic data that is visual and verifiable. Then, we evaluate it using some of the data captured from EDN DNS system. Finally, we will run Numenta Anomaly Benchmark to evaluate the detector quantitatively.

4.1.1 Synthetic dataset

In this section, we evaluate the detector using synthetically generated time series that are visual and verifiable. We generated three synthetic time series, one for each type of anomaly defined in Section 3.1.

Data stream We divide the data into two sets: batch and streaming. In the first set, the first 750 instances are used to build the initial RHF. Then in the second set, we treat the data as a stream; given observations $[750, \ldots, i]$, the detector will calculate the score for the $(i + 1)$-th observation. Our aim is to look at the anomaly score qualitatively. Note that the anomaly labeling process described in Section 3.2.4 is not used in this evaluation.

Parameters For the purpose of this evaluation, we use the following parameters. Firstly, for the tree size, we use $t = 100$ as it is normally used in evaluating most

ensemble methods [11]. Secondly, for the maximum tree depth, $d = 5$ is used per the recommendation of the original author [11]. Thirdly, we set the maximum window size arbitrarily to $w = 2048$, since the authors of STReamRHF states that the algorithm is not sensitive to the choice of window size [3]. Finally, we use a shingle size of 5 and normalizing coefficient of 2, which are arbitrarily set.

Experiment E1: Point and collective anomaly We first evaluate the detector using a time series plotted in Figure 4.1. The time series has three different point anomalies with different length. The first point anomaly is located when the RHF is initially built. The second point anomaly is located after the data is streamed. The third anomaly is much longer than the other two to evaluate how would the algorithm behave in the presence of collective anomalies.

Figure 4.1 also shows the calculated anomaly scores for the modified STReamRHF algorithm when shingling is used and not used. Observe that both methods can identify point anomalies with no problem, both when the data is batched (before streaming) and streamed (after streaming). It seems to be able to handle collective anomalies as well. Furthermore, the algorithm is able to detect global anomalies at the start of each anomalies.

Experiment E2: Seasonal anomaly Real-world time series data is often loosely described by a sinusoidal rhythm [8]. For example, the number of DNS queries is expected to have peaks during work hours, and dips during the night when most staff stop working. An anomaly occurs when the observations do not follow the usual pattern. To that end, we evaluate the detector using a sinusoidal wave with injected anomaly, as seen in Figure 4.2.

We can observe in Figure 4.2 that, without shingling, STReamRHF struggles to detect the seasonal anomalies. In contrast, the algorithm is able to detect the start and end of each anomaly when shingling is applied. However, the algorithm seems to only detect the transition between normal and anomalous observations, as opposed to the entire length of each anomaly. This is especially apparent at the third anomaly where the data flatlines; the anomaly score spikes at the transition point, but then drops.

Experiment E3: Correlation anomaly The purpose of this evaluation is to visually verify whether the detector is capable of detecting anomalies in the correlation

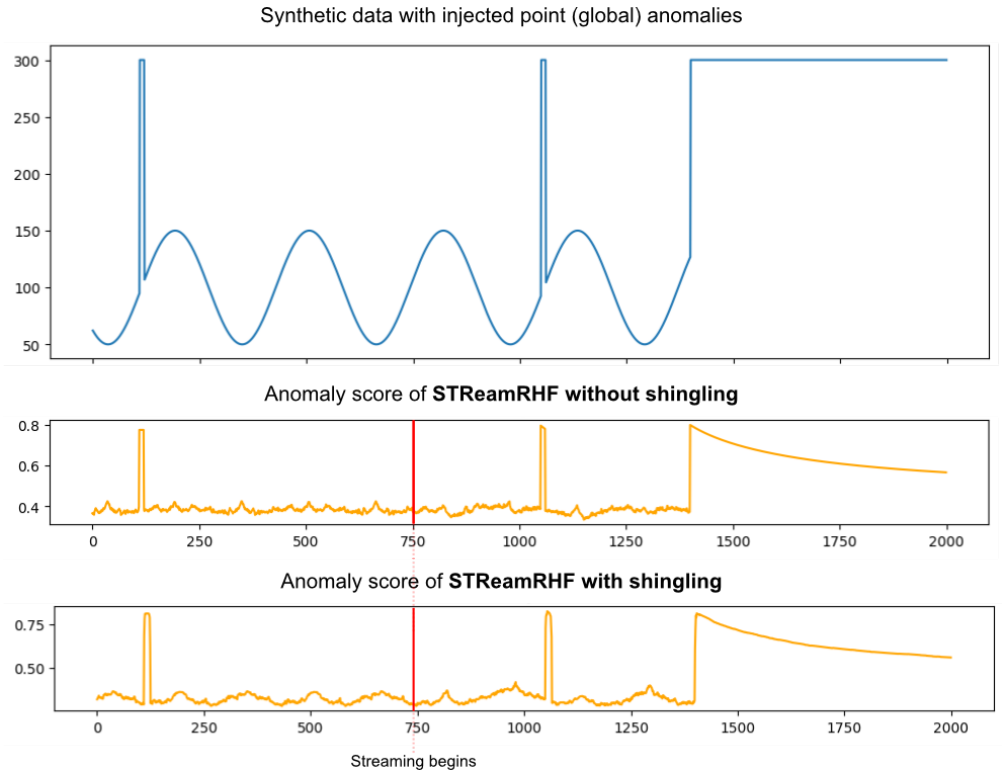

FIGURE 4.1: Anomaly scores for STReamRHF with and without shingling on synthetic data with point anomalies

between two time series. As an example, in Telstra EDN DNS, *number of incoming UDP packets* must be somewhat correlated with *number of outgoing UDP packets*, since a healthy system will respond to the incoming UDP packets. When a fault happens that prevents the EDN DNS system from responding to all incoming packets, the number of outgoing UDP packets will dip while the number of incoming UDP packets remains unchanged. Notice that independently, both time series may not be anomalous, but only the correlation between the two time series is anomalous.

To evaluate the detector's performance on detecting anomaly in the correlation, we generate two sinusoidal time series that have similar values up to a certain point. After that point, the amplitude of one of the wave is decreased. This is done to simulate a deviation in the correlation between the two time series. The synthetic time series and its anomaly scores are shown on Figure 4.3.

While STReamRHF seems capable of detecting deviation in the correlation, the use of shingling makes the detector unable to detect such deviation. At time step 1230, the anomaly score of the detector without shingling climbs gradually until it peaks at around time step 1250, where the maximum deviation occurs. However, the difference

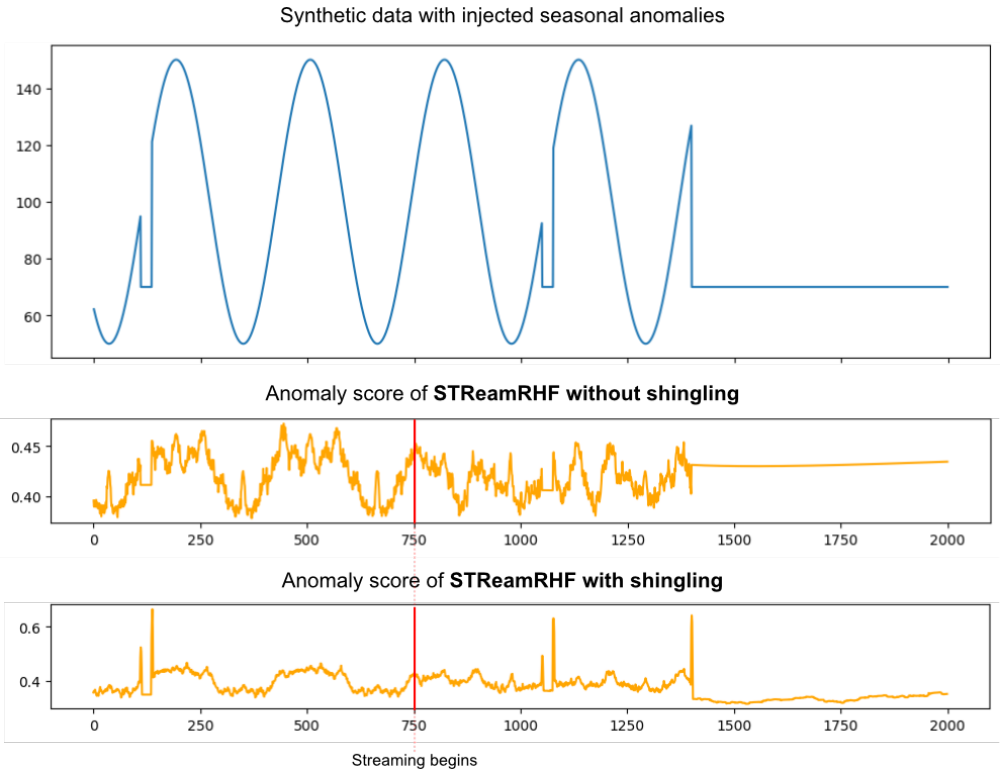

FIGURE 4.2: Anomaly scores for STReamRHF with and without shingling on synthetic data with seasonal anomalies

between anomaly score of true anomaly and noise is not significant enough to make a definitive detection. Additionally, when shingling is enabled, the detector fails to detect the anomaly, with its peak anomaly score below 0.5.

4.1.2 Real-world data: Telstra EDN DNS monitoring

Next we evaluate the detector using a real-world data captured from the EDN DNS system. Similarly, we split the data into batch and stream sets, using the first 750 instances as the batch set. We then qualitatively evaluate the anomaly scores. The data is collected every 5 minute from 1/5/2024 to 21/5/2024.

Parameters Similar to the synthetic data evaluation, we use tree size $t = 100$, maximum tree depth $d = 5$, window size $w = 2048$, and normalizing coefficient $n = 2$. However, we set the shingle size to 288, which corresponds to 24 hours of observations. The intuition is that, since most time series fluctuate daily, the past 24 hours captures the usual pattern of EDN DNS usage. For the purpose of anomaly labeling, we set window size $\delta = 30$ and threshold $m = 5$.

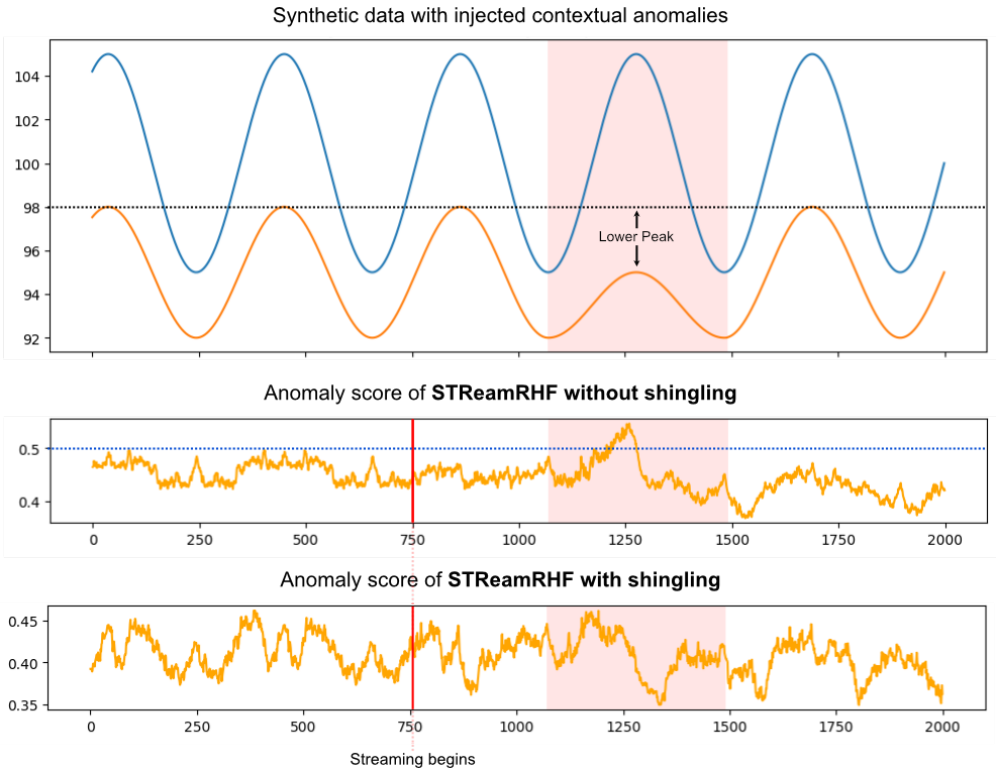

FIGURE 4.3: Anomaly scores for STReamRHF with and without shingling on synthetic data with correlation anomalies

Anomaly labelling We rely on known incidents and expert judgment of the EDN DNS team to label the anomalies in the time series. We looked at the collected data and select three time series that contain some interesting patterns. Then, we consult an expert in the EDN DNS team to label the point where each anomalies occur. For the purpose of the evaluation, we consider a positive detection as true positive if it happens within an *anomaly window*. For an anomaly at time t, the anomaly window starts from $t - d$, where d is 10% of the number of instances in the dataset divided by number of anomalies, in line with [4].

Experiment E4: Number of NXDOMAIN responses Within the data collection timeframe, the Telstra EDN DNS team encountered a known issue related to a fault in a DNS client. During a routine check of the system, a team member noticed an unusual pattern in the number of NXDOMAIN responses, which is a DNS response code returned when a client attempts to query non-existent domain name. Figure 4.4 shows the number of NXDOMAIN responses every 5 minutes within the collection period. The anomaly window is shaded in red, while positive detections are shaded in green.

Observe that the anomaly detector has successfully identified the anomaly within the

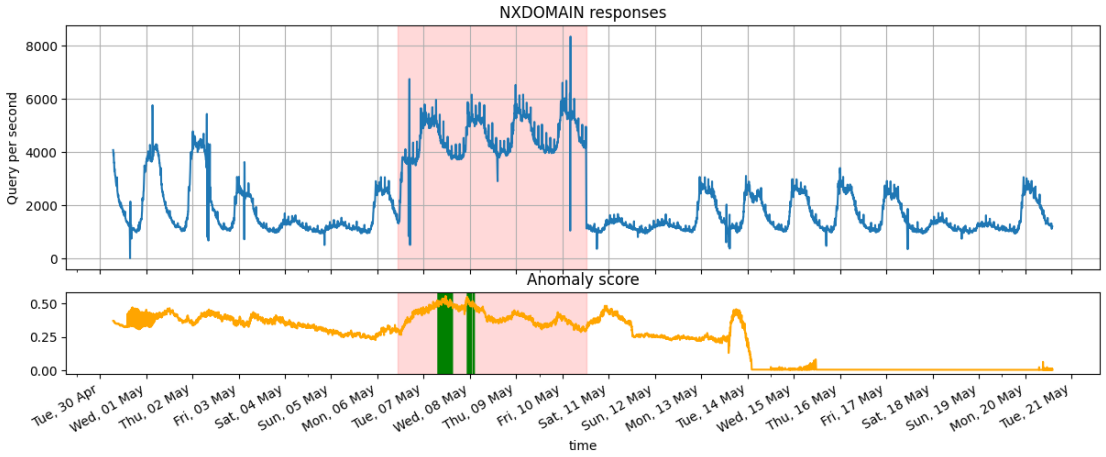

FIGURE 4.4: Number of NXDOMAIN response with anomaly scores

anomaly window. Furthermore, there are no false positive readings outside the anomaly window. However, the anomaly is detected in the middle as opposed to immediately after the anomaly starts. It takes almost 24 hours worth of observation before it makes a positive detection.

Experiment E5: Number of running processes In some servers, we noticed an abnormal pattern in the number of running processes, where the number dips for several hours, and goes back up. In some other instances, the number dips and stagnates for a long period of time (weeks or months). While the cause of both instances are unknown at the time of writing, the Telstra EDN DNS team has confirmed that it is an abnormal behaviour and something that would need to be investigated. Figures 4.5 and 4.6 shows the number of processes in two servers where the number of processes suddenly drops. Observe that, in server A, the number of processes jumps back up after some time, while in server B, the drop stays continuously.

In both cases, the anomaly detector successfully detected the anomalies within the anomaly window, and no false positives are generated. Furthermore, the anomaly scores in Figure 4.6 gradually decreases as it learns the new normal.

Experiment E6: Number of imcoming and outgoing UDP packets The aim of this evaluation is to see how the algorithm performs in multivariate settings. The third dataset that we evaluated consists of two time series, which are the number incoming and outgoing of UDP packets. Both time series should be strongly correlated to one another. While there were no recorded faults that could have caused the correlation between the two time series to break, the same anomaly as in Experiment E4 also

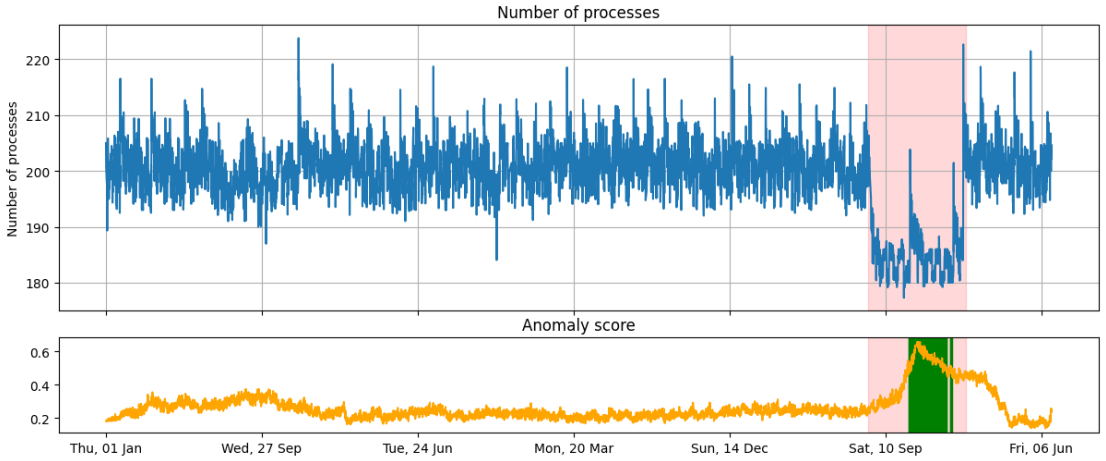

FIGURE 4.5: Number of running processes in server A

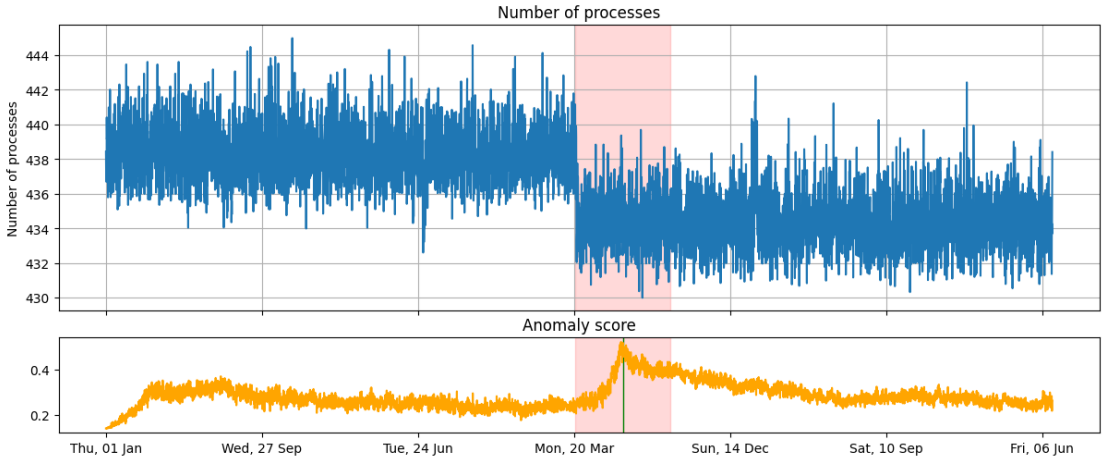

FIGURE 4.6: Number of running processes in server B

appears in each of the time series. Figure 4.7 shows the time series plot and computed anomaly scores, where the known anomaly window is shaded red, and positive detections are shaded green. Observe that our algorithm did not generate positive detections within or outside of the anomaly window.

4.1.3 Benchmarking with Numenta Anomaly Benchmark

Numenta Anomaly Benchmark (NAB) is a benchmarking system for streaming anomaly detectors [4]. When scoring the detectors, NAB provides the data points one-by-one, ensuring the detector does not have access to future values, in line with our desired properties of *detecting anomalies as new data points come*. Furthermore, in addition to precision and recall, NAB also considers how early an anomaly is detected to reward early detections. This is in line with our desired properties of *detecting problems as soon*

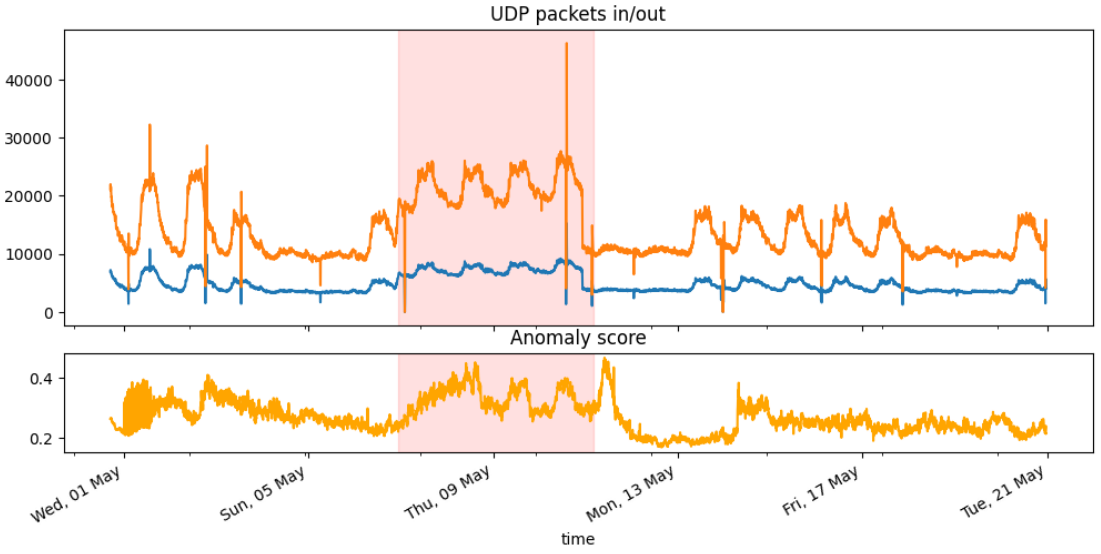

FIGURE 4.7: Number of UDP packets in and out

as possible (P3). Furthermore, NAB comes with an open source time series data with labeled anomalies, which contains both synthetic and real-world data.

Parameters To evaluate our proposed anomaly detector, we use window size of 2048, tree size of 120, max depth of 6, normalisation coefficient $n = 2$, and shingle size of 10.

Results Table 4.1 shows the calculated NAB score.

TABLE 4.1: Numenta Anomaly Benchmark results

Profile	Score
Standard profile	16.09
Reward low false positive	7.32
Reward low false negative	23.31

Since one of the desired properties of the anomaly detector is favouring false negatives over false positives, we need to consider the *reward low false positive* profile. Compared to other detectors in the NAB scoreboard [4], our anomaly detector ranks at the bottom, ranking below Etsy Skyline but above Bayesian Changepoint and EXPoSE. Furthermore, it ranks below RRCF, which is another tree-based method. We will look at some of the time series used in the benchmark later in the discussion section to dig deeper.

4.2 Discussions

4.2.1 Univariate anomaly detection

At first, our proposed anomaly detector seems promising when evaluated with univariate synthetic data. In Experiment E1, we have shown that the detector is capable of detecting the start of anomalies. This is also true in Experiment E2, where the detector is able to identify the start of the anomaly with shingling applied. This led us to believe that shingling would help encoding the time series to allow STReamRHF to detect seasonal anomalies.

However, the NAB score of our proposed anomaly detector is relatively low compared to other detectors. A closer look into some of the individual time series revealed that our detector only works well for *point* anomalies and performs badly for other types of anomalies during the NAB benchmarking. Why?

We hypothesize that the issue lies on the choice of shingle size. The main idea of shingling is that a shingle encodes the *pattern* of previous n observations, thus allowing STReamRHF to detect anomalies in the fluctuation pattern of the time series. This explains why the detector performed well during Experiment E2, since an abrupt value change is not a common pattern in a perfectly sinusoidal data and can be isolated easily.

However, shingling would not detect unusual patterns if the shingle size is too small compared to the season length of the fluctuations. With a small shingle size, the shingle only captures noise in the data, whereas a sufficiently large shingle captures both the sudden jump, peak, and sudden drop of the time series. A larger shingle size would allow the detector to deduce that the jump is unusually lower and assign a higher anomaly score.

To prove our hypothesis, we tested the detector with a much larger shingle size. Figure 4.8 shows the time series[1] and anomaly scores when the shingle size is 5 and 500. We can observe that, when the shingle size is 5, the anomaly score does not suggest that the anomaly is detected. On the contrary, when the shingle size is 500, we can see that the

[1] github.com/numenta/NAB/blob/master/data/artificialWithAnomaly/art_daily_jumpsdown.csv

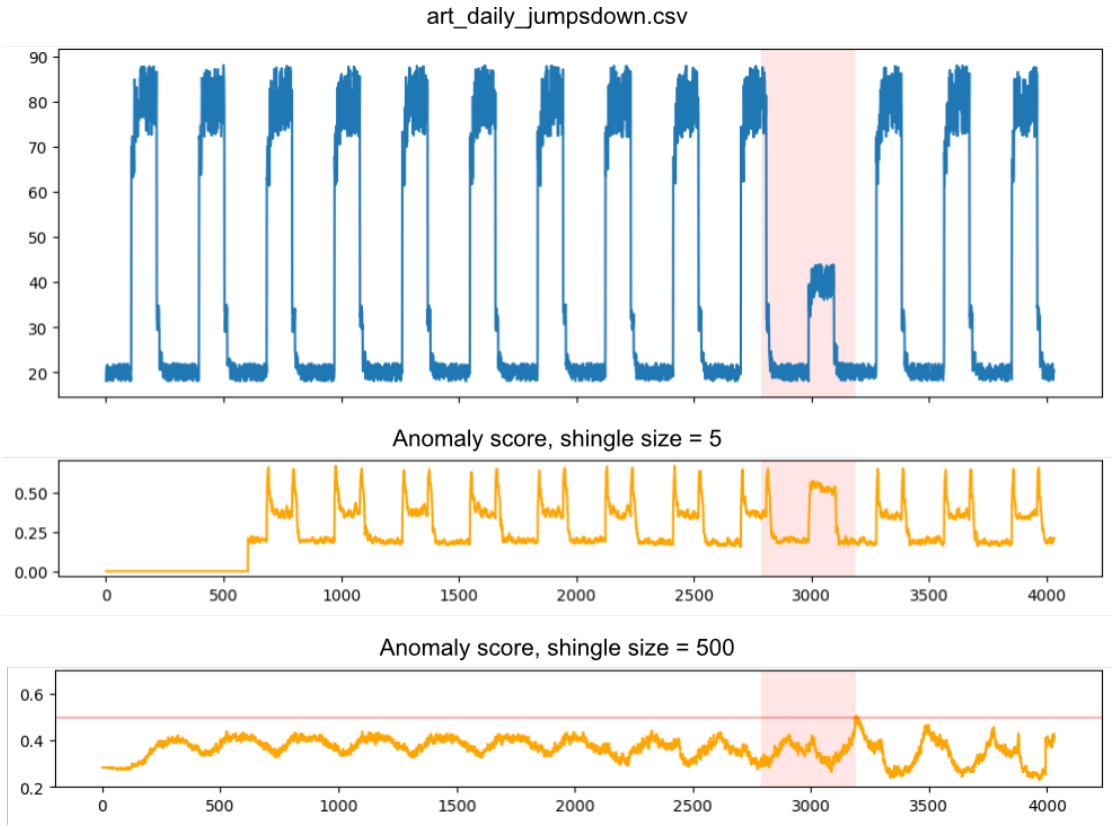

FIGURE 4.8: Anomaly scores of art_daily_jumpsdown.csv with shingle size of 5 and 500

score increases when the abnormal pattern is detected. Although, it is only detected at the end of the anomaly window (shaded in red).

We deduce that the shingle size must be sufficiently large to allow STReamRHF to capture patterns that occur over longer time steps. For seasonal time series, we recommend the value to be larger than half the period length so that seasonal deviations will be captured by the shingle. One thing to note, however, doing so would increase the worst-case complexity and memory footprint of the algorithm, thus significantly increasing the detection time. In addition, larger shingle size could cause anomalies to be detected later.

To that end, during the second evaluation using real-world Telstra data, we used a larger shingle size that corresponds to 24 hours worth of observation (288 time steps). In all three tests, the anomaly detector successfully identified all anomalies without generating false positives. Although, the anomaly was detected almost 24 hours since the start of the anomalies.

Additionally, Experiment E5 has shown that the anomaly detector is capable of adapting to concept drift (property P5). This is apparent in Figure 4.6, where the gradual decline of the anomaly score suggests the detector's ability to adapt to a new normal.

Although, please note that the tests were conducted in a small subset of real-world data that is labeled by human experts. While the results seem promising, we need more labeled data to accurately gauge the performance of the detector.

4.2.2 Multivariate anomaly detection

We evaluated the multivariate capability using a synthetic dataset and real-world dataset captured from the EDN DNS system.

In the synthetic data evaluation (Experiment E3), it appears that shingling renders the detector unable to detect anomalies in the correlation between two time series. This is evident because when shingling is used, there is no apparent peak in the anomaly score within the anomaly window.

In the real-world evaluation (Experiment E6), we used two time series: number of incoming and outgoing UDP packets. Recall that these time series should be correlated with one another because, in a healthy system, all valid incoming packets should be responded to. A break in the correlation would indicate some problems with the system.

The EDN DNS team had recorded no faults that could have caused the correlation between the two time series to break. Therefore, we expected the detector to have no positive detections regarding the correlation anomaly. However, the known anomaly that we have seen in the Experiment E4 also manifested in these time series (shaded red in Figure 4.7). Here, we expected the detector to generate some positive detections within the anomaly window.

Interestingly, the detector made no positive detections. In some ways, the detector performed well because it resulted in no false positive when detecting correlation anomalies. However, when detecting two correlated time series, it failed to detect anomalies in individual time series.

This led us to believe that our naive method of combining shingling and multiple variables described in Section 3.2.1 is not optimal. It appears that shingling undermines

the ability to detect correlation anomalies, as evident in Experiment E3. Additionally, having multiple shingles of different variables reduces the effectiveness of shingling in detecting anomalies, as seen in Experiment E6.

A more robust method is to use use separate trees for detecting anomalies in univariate and multivariate data stream. The first type of tree would focus on detecting anomalies in a single time series. Meanwhile, the second type of tree would focus on detecting the anomaly in the correlation between two (or more) time series. This way, we could detect point and seasonal anomalies in each time series, while also detecting correlation anomalies in a combination of multiple time series.

Although, we would like to highlight a limitation of using STReamRHF to detect correlation anomalies. In a multivariate input, given an instance with two variables A and B, STReamRHF will isolate instances where the values of A and B do not normally appear in the same instance. This method would work when time series are *normally* correlated and anomalies happen when the correlation breaks. However, this method would not work when the time series are normally not correlated, but anomalies happen when the time series are suddenly correlated. Therefore, an additional method should be incorporated to handle such anomalies.

4.2.3 Summary

We have deduced that our proposed anomaly detector could detect point and seasonal anomalies in univariate time series. However, the effectiveness lies on the choice of shingle size. A larger shingle size could capture seasonality that happens over a longer period of time, but it could cause anomalies to be detected later. Furthermore, a larger shingle size increases the worst-case complexity of the algorithm.

In addition, we deduced that STReamRHF is able to detect correlation anomalies. However, when coupled with shingling, this capability is drastically reduced. We therefore recommend to create a separate forest for detecting univariate and multivariate time series. We also have highlighted the limitation of STReamRHF in detecting correlation anomalies, where it only works to detect breaks in correlation, as opposed to the appearance of a new correlation.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have evaluated the use of shingling with STReamRHF using synthetic data, real-world data collected from Telstra EDN DNS system, and the Numenta Anomaly Benchmark. While the results of the synthetic and real-world data are promising, the NAB score is relatively low compared to other detectors.

During our investigation, we have found that using STReamRHF with shingling and score normalisation is effective to detect point and seasonal anomalies in a streaming data, given a sufficiently large shingle size. However, a larger shingle size would translate into higher computational complexity and later detections. Furthermore, the use of shingling undermines STReamRHF capability to detect correlation anomalies.

To that end, we conclude that our proposed detector has satisfied properties P1, P2, P5, P6, and P8. On the contrary, we cannot yet conclude that our proposed detector has satisfied P3, P4, and P7. For P3, how early the detector can detect anomalies highly depend on the choice of shingle size, which will also determine the detection effectiveness. This relates to P4, where the performance of the detector is sensitive to the choice of shingle size. For P7, while the detector is inherently multivariate, we have showed that our proposed detector is ineffective in detecting correlation anomaly. Therefore, we conclude that our proposed detector is not yet optimal for network monitoring.

5.2 Future work

There are more research that can be conducted to further this work.

- Firstly, future research may attempt to explore alternative methods other than shingling to represent time series. Furthermore, forecasting techniques may be explored to enhance the detector with the ability to detect seasonal anomalies over a longer period of time.
- Secondly, the algorithm we have described in this work can be further optimised to allow for larger dimensions of time series and larger shingle size. Related to that, while STReamRHF has a linear worst-case complexity, it is still quite slow for larger data dimensions. Techniques such as parallelisation can be explored to enhance the scalability of the algorithm.
- Thirdly, we have not covered explainability of the algorithm in depth in this work. Future work may attempt to explore methods of visualising an anomaly to allow operators to justify criticality of the alert and pinpoint the issue easily.
- Finally, future work can look into how to deploy our proposed anomaly detector at a larger scale, when the data volume is more fitting to larger network systems.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *ACM Computing Surveys*, 41:1–58, 7 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882.

[2] Jake D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *Proceedings of the 14th USENIX Conference on System Administration*, LISA '00, page 139–146, USA, 2000. USENIX Association.

[3] Stefan Nesic, Andrian Putina, Maroua Bahri, Alexis Huet, Jose Manuel, Dario Rossi, and Mauro Sozio. STREAMRHF: Tree-Based Unsupervised Anomaly Detection for Data Streams. In *AICCSA 2022 - 19th ACS/IEEE International Conference on Computer Systems and Applications*, Abu Dhabi, United Arab Emirates, December 2022. URL `https://inria.hal.science/hal-03948938`.

[4] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, December 2015. doi: 10.1109/icmla.2015.141. URL `http://dx.doi.org/10.1109/ICMLA.2015.141`.

[5] Guansong Pang and Charu Aggarwal. Toward explainable deep anomaly detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 4056–4057, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3470794. URL `https://doi.org/10.1145/3447548.3470794`.

[6] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Tadgan: Time series anomaly detection using generative

adversarial networks. pages 33–43. IEEE, 12 2020. ISBN 978-1-7281-6251-5. doi: 10.1109/BigData50022.2020.9378139.

[7] Jarkko Ekberg, Jorma Ylinen, and Pekka Loula. Network behaviour anomaly detection using holt-winters algorithm. In *2011 International Conference for Internet Technology and Secured Transactions*, pages 627–631, 2011.

[8] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *ICML 2016*, 2016. URL `https://www.amazon.science/publications/robust-random-cut-forest-based-anomaly-detection-on-streams`.

[9] Leo Breiman. *Mach. Learn.*, 45(1):5–32, 2001.

[10] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi: 10.1109/ICDM.2008.17.

[11] Andrian Putina, Mauro Sozio, Dario Rossi, and José Manuel Navarro. Random histogram forest for unsupervised anomaly detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1226–1231, 2020. doi: 10.1109/ICDM50108.2020.00154.

[12] Tianyang Lei, Chang Gong, Gang Chen, Mengxin Ou, Kewei Yang, and Jichao Li. A novel unsupervised framework for time series data anomaly detection via spectrum decomposition. *Knowledge-Based Systems*, 280:111002, 2023. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2023.111002. URL `https://www.sciencedirect.com/science/article/pii/S0950705123007529`.

[13] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks, 2021.

[14] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018.

[15] Implementation of the robust random cut forest algorithm for anomaly detection on streams. *Journal of Open Source Software, The Open Journal*, 4(35):1336, 2019.

[16] Durgesh Samariya and Amit Thakkar. A comprehensive survey of anomaly detection algorithms. *Annals of Data Science*, 10(3):829–850, Jun 2023. ISSN 2198-5812. doi: 10.1007/s40745-021-00362-9. URL `https://doi.org/10.1007/s40745-021-00362-9`.

[17] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[18] Vic Barnett, Toby Lewis, et al. *Outliers in statistical data*, volume 3. Wiley New York, 1994.

[19] Tianming Hu and Sam Y Sung. Detecting pattern-based outliers. *Pattern Recognition Letters*, 24(16):3059–3068, 2003.

[20] Mon-Fong Jiang, Shian-Shyong Tseng, and Chih-Ming Su. Two-phase clustering process for outliers detection. *Pattern recognition letters*, 22(6-7):691–700, 2001.

[21] Mohammad Braei and Sebastian Wagner. Anomaly detection in univariate time-series: A survey on the state-of-the-art, 2020.