Benchmarking Anomaly Detectors on Streaming Data

Michail Giannoulis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Professor *Vassilis Christophides*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been partially supported by SAP France.

University of Crete
Computer Science Department

Benchmarking Anomaly Detectors on Streaming Data

Thesis submitted by
Michail Giannoulis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Michail Giannoulis

Committee approvals: _____
Vassilis Christophides
Professor, Thesis Supervisor

_____
Ioannis Tsamardinos
Professor, Committee Member

_____
Panagiotis Tsakalides
Professor, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, March 2020

Benchmarking Anomaly Detectors on Streaming Data

Abstract

The experimental evaluation of unsupervised anomaly detection algorithms is a constant challenge within diverse research areas and applications domains. However, little is known regarding the strengths and weaknesses of online anomaly detection methods and the impact of their parameters. This paper elaborates on the design and development of a benchmark framework to perform an extensive experiment study on tree and nearest-neighbor based methods of top-notch unsupervised online outlier detectors (including their offline) in streaming manner, across a wide variety of multivariate datasets contaminated by sub and full space outliers. Initially, we present the semantics and functionalities of the detectors through a comprehensive example. Then, we introduce the benchmark environment providing a descriptive (meta) analysis of the datasets and implementation choices of detectors, posing also the set of their hyper-parameters and candidate values. The fair evaluation of detectors is guaranteed through an adequately analysis of critical methodological questions such as stream simulation and partitioning, evaluation protocols and metrics, detectors optimization and ranking. Through this study, we ascertain that online detectors not only approximate very well offline detectors (2.296 vs 2.266, respectively; Ranking value) but also outperform them under certain conditions. In addition, we surprisingly establish the robustness of online detectors' dynamic model on scaling data and subspace dimensionality. Nevertheless, they shown a decreasing effectiveness while scaling window parameters. We also examine the fundamentals of a dynamic model highlighting the need for a forgetting mechanism. To the best of our knowledge, this is the most complete online anomaly detection benchmark attempt on multivariate data.

Πειραματική Αξιολόγηση Ανιχνευτών Ανωμαλιών σε Ροές Δεδομένων

Περίληψη

Η πειραματική αξιολόγηση των αλγόριθμων ανίχνευσης ανωμαλιών χωρίς επίβλεψη αποτελεί μια σταθερή πρόκληση σε διάφορους τομείς έρευνας και εφαρμογών. Ωστόσο, λίγα είναι γνωστά όσον αφορά τα πλεονεκτήματα και τις αδυναμίες των μεθόδων ανίχνευσης ανωμαλιών εντός σύνδεσης και το αντίκτυπο των παραμέτρων τους. Η παρούσα ερευνητική δημοσίευση αποσκοπεί στο σχεδιασμό και την ανάπτυξη ενός πλαισίου συγκριτικής αξιολόγησης για την εξαγωγή μιας εκτενούς πειραματικής μελέτης πάνω στις δεντρικές και πλησιέστερου γείτονα μεθόδους των κορυφαίων μη επιτηρούμενων ανιχνευτών ανωμαλιών εντός σύνδεσης (συμπεριλαμβανομένου και των εκτός σύνδεσης) σε συνεχή ροή, μέσα από μια μεγάλη ποικιλία από πολυδιάστατα δεδομένα τα οποία έχουν μολυνθεί είτε με ανωμαλίες υποχώρου είτε πλήρους διανυσματικού χώρου. Αρχικά, παρουσιάζουμε τη σημασιολογία και τις λειτουργίες των ανιχνευτών μέσω ενός περιεκτικού παραδείγματος. Στη συνέχεια, εισάγουμε το περιβάλλον πειραματικής μελέτης το οποίο παρέχει περιγραφική (μέτα) ανάλυση των δεδομένων και τις επιλογές υλοποίησης των ανιχνευτών, θέτοντας επίσης το σύνολο των υπερπαραμέτρων και των υποψήφιων τιμών τους. Η δίκαιη αξιολόγηση των ανιχνευτών εξασφαλίζεται μέσω επαρκούς ανάλυσης κρίσιμων μεθοδολογικών ερωτημάτων όπως η προσομοίωση και διαμοιρασμός ροής δεδομένων, τα πρωτόκολλα και οι μετρικές αξιολόγησης, η βελτιστοποίηση των ανιχνευτών και η κατάταξή τους. Μέσω αυτής της μελέτης, διαπιστώνουμε ότι οι ανιχνευτές εντός σύνδεσης όχι μόνο προσεγγίζουν πολύ καλά τους ανιχνευτές εκτός σύνδεσης (2.296 έναντι 2.266 αντίστοιχα, τιμές κατάταξης) αλλά τους ξεπερνούν υπό συνθήκες. Επίσης διαπιστώνουμε με έκπληξη την αντίσταση του δυναμικού μοντέλου των ανιχνευτών εντός σύνδεσης στην κλιμάκωση της διάστασης των δεδομένων και των υποχώρων τους. Πάραυτα, παρουσιάζουν μειωμένη απόδοση καθώς κλιμακώνονται οι τιμές των υπερπαραμέτρων των παραθύρων τους. Εξετάζουμε επίσης τα θεμελιώδη στοιχεία ενός δυναμικού μοντέλου, υπογραμμίζοντας την ανάγκη μηχανισμού λήθης. Από όσο γνωρίζουμε, αυτή είναι η πιο ολοκληρωμένη προσπάθεια πειραματικής αξιολόγησης ανιχνευτών ανωμαλιών σε σύνδεση πάνω σε πολυδιάστατα δεδομένα.

Ευχαριστίες

Καταρχήν, θα ήθελα να ευχαριστήσω τον επόπτη καθηγητή μου κ. Βασίλη Χριστοφίδη για την υπομονή και την καθοδήγησή του καθ' όλη τη διάρκεια αυτής της εργασίας, όπως επίσης και για την εμπιστοσύνη που μου έδειξε δίνοντάς μου την δυνατότητα να εξελιχθώ τόσο σε προσωπικό όσο και σε γνωστικό επίπεδο. Εξίσου σημαντικό ρόλο επιτέλεσε ο ακαδημαϊκός σύμβουλός μου καθηγητής κ. Ιωάννης Τζίτζικας, όπου ως αρωγός βρίσκονταν πρόθυμος να με κατευθύνει με τις ουσιαστικές του συμβουλές. Επιπλέον θα ήθελα να ευχαριστήσω τα μέλη της ερευνητικής ομάδας MXM και ιδιαίτερα τον επικεφαλή αυτής, τον καθηγητή κ. Ιωάννη Τσαμαρδίνο για τις συζητήσεις μας και τα πολύτιμα σχόλιά τους. Δεν θα μπορούσα να μην εκφράσω την ευγνωμοσύνη μου στον επικεφαλή επιστήμονα κ. Ερικ Σιμον, ο οποίος επισφράγισε τη δυνατότητά μου να βιώσω όλα τα οφέλη και τις υποχρεώσεις που έχει ένας εργαζόμενος σε μεγάλη εταιρία. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αστείρευτη αγάπη και υποστήριξή τους. Καθώς η ολοκλήρωση αυτής της εργασίας συνάμα σηματοδοτεί την ολοκλήρωση μιας πορείας δύο ετών επιμονής, τόλμης, θέλησης και υπομονής, τι καλλίτερο από το να κλείσω ετούτο το κεφάλαιο με ένα απόφθεγμα του μεγαλύτερου Κρητικού και Έλληνα συγγραφέα Νίκου Καζαντζάκη *Μην καταδέχεσαι να ρωτάς: «Θα νικήσουμε· Θα νικηθούμε·» Πολέμα!*

στους γονείς μου

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With sensors pervading our everyday lives, we are seeing an exponential increase in the availability of streaming data. It is remarkable that in the current era of smart and connected devices there are more than 12 billion of Internet of Things (IoT), and it is estimated to exceed 25 billion by 2025 [48]. That estimation looks achievable considering the release of 5G network, which will enable even more critical IoT applications including remote healthcare system (clinical remote monitoring and assisted living), traffic and industrial control (Drone/Robot/Vehicle), remote control of heavy machinery in hazardous environments, thereby improving worker safety, and even remote surgery [13]. In general, the various applications opportunities enabled by the IoT are countless and its full potential will only be realized by ensuring that more smart devices are connected through the Internet.

Analyzing data as they are generated in streams provides valuable insights in several application settings. In particular, online detection of anomalies is important analytic task in several use cases such as preventative maintenance, fraud prevention, intrusion detection, persons' or systems' health monitoring, etc. An anomaly is an observation that deviate so significantly from other observations as to arouse suspicion that it was generated by different mechanism [29]. For example, an unusual high amount of money spent from a credit card or amount of traffic generated in a router are considered as anomalies.

Three types of anomalies [12] are widely studied in the literature [53], [30]: Point, Contextual and Collective. A *Point*, also called global, anomaly is a data point that lies far way from the remaining points in a dataset. A *Contextual* anomaly, is a data point that deviates from the rest of the data points in a specific context. Note that that same data point may not be considered as an anomaly if it occurs in a different context. A *Collective* anomaly is a set of data points that as a collection deviates significantly from the entire dataset. Note that individual data points of this collection are not necessarily anomalous in either a contextual or global sense. Point anomalies can be found in set oriented datasets while contextual and collective anomalies on sequence oriented datasets, in which the ordering of data points matters.

In this thesis we are interested in unsupervised methods for detecting anomalies in multivariate datasets that do not require to label data as normal (or abnormal). This is motivated by the fact that in the streaming analytics applications mentioned above it is either expensive or infeasible to obtain such labels by human experts. Unsupervised point anomaly detection methods proposed by machine learning and database communities, can be grouped in two main categories [37]: Statistical and Proximity based. *Statistical* methods fit a model of normality using data sampling. Data points that do not fit this model are considered to be anomalous. Typical examples of non-parametric statistical methods are tree-based detectors such as iForest [45], HST [66] and RRCF [26]. Proximity methods on the other hand, rely on the distances of observations in a metric space. Typical examples of unsupervised methods are KNN_W [56] and MCOD [34] or density-based detectors such as LOF [10]. Note that iForest, KNN_W, and LOF are offline detectors while HST, RRCF and MCOD online detectors.

As anomaly detection has been actively area of research for decades, several experimental studies have been conducted to benchmark the effectiveness of detectors using well-known evaluation protocols and metrics [50], [18], [11], [2], [21], [1] and [68]. In their majority, they focus on the effectiveness of offline detectors using multivariate datasets contaminated by point anomalies. These studies lead to the following main recommendations.

- Use Isolation-based methods, such as iForest, because (i) they can scale up to large datasets up to 500 dimensions, (ii) they have low space complexity with linear sample and ensemble size, (iii) they have known behaviors under different data characteristics and they can deal with different types of anomalies and (iv) they are not too sensitive to their hyper-parameter tuning.

- Use Proximity-based methods, such as LOF or KNN_W for detecting highly clustered anomalies in medium sized datasets. However, the run-time of these algorithms is slower, their space complexity is high while their hyper-parameters like min-neighbors is sensitive to the data size.

For the sake of completeness, we would like to also mention the Numenta benchmark (NB) of online anomaly detection over univariate time series [3] contaminated by contextual outliers. Unfortunately, NB focus explicitly on time series and do not provide insights for online detectors applicable to multivariate sequential data.

The aforementioned studies provide valuable insights regarding essentially offline anomaly detectors. In this respect, several questions regarding online detectors over multivariate data streams have not yet addressed in the literature. First, it is left open to what extent online detectors approximate the effectiveness of offline detectors and under which conditions (e.g., number of outliers in data streams, number of features irrelevant to the outliers). Second, little evidence is provided regarding which update mechanisms of detectors models (i.e., micro-clusters or random trees) are more suited to capture outliers in a data stream and

with what computational cost. Third, we are interested to know how sensitive online detectors are to their hyper-parameters[1] (i.e., model or window specific).

More precisely, no previous experimental study has compared using the same datasets tree-based (i.e., HST/F, RRCF, iForest) with nearest-neighbor based detectors (MCOD, LOF, KNN) in both batch and streaming modes. Moreover, most of the related benchmarks consider outliers defined over the full feature space of the datasets. No experiments have been reported regarding the effectiveness of online detectors on outliers visible only on a subset of the feature space. Last but not least, unsupervised anomaly detectors have been executed in reported experiments using the default hyper-parameter configurations provided by the original authors of the algorithms. This choice compromises the reported results as default hyper-parameters are agnostic of the actual data characteristics especially for proximity-based detectors.

To address the missing insights, we introduce a general framework for benchmarking online and offline anomaly detectors over multivariate streaming data, as well as for ranking their effectiveness in the same collection of datasets. It relies on a correct and fair benchmarking methodology that guides the design of experiments by taking into account both the datasets and detectors characteristics. We pay particular attention to the contamination of datasets with real or synthetic outliers that are visible only to a subset of the original feature space. Such *subspace outliers* are quite frequent in scientific and industrial monitoring applications using sensors or IoT devices. The introduced benchmark framework has been developed in the context of my internship at SAP France [62], and extends an existing platform for benchmarking offline anomaly detectors [51].

The main contribution of this Master Thesis, is to bring light to the following questions:

Section 5.1. *To what extent online detectors approximate the effectiveness of offline detectors on real datasets contaminated by subspace or fullspace outliers?* The gist of our findings is that online detectors are quite effective in detecting *increasing rates of subspace outliers* in data streams. In particular, proximity-based detectors like MCOD and tree-based detectors like RRCF prove to be more efficient than their offline counterparts (LOF and iForest) due to the heavy reconstruction of their models as new data points arising in a stream. For *fullspace outliers* less computationally costly mechanisms that update only mass profile counters employed by HSTF seems to sufficiently work. In addition, RRCF and HSTF are best for detecting *outliers with extreme feature values* while MCOD when adequately tuned can overcome outliers *swamping and masking* problems even in high dimensional datasets [76].

Section 5.2. *How robust are online and offline detectors against increasing data and subspace dimensionality?* Online detectors (HST/F, RRCF and MCOD)

[1]Recall that hyper-parameter is an important parameter which cannot be directly estimated from the data [38].

are more robust than offline (LOF, KNN and iForest) detectors against an increasing *ratio of features which are irrelevant* to the subspace outliers contaminating a dataset. In particular, MCOD well-tuned to the characteristics of the data stream archives an optimal effectiveness for every data and subspace dimensionality. Online detectors updating the entire structure of their model (MCOD and RRCF) are more robust than detectors updating only the mass profiles of tree leaves (HST/F) against *subspace outliers of increasing dimensionality*. In particular, RRCF exhibits the best effectiveness among all tree-based detectors.

Section 5.3. *How sensitive online detectors are to their hyper-parameters?* Online detectors are affected by both model (e.g., min-neighbors, max-distance, trees, tree-depth) and window (e.g., size and slide) hyper-parameters. Model hyper-parameters of proximity-based detectors like MCOD (min-neighbors and max-distance) prove to be harder to optimize than of tree-based detectors like HST/F and RRCF (number and depth of trees). In addition, MCOD is very sensitive to the window hyper-parameters, since it is based on the window slide to forget past points from the models build. Moreover, the introduction to HST of forgetting mechanism (HSTF) not only improves its effectiveness, but also its sensitivity w.r.t. the tree-depth hyper-parameter.

The presentation of the thesis is organized as it follows. In Chapter 2, we describe and discuss our selection of anomaly detectors. In Chapter 3, we introduce our general benchmarking environment including details of the platform deployment, datasets, detector design choices and the hard/software characteristics of the machine under which benchmark took place. In Chapter 4, we introduce the benchmark methodology representing the manual of our benchmark pipeline. In Chapter 5, the results and delightful insights of our experiments, are summarized. Finally conclusions and future work are discussed in Chapter 6.

Chapter 2

Anomaly Detection Algorithms

In this chapter, we present several anomaly/outlier detection algorithms, simply called detectors, that employ either binary or continuous scores to distinguish abnormal from normal data points. Given the scarcity of labeled data in many real applications (e.g., Internet of Things), we focus on online state-of-the-art algorithms as well as the offline ones that they inspired by, relying on unsupervised outlierness criteria, such as distance, density or tree-based. Other outlierness criteria that mainly focus on anomaly detection over time series data [27] are out of the scope of this work.

From online distance-based detectors, MCOD is considered the best in terms of speed and memory as experimentally proved in the well-known benchmark [69]. From online tree-based detectors, HST is considered the best in terms of speed [66] and RRCF is considered the best in terms of effectiveness [26] as shown in the results of their original work. To the best of our knowledge, our work is the first ever benchmark over these online tree-based detectors.

Object	**L/W**	**NS**	**S**
penny	1.5	3	true
dime	1	3	true
knob	1	4	true
eraser	2.75	6	true
box	1	6	true
block	1.6	6	true
screw	6	3	true
battery	5	3	true
key	4.25	3	false
bead	1	2	true

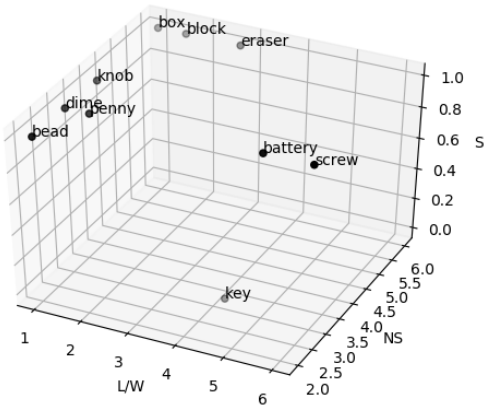

Table 2.1: Example dataset of 10 data points, with 9 normal and 1 anomalous.

From offline nearest-neighbor detectors, KNN_W is a distance-based and LOF

is a density-based who inspired MCOD. From offline tree-based detectors, iForest is considered the one who inspired both HST and RRCF. These offline detectors have been extensively evaluated and compared in the well-known benchmarks [17], [18] and [11].

In Table 2.1 we present ten 3D data points, where each data point represent an object characterized by its *Length/Width*, *Number of Surfaces* and *Smoothness*. A human expert used his prior knowledge to cluster these objects into three clusters; Skinny {battery, screw}, Corners {box, block, eraser} and Round {Knob, dime, bead, penny}. According to that expert, the object *key* is closer to the Skinny cluster, but still in distance since its the only not smooth object. Therefore, all objects are inliers except *key* which is a real outlier.

In the following sections, we present the concept of anomaly detection from the algorithmic perspective and we make use of the above dataset as our running example, in which detectors quote their own belief of anomalies over these objects.

2.1 Categorization of Anomaly Detection

2.1.1 Labels

There are three general methodologies of a detector according to the use of labeled data for training the algorithms [30]; Supervised, Semi-supervised and Unsupervised learning. *Supervised* algorithms involve training a model with labeled data regarding both inliers and outliers. *Semi-supervised* algorithms involve an interactive with an oracle training of a model with a small amount of labeled data (e.g., of inliers) bolstering a larger set of unlabeled data. *Unsupervised* algorithms detect anomalies without the use of preassigned labels for both inliers and outliers. Supervised methodologies are out of the scope of this work.

2.1.2 Modes

Another important issue for anomaly detection is the mode of a detector, which is defined according to its outlier constraints in train phase [31]; Novelty and Outlier. *Novelty* involves training a model in a no polluted by outliers train phase. *Outlier* involves training a model in a polluted by outliers train phase.

2.1.3 Types

There are two types of detector according to its model characteristics [27]; Offline and Online. *Offline* called a detector which maintains a static model over data. In other words, its model never gets updated. *Online* called a detector which has a dynamic model over data. In other words, its model gets updated periodically. Moreover, an offline detector processes all data points at once, when an online detector process points over windows.

2.1.4 Windows

The online type of detectors need a windowing sampling technique to process the stream data. Count-based windows is a sampling technique refers to splitting data into chunks, based on the order that data arrives [4]. A count-based window, simply called window, has two hyper-parameters; *size* and *slide*. The window size indicates the total number of data points in a window, where all windows must have the same size except probably the last one. The window slide indicates the step size of the window over data, that is the number of data points that a window will be shifted. There are two types of windows according to its hyper-parameters [43]; Sliding and Tumbling. *Sliding* is a window that slides across a data stream according to a specific interval, and therefore may contain overlapping data; a point bay belong to more than one sliding window. *Tumbling* is a window of data grouped in a non-overlapping interval, and therefore a data point belongs to only one window (i.e., window size = window slide).

2.1.5 Output

Detectors may spot either a binary outcome (anomaly or not) or a real-valued score 8 [36]. The binary outcome is valued in a range of exactly two values; 0 to indicate inlier and 1 to indicate outlier. The continuous score is valued in a range of a closed $[0, 1]$ or an open bound $(-\infty, +\infty)$ interval; the higher the value, the higher the degree of outlierness.

2.2 Offline Detectors: KNN_W, LOF, iForest

There is a plethora of offline detectors using different outlines criteria based on distance, density or random trees.

2.2.1 Weighted K-Nearest Neighbor (KNN_W)

The Weighted KNN (KNN_W) is an unsupervised score based variation of the distance based K Nearest Neighbors (KNN) supervised classifier [56], which computes the distance of a given data point with respect to the rest points. It considers as outlier, points that are in substantially higher distance than the rest points (i.e., they have few neighbors) [20].

In the training phase it constructs a matrix of distances, called *MD*. The *MD* is a 2D array, where each row represents a data point and each column represent the distance from a nearest neighbor.

The prediction phase makes use the MD to assign a real-valued score for each (row) data point. The score of a data point p is computed as the maximum distance *dist* over its K (columns) nearest neighbors n, given by the Equation 2.1. The higher the score, the more probable the point to be an outlier.

$$score(p) = max(dist(p, n(q))) : 1 \leq q \leq K \tag{2.1}$$

The training and predict phase of KNN_W take place on the same set of data. The detector originally designed to run in *outlier* mode.

The KNN_W has two hyper-parameters; K and *Metric*. The K is the number of nearest neighbors. The *Metric* is the name of the distance metric through which calculates the distance from its neighbors.

The main concern with optimizing KNN_W is to select the right number of neighbors K to be considered.

2.2.2 Example

Let us assume the ten objects of the Table 2.1 and the KNN_W detector with K = 3 and Metric = Euclidean. The KNN_W make use all of the 10 objects to first train its model and then score them using that model.

Object	**K=1**	**K=2**	**K=3**
penny	0.0	0.5	1.12
dime	0.0	0.5	1.0
knob	0.0	1.0	1.12
eraser	0.0	1.15	1.75
box	0.0	0.60	1.75
block	0.0	0.60	1.15
screw	0.0	1.0	2.02
battery	0.0	1.0	1.25
key	0.0	1.25	2.02
bead	0.0	1.0	1.12

Table 2.2: The model of KNN_W over the objects of Table 2.1.

Table 2.2 illustrates the constructed model of KNN_W as a 10x3 MD, where each row represent an object and each column represent the Euclidean distance from one out of three nearest neighbors.

Thereafter, the model is used to assign an anomaly score for each of the ten objects, as the maximum Euclidean distance per row. The descending order of the anomaly scored objects is as it follows: {key:2.02, screw:2.02, eraser:1.75, box:1.75, battery: 1.25, block:1.15, knob:1.12, penny:1.12, bead:1.12, dime:1.0}.

According to authors in [56], a KNN_W model has quadratic training complexity O(n*n) and linear predict complexity O(n). The description of this parameter is shown in Table 2.3.

Parameters	**Description**
n	Number of data points

Table 2.3: The complexity parameters of the KNN_W.

2.2.3 Local Outlier Factor (LOF)

The Local Outlier Factor (LOF) is an unsupervised density based outlier detector, which computes the local density deviation of a given data point with respect to its neighbors. It considers as outlier, points that have substantially lower density than its neighbors [10].

In the training phase it constructs a matrix of densities, called *MD*. The *MD* is a 2D array, where each row represents a data point and each column represent the local reachability density (LRD) ratio of that point to a nearest neighbor point. To get the LRD of a data point a, we first calculate the reachability density of a to all its K neighbors and take the average of that number. The LRD is then simply the inverse of that average. By intuition the local reachability density tells how far we have to travel from our point to reach the next point or cluster of points. The lower it is, the less dense it is, the longer we have to travel. The formula of LRD is given in Equation 2.2.

$$lrd(a) = 1/(sum(rd(a,n))/K) \tag{2.2}$$

The reachability distance is simply the maximum of the actual distance of two points and the k-distance of the second point. Basically if a point a is within the K neighbors of point b, the rd(a,b) will be the k-distance of b. Otherwise, it will be the actual distance of a and b. The k-distance is the actual distance of a point to its K_{th} neighbor. The formula or RD is given in Equation 2.3.

$$rd(a,b) = max\{kd(b), d(a,b)\} \tag{2.3}$$

The prediction phase makes use the MD to assign a real-valued score for each (row) data point. The score of a data point p is computed as the average of the LRD ratios over its K (columns) nearest neighbors, given by the Equation 2.4. The higher than 1 the score is, the more probable the point to be an outlier. An approximately to 1 score indicates inlier.

$$score(p) = sum(lrd_{ratio}(n(q)))/K : 1 \leq q \leq K \tag{2.4}$$

The training and predict phase of LOF take place on the same set of data. The detector originally designed to run in *outlier* mode.

The LOF has two hyper-parameters; K and *Metric*. The K is the number of nearest neighbors. The *Metric* is the name of the distance metric through which calculates the distance from its neighbors.

The main concern with optimizing LOF is to select the right number of neighbors K to be considered.

2.2.3.1 Example

Let us assume the ten objects of the Table 2.1 and the LOF detector with K = 3 and Metric = Euclidean. The LOF make use all of the 10 objects to first train its model and then score them using that model.

Object	**K=1**	**K=2**	**K=3**
penny	0.98	1.09	1.21
dime	1.02	1.12	1.24
knob	1.22	1.20	0.91
eraser	1.47	1.0	1.0
box	1.47	1.0	1.0
block	1.47	1.0	1.0
screw	1.61	1.0	1.0
battery	2.17	1.0	1.0
key	2.17	1.0	1.0
bead	0.82	0.81	0.90

Table 2.4: The model of LOF over the objects of Table 2.1.

Table 2.4 shows the constructed model of LOF as a 10x3 MD, where each row represent an object and each column the LRD ratio of one out of three nearest neighbors.

Thereafter, the model is used to assign an anomaly score for each of the ten objects, as the average LRD ratio per row. The descending order of the anomaly scored objects is as it follows: {key:1.39, battery:1.39, knob:1.20, screw:1.20, eraser:1.16, box:1.16, block:1.16, dime:1.12, penny:1.10, bead:0.84}.

According to authors in [10], a LOF model has quadratic training complexity O(n*n) and linear predict complexity O(n). The description of this parameter is shown in Table 2.5.

Parameters	Description
n	Number of data points

Table 2.5: The complexity parameters of the LOF.

2.2.4 Isolation Forest (iForest)

The Isolation Forest (iForest) is an unsupervised tree-based ensemble outlier detector, which isolates outliers directly without relying on an explicit distance or density metric. It considers as outlier, points that are in substantially shallower average trees depth [45].

In the training phase it constructs a forest of a *Trees* number of isolation trees, called *IF*, using bootstrapping of size *Max Samples*. An isolation tree, called *IT*, is a collection of internal and external nodes stored in a full binary tree. Each internal node stores a random and uniformly selected *feature* and a random and uniformly selected *value* from its value range. Each external node stores the number of data points that fells into, called *size*, and the number of internal nodes in the path from root to this node, called *actual depth*, which is limited up to *Max Depth*.

The prediction phase makes use the *IF* model to assign a real-valued score for each data point. The score of a data point p is computed by the formula of Equation 2.5. The closer to 1 the score is, the more probable the point to be an outlier. The much smaller than 0.5 score indicates inlier.

$$score(p, n) = 2^{-E(h(p))/c(n)} \tag{2.5}$$

In that scoring formula, n is the *max samples* size, *E(h(p))* is the average of *h(p)* and *c(n)* is used for normalization, given by the Equation 2.6.

$$c(n) = 2 * (ln(n-1) + 0.5772156649) - 2 * ((n-1)/n \tag{2.6}$$

The *h(p)* is derived as the *actual depth* of the external node that p is terminated to. When p is terminated to an external node, in which the *size* is larger than one, the return value is plus an adjustment c(size).

The training and predict phase of iForest can take place either on the same either on different set of data. In case of different sets, a train set used for the training phase and a test set used for the predict phase, where each set is a collection of data points. The detector originally designed to run in *outlier* mode.

The iForest has three hyper-parameters; *Trees*, *Max Samples* and *Max Depth*. The *Trees* is the number of tree detectors. The *Max Samples* is the number of data points that are used to build a tree model. The *Max Depth* is the maximum depth that an external node can reach.

The main concern with optimizing a non-deterministic iForest model is to select a correct value for *Max Depth* and *Trees*. Authors are aware of this, so they

suggested the $ceil(log_2(n))$ as a dynamic max depth of a tree and a high number of tree detectors.

2.2.4.1 Example

Let us assume the ten objects of the Table 2.1 and the iForest detector with Trees = 1, Max Samples = 5 and Max depth = 2. The iForest make use all of the 10 objects to first train its model on a sub sample of 5 objects and then score all of the 10 objects using that model.

Object	L/W	NS	S
dime	1	3	true
knob	1	4	true
box	1	6	true
battery	5	3	true
bead	1	2	true

Table 2.6: The sub sample of 5 objects.

In Table 2.6 we show the sub sample of five objects selected via bootstrapping on the full data set defined in Table 2.1.

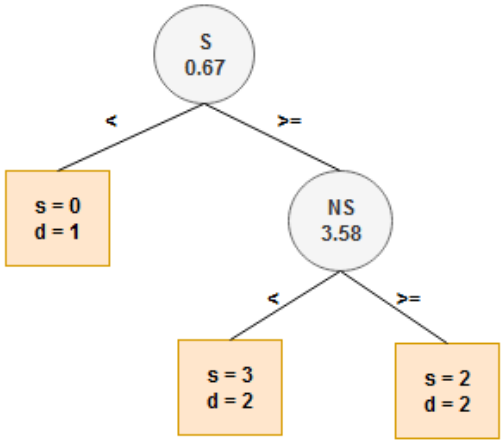

Figure 2.1: The model of iForest over the objects in Table 2.6.

Figure 2.1 depicts the constructed IT, by recursively partitioning the data space defined in Table 2.6. On the first partition, the feature S selected with value 0.67 and on the second partition, the feature NS selected with value 3.58. In the external node of depth 1, no object fell into. In the external nodes of depth 2,

three objects {dime, battery, bead} fell into the left node and two object {knob, box} fell into the right node.

Thereafter, the model is used to assign an anomaly score for each of the ten objects of Table 2.1, using c(5) = 2.3270 and c(3) = 1.2074. If we take a look to the object *penny*, it terminates to the external node of depth 2 and size 3. So $score(penny) = 2^{-(2+c(3))/c(5)} = 2^{-(2+1.2074)/2.3270} = 2^{-1.3783} = 0.38$. The score of the rest objects computed in the same way. The descending order of the anomaly scored objects is as it follows: {key:0.74, knob:0.53, eraser: 0.53, box: 0.53, block: 0.53, screw: 0.38, battery: 0.38, penny: 0.38, dime: 0.38, bead: 0.38}

According to authors in [45], an iForest model has linear training complexity O(t*m*h) and linear predict complexity O(n*t*h). The description of the parameters is shown in Table 2.7.

Parameters	**Description**
t	Number of trees
m	Number of sub sample
h	Max height of a tree

Table 2.7: The complexity parameters of the iForest.

2.3 Online Detectors: MCOD, HST/F, RRCF

There are several online detectors using different outlierness criteria. In this work, we focus on distance and random tree based online detectors. In opposed to offline, online detectors use a window technique to process the data points over a stream such as sliding and tumbling windows.

In Table 2.8 we make use the objects of Table 2.1 in streaming manner and therefore we partition objects into sliding and tumbling windows. In our example, objects grouped in sliding windows of *size* = 8 and *slide* = 2 as well as tumbling windows of *size* = 5. So, online detectors will be applied on two sliding (S0, S1) or two tumbling (T0, T1) windows.

2.3.1 Micro-cluster Outlier Detector (MCOD)

The Micro Cluster Outlier Detection (MCOD) is a distance based detector, that captures the data neighboring regions over a stream using micro-clusters to eliminate the need for range queries; distance re-calculation between all points would be very expensive, especially when carried out on large datasets. MCOD processes a stream using sliding windows, through which reconstructs its model [34].

In the training phase it constructs a number micro-cluster lists and a proceeding list. Micro-cluster is a list, called *MC*, composed of no less than *K+1* data points,

Object	Sliding W	Tumbling W
penny	0	0
dime	0	0
knob	0,1	0
eraser	0,1	0
box	0,1	0
block	0,1	1
screw	0,1	1
battery	0,1	1
key	1	1
bead	1	1

Table 2.8: The 10 objects of the Table 2.1 in streaming manner.

where each point can only belong to one MC. According to the triangular inequality in metric space, the distance between every pair of data points in a *MC* is no be greater than *R*. The rest points are stored in proceeding list called *PD*.

The prediction phase make use of the MC and PD lists to assign a score for each data point. A data point p is an outlier when its actual distance from the center of any MC is higher than *R/2* or the potential MC that belongs to has less than *K+1* points. The binary score is calculated by the Equation 2.7. The score 1 indicates outliers and score 0 indicates inliers.

$$score(p) = (dist(p, MC) < \frac{R}{2} \, or \, cardinal(MC(p, PD)) < (K + 1)) \, ? \, 1 \, : \, 0 \quad (2.7)$$

In the update phase points can be added and/or removed from MC and PD lists. When a window slides, a number of data points gets expired. That may dissolve a MC, with the remaining points to be stored in PD list. In every window, the points in PD list are processed as new data points. A new point can: i) be enough to form a new MC when there at least K more data points within R distance in PD list. ii) be added to the closest MC when the distance from its center is within R/2. iii) remain to the PD list.

The training, predict and update phase of the MCOD take place on the same set of data, which is a collection of sliding windows. The points of the sliding windows are first used to train/update the model and then to be scored by that trained/updated model. The detector originally designed to run in *outlier* mode.

The MCOD has two hyper-parameters; *Max distance*, *Min neighbors* and *Metric*. The *Max Distance* (R) corresponds to the actual radius R of a micro cluster. The *Min Neighbors* (K) correspond to the minimum number of neighbors K of data points that a cluster need to be constructed. *Metric*, is the name of the distance metric through which calculates the distances. MCOD is additionally get affected by the hyper-parameters of sliding windows.

MCOD is a very difficult detector to optimize, because is highly dependent on the data characteristics. A wrong set of hyper-parameters can cause many false positives/negatives. MCOD with high R on a dataset with small feature value range, will classify all points as inliers. On the other hand, MCOD with low R on a dataset with high feature value range, will classify all points as outliers. In addition, MCOD with a very high K may not capture a small cluster of data and on the other hand with a very low K may capture many false positives.

2.3.1.1 Example

Let us assume the ten objects of the Table 2.8 and MCOD detector with Min Neighbors = 1, Max Distance = 2.0 and Metric = Euclidean. The MCOD make use the S0 window to first train its model then predicts using that model and thereafter the S1 window to first update its model and then predict using the updated model.

Object	**List Type**
penny	MC1
dime	MC1
knob	MC1
eraser	MC2
box	MC2
block	MC2
screw	MC3
battery	MC3

Table 2.9: The model of MCOD over the S0 window objects in Table 2.8.

We compute the Euclidean distance between all objects. Table 2.9 presents the constructed model of MCOD as the lists MC1, MC2 and MC3. We observe that there are no objects in PD list, and therefore all the eight objects distributed in MC clusters. The MC1 constructed, because three objects {penny, dime, knob} found in no more 2.0 distance between them. The MC2 and MC3 constructed for the same reasons.

Thereafter, the model is used to assign an anomaly score for each object of this window. All the objects of PD list scored by 1 and the rest by 0. The descending order of the anomaly scored objects is as it follows: {screw:0, battery:0 penny:0, dime:0, knob:0, eraser:0, box:0, block:0}.

Object	**List Type**
knob	MC1
eraser	MC2
box	MC2
block	MC2
screw	MC3
battery	MC3
key	PD
bead	MC1

Table 2.10: The updated MCOD model over the S1 window objects in Table 2.8.

In Table 2.10 we show the re-constructed model of MCOD as the lists MC1, MC2, MC3 and PD. We observe that two objects {penny, dime} expired from MC1, but the presence of the new object {bead} reconstructs the dissolved MC1. The MC2 and MC3 lists remains the same. The new object {key} stored in PD list, because in its closest cluster MC3 the object {eraser} is in distance 2.1 which is higher than the 2.0.

Thereafter, the updated model is used to assign an anomaly score for each object of this window. The descending order of the anomaly scored objects is as it follows: {key:1, screw:0, battery:0, bead:0, knob:0, eraser:0, box:0, block:0}.

According to authors in [69], an MCOD model has linear time complexity O((1-c)*w*log((1-c)*w)+k*w*log(k))) and linear space complexity O(c*w+(1-c)*k*w). Therefore, MCOD is faster than LOF and KNN_W, but still quite slow. The description of these parameters is shown in Table 2.11.

Parameters	**Description**
c	Number of classes
k	the number of nearest neighbors
w	The number of data points in a window

Table 2.11: The complexity parameters of the MCOD.

2.3.1.2 Real-valued Scoring Function

MCOD uses a binary outcome scoring function, resulting 0 for inliers and 1 for outliers. We investigate a way to assign a real valued score to the predicted outliers and therefore to result different outlier degrees. According to the original MCOD implementation [35], a prediction for a point p is computed by the formula: prediction(p) = succeeding neighbors(p) + preceding neighbors(p) < min neighbors ? 1 : 0. Neighbors of p, are the data points within distance R/2. The preceding neighbors of p, are the neighbors of p, P_p, that will expire before p, and the succeeding neighbors of p, S_p, that will persist during the entire lifetime of p. When a p predicted as an outlier p_o, we update its score from 1 into a real-valued score given by the formula 2.8.

$$\text{min neighbors - (succeeding neighbors}(p_o) + \text{preceding neighbors}(p_o)) \quad (2.8)$$

2.3.2 Half Space Trees (HST)

The Half Space Trees (HST) is an ensemble tree-based detector, that learns the sketch of a stream. It captures the density changes of adding a point in the sketch by profiling its mass. HST processes a stream using tumbling windows, through which update its model [66] without any time-fading mechanism (i.e., always remembers points).

In the training phase it constructs a forest of a *Trees* number of trees, called HST, using bootstrapping of size *Max Samples* and a workspace per tree. Workspace is a random perturbation of an original feature space. It consists of a set of work ranges; a different work range for every feature. A work range wr of a feature f is given by the formula in Equation 2.9.

$$wr(f) = Sf \pm 2 * max(Sf, fmax - Sf) \quad (2.9)$$

In that equation the *Sf* is a real number, randomly and uniformly selected from the actual value range [fmin, fmax] of the *f*. HST considers perfect binary trees, where each internal node has exactly two child nodes and all external nodes are in the same depth. An internal node stores a random and uniformly selected feature and the middle value of its work range. An external node stores the number of points that fells into, called *mass*, and the number of internal nodes in the path from root to this node, called *actual depth*, which is equal to *Max Depth*.

The prediction phase makes use the *HST* model to assign a score for each data point. The score of a point p is the sum of scores obtained from each T in the *HST*, given by the Equation 2.10.

$$score_{HST}(p) = sum(score(p, t)) : 1 \leq t \leq T \quad (2.10)$$

The score of p in a T is computed by the formula in Equation 2.11. In that equation, *Node* is the terminal node that p fells into and r, k correspond to its *mass* and *actual depth* respectively. The lower the mass profiles that a point ends up into, the lower the score that is assigned to and therefore the more probable to be an outlier. A high score indicates an inlier.

$$score(p, t) = Node.r * 2^{Node.k} \tag{2.11}$$

In the update phase, the data points in a tumbling window increase the mass profiles of each T in *HST*. The mass profile of an external node in T gets increased by one for every point that fells into. The HST model never forgets, and therefore mass profiles can never get reduced.

The training, predict and update phase take place on different set of data. A train set is used only for the training phase, and a test set is used for both predict and update phases. Each set is a collection of tumbling windows. In the test set, HST uses each tumbling window to first score all points and then to update its Model. The detector originally designed to run in *novelty* mode.

The HST has three hyper-parameters; *Trees*, *Max Depth* and *Max Samples*. The *Trees*, is the number of tree detectors. The *Max Depth* is the maximum level that an external node can reach. The *Max Samples*, is the number of points that are used to build a tree model. HST is additionally get affected by the hyper-parameters of tumbling windows.

HST is important to be optimized. HST with small number of trees can cause unstable effectiveness, because of the non-deterministic construction of the perfect binary trees. HST with high max depth can cause false positives, because of the high number of low mass external nodes.

2.3.2.1 Example

Let us assume the ten objects of the Table 2.8 and HST detector with Trees = 1, Max Samples = 3 and Max depth = 2. The HST make use of the T0 window to train its model on a sub sample of 3 objects and thereafter the T1 window to first score and then update its model.

Object	**L/W**	**NS**	**S**
penny	1.5	3	true
eraser	2.75	6	true
box	1	6	true

Table 2.12: The sub sample of 3 objects.

In Table 2.12 we present the three objects selected via bootstrapping on T0 window defined in Table 2.8.

Feature	WR Min	WR Max
L/W	-1.56	4.68
NS	-5.38	16.14
S	-1	3

Table 2.13: The workspace over the objects in Table 2.12.

Moreover, in Table 2.13 we present the features work range according to the sub sample of Table 2.12.

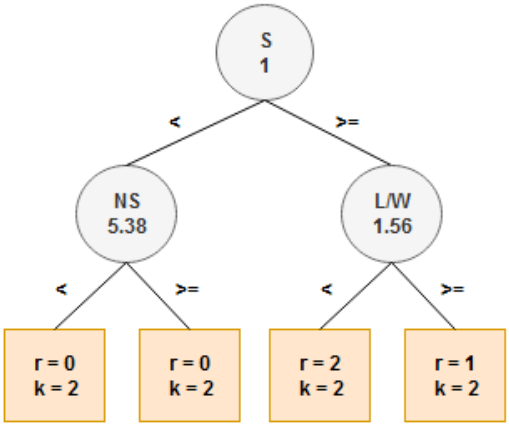

Figure 2.2: The model of HST over the objects in Table 2.12 using the workspace of Table 2.13.

Figure 2.2 illustrates the constructed model of HST, by recursively partitioning the data space defined in Table 2.12 using the workspace defined in Table 2.13. On the first partition, the feature S selected with value 1, on the second partition the feature NS selected with value 5.38 and on the third partition the feature L/W selected with value 1.56. All the external nodes are located in depth 2. No objects fell under the external nodes of the second partition. All the three objects fell under the external objects of the third partition; {penny, box} on the left and {eraser} on the right. Thereafter, the model is used to assign an anomaly score for each object in the T1 window of Table 2.8. If we take a look to the object *block*, it terminates to the depth (k) 2 and mass (r) 2 external node. So $score_{HST}(block) = score(block, T) = 1 * 2^2 = 4$. The score for the rest objects is computed in the same way. The descending order of anomaly scored objects is as it follows: {key:0, block:4, screw:4, battery:4, bead:8}.

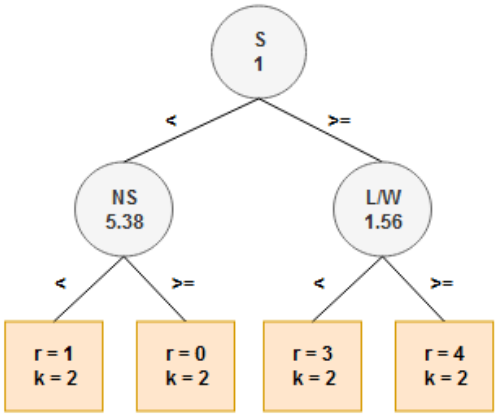

Figure 2.3: The updated HST model over the T1 window objects in Table 2.8.

In Figure 2.3 we illustrate the updated *HST* model, using the objects of the T1 window in Table 2.8. We observe that almost all the mass profiles have been updated. One object fell under the left external node of the second partition; {key}. Four objects fell under the external nodes of the third partition; {bead} on the left and {block, screw, battery} on the right.

According to to authors in [66], an HST model has linear training complexity $O(t*(2^{h+1}-1))$, that is because in the worst case scenario each data point end up to a different leaf. It has linear update complexity O(t * h * w), because every new point requires updating a mass profile of each tree. It has linear predict complexity O(w * t * h), because every new point traverses each tree up to an external node. Therefore, complexities are amortized (constant) when h, t and w are set. The description of these parameters is shown in Table 2.14.

Parameters	**Description**
t	Number of trees
h	Max height of a tree
w	Number of data points in a window
$2^{h+1}-1$	Number of nodes in a perfect binary tree

Table 2.14: The complexity parameters of the HST.

2.3.3 HST with Forgetting (HSTF)

The Half Space Trees with forgetting (HSTF) is our variation of the original HST [66], in which a forgetting mechanism has been integrated to the update phase. This forgetting mechanism is inspired by RRCF [26]. As we mentioned earlier

HST never decreases its mass profiles and therefore never forget a learned data point.

HSTF uses an aging mechanism to forget the oldest points. More specific when the overall mass of a tree in forest exceed a forget threshold, the mass profile of the K oldest external nodes in the tree will be decremented by one. An external node is called old when its mass profile has not been updated by the newest tumbling window. The K parameter correspond to the window size.

The phases of HST are all extended by HSTF. In the training phase, all the external nodes with zero mass profile are characterized as *old* leaves. The rest as *not old* leaves. In the updating phase, we have two categories of incremental update; insert and forget. The insertion of a new window of data points, will characterize as not old the external nodes that their mass profile has been updated and the rest external nodes as old. The forget update, decreases by one the (non-zero) mass profiles of K uniformly selected old leaves. Note that the forget of a point is only applied after a point insertion.

HSTF extends HST by one hyper-parameter; Forget Threshold. The *Forget Threshold* is the total mass profile of a tree that has to be reached (at least) so that the forgetting mechanism is activated.

In HSTF the value of Forget Threshold has to be correlated with the window size and therefore suggested values are; 125, 256, 512, 1024, 2048, 3072, etc. Moreover, we suggest the forget threshold to be the double (or more) than the window size, because the mass profiles must get increased enough before start decreasing them. It is also noticeable that the value of the forget threshold, adjusts the speed that HSTF gets adapted to the data changes. The lower the value of forget threshold, the faster a high mass profile can end up into a low mass profile.

HSTF has the exact same complexity as the HST. That is because the addition of a forgetting mechanism, does not affect the linear complexity over the different phases.

2.3.4 Robust Random Cut Forest (RRCF)

The Robust Random Cut Forest (RRCF) is an ensemble tree-based detector, that learns the sketch of a stream. It captures the differential effect of adding/removing a point from the sketch by profiling its collusive displacement. RRCF processes a stream using sliding windows, through which reconstruct its model including a time-decaying mechanism (i.e., forgetting points) [26].

In the training phase it constructs a forest of a *Trees* number of full binary trees, called *RRCF*, using bootstrapping of size *Max Samples*. A full binary tree, called T, is a collection of internal and external nodes, where each internal node has exactly two child nodes and the external nodes may be in different depth. An internal node stores a random and proportionally to the value range selected feature and a value selected randomly and uniformly from its value range. Additionally, each internal node has a bounding box; a data structure that stores the actual value range [fmin, fmax] of each feature f from the data points traversed by this internal

node. An external node stores a features set of the data point that fells into, the number of point's replicas and a unique index that indicates its age in tree. Note that, the maximum depth of a T depends on the number of the unique data points.

In the update phase, the data points in a sliding window reconstruct the structure of each T in *RRCF*. The tree reconstruction can be cause by two tasks; *Insert* a new point and *remove* the oldest point (FIFO). A point p traverses the internal nodes of a tree as long as is within their bounding boxes. If p reach an internal node Ni that does not fit into its bounding box, then a new constructed external node Np in which p terminates and the Ni will become the two child nodes of a new internal node that is thereafter connected to the rest tree. Else, p traverses successfully all internal nodes until reach to an external node Ne. If the feature set of p is the same with the one in Ne, then the number of replicas in Ne is increased by one. Else, a new constructed external node Np in which p terminates and Ne will be the new child nodes of a new internal node that is thereafter connected to the rest tree. Furthermore, the oldest external node of a tree gets removed, if the total number of external nodes in tree exceed the *Forget Threshold* size. When an external node Ne gets removed, then the internal parent node Ni also gets removed and replaced by its sibling node Ns.

The prediction phase makes use the *RRCF* model to assign a score for each data point. In order to compute the score of a point p, it is mandatory for p to already be inserted in *RRCF*. The score of a point p is the average of the collusive displacement (CoDisp) obtained from each T in *RRCF*, given by the Equation 2.12.

$$score_{RRCF}(p) = avg(coDisp(p,t)) : 1 \leq t \leq T \tag{2.12}$$

The collusive displacement of p in T is the maximal displacement given by the formula in Equation 2.13. The $disp(p,d)$ is the ratio between the total size of external nodes under the sibling and the current node that p traversed through in each depth d. The higher the collusive displacement of a point, the higher the score that is assigned to and therefore the more probable to be an outlier. A low score indicates inlier.

$$coDisp(p,t) = max(disp(p,d)) : 1 \leq d \leq D \tag{2.13}$$

The training, update and predict phase take place on different set of data. A train set is used only for the training phase, and a test set is used for both update and predict phases. Each set is a collection of sliding windows. In the test set, each point of the sliding windows, is used to first update the model and then to assign a score using that updated model. The detector originally designed to run in both *novelty* and *outlier* mode.

RRCF has three hyper-parameters: *Trees*, *Forget Threshold* and *Max Samples*. The *Trees*, is the number of tree detectors. The *Forget Threshold* is the total number of leaf nodes that has to be reached (at least) so that the forgetting mechanism is activated. The *Max Samples*, is the number of points that are used

to build a tree model. RRCF additionally get affected by the hyper-parameters of sliding windows.

RRCF is easier to get optimized than the rest online detectors. It is also noticeable that RRCF implementation can support up to *Max Leaves* number of leaf nodes depending on how powerful the setup is, 5.000 by default. That means that when the total number of leaf nodes exceed this limit, then the forgetting mechanism is being automatically activated to prevent data points leak.

2.3.4.1 Example

Let us assume the ten objects of the Table 2.8 and RRCF detector with Trees = 1, Max Samples = 5 and Forget Threshold = 6. The RRCF make use of the S0 window to train its model on a sub sample of 5 objects and thereafter the S1 window to both update its model and score on each data point.

Object	**L/W**	**NS**	**S**
knob	1	4	true
eraser	2.75	6	true
box	1	6	true
block	1.6	6	true
battery	5	3	true

Table 2.15: The sub sample of 5 objects.

In Table 2.15 we present the five objects selected via bootstrapping on S0 window defined in Table 2.8.

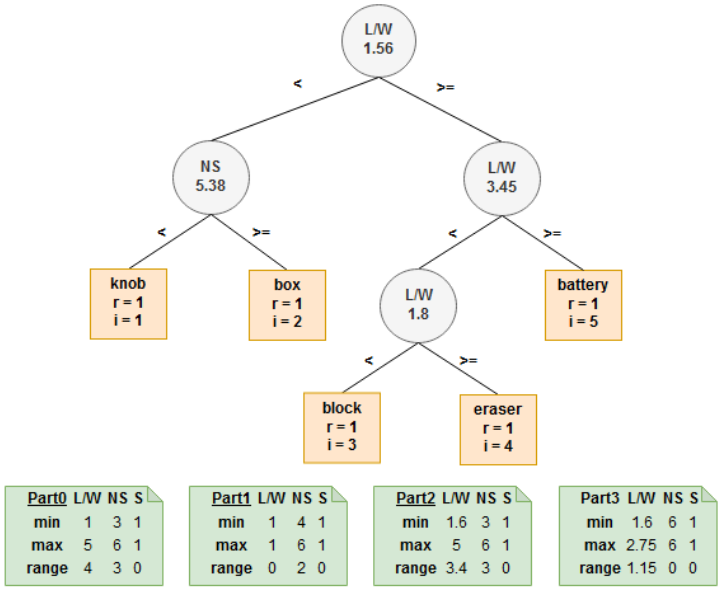

Figure 2.4: The model of RRCF over the objects in Table 2.15.

Figure 2.4 illustrates the constructed model of RRCF, by recursively partitioning the data space defined in Table 2.15. On the first partition (Part0), the feature L/W selected with value 1.56, because that feature has the larger value range according to its bounding box. In the same way, the feature NS with value 5.38 selected for the second partition (Part1). On the third partition (Part2), the feature L/W selected with value 3.45. The feature L/W with value 1.8 selected for the fourth partition (Part3). Furthermore, the external nodes are located in depth 2 and 3. Two objects fell under the external nodes of the second partition; {knob} on the left and {box} on the right. One object fell under the right external node of the third partition; {battery}. Two objects fell under the external nodes of the fourth partition; {block} on the left and {eraser} on the right. Note that r indicates the replicas of the object and i indicates its unique index.

Thereafter, we first update the *RRCF* model and then predict for each object in the S1 window of Table 2.8. If we take a look to the first object of the S1 window *knob*, we can observe that is already in the left external node of the second partition (part1). So, *knob* updates the *RRCF* model by increasing the number of replicas (r) of its external node from 1 to 2. The *knob* terminates in depth 2, so two displacements need to be computed; $disp(knob, 2) = 1/2 = 0.5$ and $disp(knob, 1) = 3/3 = 1$. The final score of *knob* is $score_{RRCF}(knob) = coDisp(knob, T) = 1$. In the same way each of the objects {eraser, box, block} increase the r value of the external node that terminates into and assigned by the scores; $score_{RRCF}(eraser) = 3/4 = 0.75$, $score_{RRCF}(box) = 4/4 = 1$ and $score_{RRCF}(block) = 2/2 = 1$.

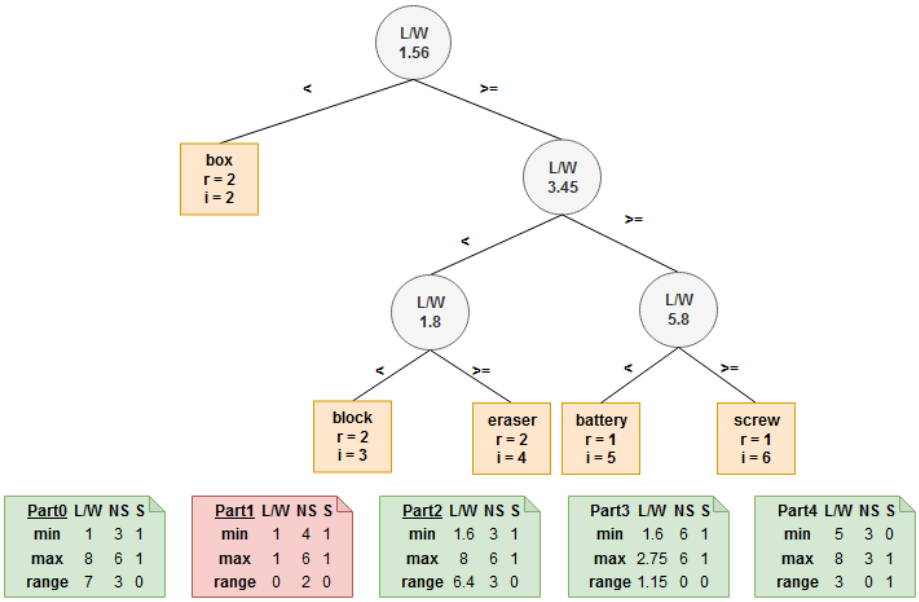

Figure 2.5: The updated RRCF model over the first five objects of S1 window.

The next object in the S1 window is the *screw*. The object *screw* is not already in the current model and therefore the construction of a new external node i=6 is necessary. We notice two things: a) it will terminate on the right side of the third partition under which external node i=5 located and b) it will exceed the L/W max bounding box value of the first (part0) and third (part2) partitions. So, the insertion of the object *screw* first extends the third's partition right side by a new fifth partition (part4) under which the external nodes i=5 and i=6 will co-located and then updates the bounding boxes of its path. The feature L/W with value 5.8 has been selected for the fifth partition. After the insertion, the total number of external nodes is equal to 6, and therefore the forgetting mechanism gets activated. In the deletion process, the i=1 external node is going to be deleted as well as its parent node, which is going to be replaced by its i=2 sibling external node. The updated RRCF model according to the first five objects {knob, eraser, box, block, screw} of the S1 window is illustrated in Figure 2.5. According to that model, the assigned score of screw is $score_{RRCF}(screw) = 4/2 = 2$.

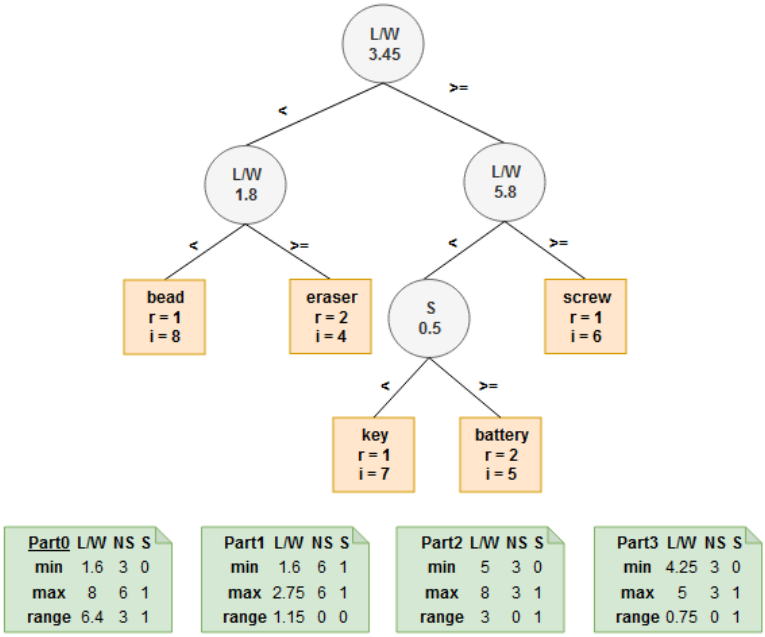

Figure 2.6: The final RRCF model over the S1 window objects in Table 2.8.

In the same way, each of the rest three objects of the S1 window {battery, key, bead}, update the model and assigned by a score; $score_{RRCF}(battery) = 4/3 = 1.33$, $score_{RRCF}(key) = 2/1 = 2$ and $score_{RRCF}(bead) = 2/1 = 2$. In total, the RRCF model has been updated eight times as much as the number of objects in S1 window. The final model is illustrated in Figure 2.6. The descending order of all the anomaly scored objects is as it follows: {key:2, screw:2, bead:2, battery:1.33, block:1, box:1, knob:1, eraser:0.75}.

According to authors in [26], an RRCF model has linear training complexity $O(t * (2 * n - 1))$, that is because in worst case scenario each data point end up to a different leaf. It has linear update complexity $O(t * \log_2(n))$, because in worst case scenario each new data point is a new leaf at the maximal height updating the entire sub tree. It has linear predict complexity $O(t * \log_2(n))$, because every new point traverses each tree up to an external node. Therefore, complexities are amortized when t and n are set. The description of these parameters is shown in Table 2.16.

2.3.4.2 Shingling Technique

RRCF uses a technique that turns out to be useful for detecting outliers over univariate streams, called *shingling*. This technique groups X points into a new shingle point of X features. We need two assumptions for the correctness of the technique; The shingle size is equal to the window size and the window slide must be one. A shingle encapsulates a typical shape of a curve; a departure from a

Parameters	Description
t	Number of trees
n	Maximum number of data points in a tree
2 * n - 1	Number of nodes of a perfect binary tree with n leaves
$\log_2(n)$	Minimal height of a perfect binary tree with n leaves

Table 2.16: The complexity parameters of the RRCF.

typical shape could be an outlier. A very small shingle size may catch naturally varying noise in the signal and trigger false alarms. On the other hand, a very large shingle size may increase the time it takes to find an alarm or even miss the alarm.

Let us assume a sliding window of size = shingle size = 4 and slide = 1. Also a stream S = {p1, p2, p3, p4 ,p5}. The first window = {p1, p2, p3, p4} used by the shingling technique to construct the first 4D shingle point. The second window = {p2, p3, p4, p5} used by the shingling technique to construct the second 4D shingle point, and so on. RRCF model gets applied on each shingle point.

2.4 Discussion

Online detectors are essentially incremental versions of offline detectors that are able to analyze data streams. According to their original authors, HST and RRCF are tree-based online detectors inspired by iForest while MCOD is a distance-based online detector inspired by KNN_W and LOF. These offline detectors are considered as *baseline* algorithms of the corresponding online detectors (see Table 2.17).

Online Detector	Model Baseline
HST/F	iForest
RRCF	iForest
MCOD	LOF, KNN_W

Table 2.17: The baseline models of online detectors.

Chapter 3

Benchmarking Environment

In this chapter, we propose a benchmarking framework general enough to experimentally evaluate and rank the effectiveness of online and offline detectors presented in Chapter 2. To assess the impact of various factors on effectiveness we use a dozens of real and synthetic datasets. These datasets exhibit different characteristics in terms of inlier and outlier points and are widely used in empirical evaluations of anomaly detection algorithms [72], [17], [18] and [11]. Moreover, we describe critical choices made in the implementation of the tree-based online algorithms (HST/F and RRCF) that we implemented from scratch as well as the third-party detectors (KNN_W, LOF, iForest) integrated in our platform. Finally, we present the machine characteristics under which all experiments have been conducted.

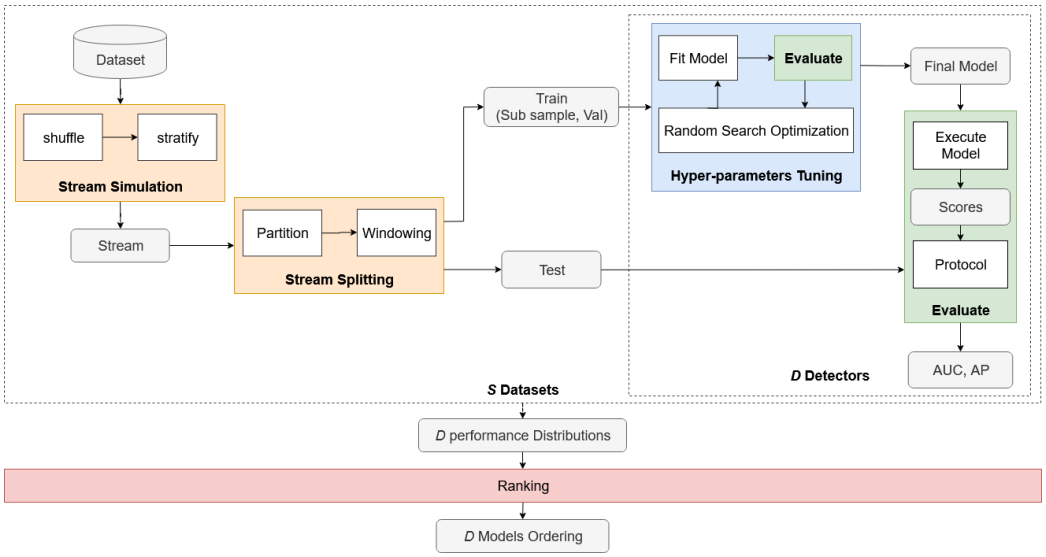

Figure 3.1: The high level architecture of our Experimental Environment.

The architecture of our experimental environment as a generalized benchmark

framework applicable to both online and offline detectors on any number of contaminated streams, is illustrated in Figure 3.1. The data components are colored grey and the functionalities are grouped by different colors denoting the different methodological questions that belong to (see Chapter 4); MQ1 is colored yellow, MQ2 is colored green, MQ3 is colored blue and MQ4 is colored red. The dotted rectangles, indicate iterations for the grouped components.

3.1 Benchmark Platform

Instead of implementing from scratch a new benchmark platform, we decided to use an already implemented platform and extend it according to our generalized architecture presented in Figure 3.1. There are two published implemented benchmark platforms that could fit to our needs under proper extensions, namely, Macrobase and Numenta NAB.

Macrobase originally introduced, by Stanford university, as an analytic monitoring engine, specialized for one task: finding and explaining unusual or interesting trends in data [5]. Later, a master student intern in SAP company, reviewed the Macrobase functionalities and developed a benchmarking platform to evaluate Percentile, MAD, FastMCD, LOF as well as iForest from offline and MCOD from online anomaly detection algorithms over ten real and synthetic datasets [51]. That Macrobase benchmark platform is published under Apache 2.0 license in GitHub [52], so it is easy for someone to use its source code or data as part of an open source project for commercial or private use.

Numenta Anomaly Benchmark (NAB) introduced, by Numenta company, as a benchmark platform for online anomaly detection in univariate time series data [39]. It consists of the benchmark itself and mostly real datasets. All NAB datasets are univariate contaminated by collective and contextual anomalies. The platform mainly focuses on the evaluation of anomaly detectors over time series. That is because, originally developed for evaluation of Numenta HTM algorithm [3]. NAB is published under AGPL 3.0 license in GitHub [40], so it is difficult for someone to use its source code or data as part of some closed source product or project with different license.

3.1.1 Extended Macrobase

To this direction we make use Macrobase instead of Numenta NAB, since the offline evaluation pipeline as well as logging functionalities are already implemented according to our needs (see [51]). During my internship in SAP company as a master student, we developed the online evaluation pipeline of Macrobase, including architecture extensions such as a streaming generator, window types as well as window manager illustrated in Figure 3.2 and online detectors (HST/F and RRCF) illustrated in Figure 3.3.

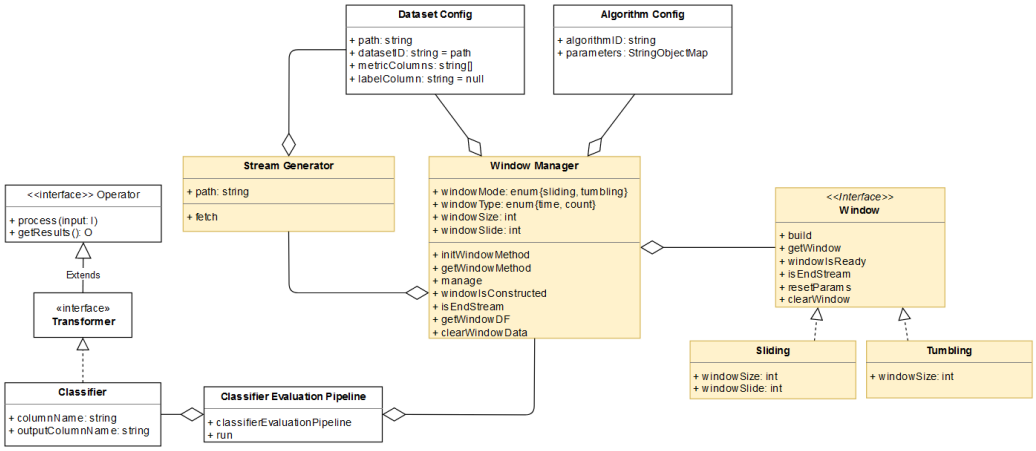

Figure 3.2: An extended by online functionalities part of Macrobase architecture presented in [51].

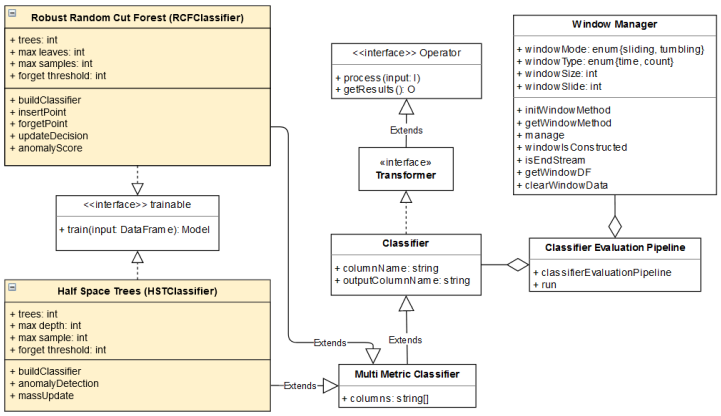

Figure 3.3: An extended by online detectors part of Macrobase architecture presented in [51].

During this internship, we developed an extended Macrobase platform which supports online and offline detector evaluation over real and synthetically contaminated stream data. Our extended version of Macrobase, the online detectors presented in Section 3.3 and the datasets presented in Section 3.2, are all publicly available in our GitHub repository [23] under Apache 2.0 license.

3.2 Datasets

We have used in our experiments in total 10 real and 7 synthetic datasets for either testing the sanity of the implemented online detectors (see Section 3.3) or to evaluated their effectiveness in conjunction with their counterpart offline

detectors (see Chapter 5). The bulk of our experimental evaluation lies on real data which in their majority outliers have been synthetically generated. Moreover, we used datasets with synthetically generated both inliers and outliers to simulate the effect of particular factors (i.e, dimentionality, outlier ratio, window size and speed, etc.) on the effectiveness of detectors.

There are four well-known ways to synthetically simulate anomalies, in a dataset that no expert already defined data points as real anomalies. In *Minority*, anomaly considered a data point that belong to a class with the smallest or a small number of data points. A *Statistical* anomaly, is generated according to a statistical distribution. In *Density*, anomaly is a data point thrown far away from the dense regions. An *Implanted* anomaly, is a data point in which a set of its values replaced by noise.

Besides that, the nature of an anomalous point may differ with respect to the gap between the cardinal of its full feature space and the actual space in which considered anomalous. *Fullspace* anomaly, is a data point that considered anomalous according to the values of its full feature set. Otherwise is *Subspace* anomaly. Our experimental interest focuses on datasets contaminated by subspace outliers, since detectors gets applied on full data dimensionality.

3.2.1 Real

Table 3.1 depicts the number of samples and features of the real world datasets used in our experiments. These datasets have been originally introduced and made freely available on UCI [19] and OpenML [70] repositories, for the problem of multi-class classification. The samples and features information in the original repositories, may slightly differ from the ones presented in our table. That is because, to evaluate a plethora of detectors we had to contaminate these datasets and therefore transform them from multi-class to binary-class.

To this direction we used the already contaminated versions of these real datasets, from open-source published repositories; GLOSS {Arrhythmia, Diabetes, Glass, Hypothyroid, Ionosphere and Pendigits} [65], ODDS {MNIST, Shuttle} [58], DATAHUB {Electricity} [47] and Numenta NAB {NYC Taxicab} [3]. Note that, datasets are not normalized and that in Electricity dataset the first ten thousand data points were enough.

In general, a dataset can be classified in one of the total four clusters according to its sample-feature high/low combinations. Empirically, as high we consider more than 5000 samples or 20 features. The dataset {MNIST} classified to the cluster of high both samples and features. The datasets {Glass, Diabetes} classified to the cluster of low both samples and features. In the cluster of high samples and low features classified the datasets {Pendigits, Electricity, NYC Taxicab}. The datasets {Arrhythmia, Hypothyroid, Ionosphere} classified to the cluster of low samples and high features. We observe that our selected datasets belong to all the possible combinations, which explains our interest on them.

We further investigate the contamination of these datasets in three directions;

Dataset	Samples	Features
Arrhythmia	452	279
Diabetes	768	8
Glass	214	9
Hypothyroid	3772	29
Ionosphere	351	34
Pendigits	10992	16
MNIST	7603	100
Electricity	10000	8
NYC Taxicab	10320	1
Shuttle	46464	9

Table 3.1: The 10 real world datasets.

Ratio, *Type* and *Nature* of outliers. Table 3.2 depict these characteristics. We observe a variability of the type and nature of outliers over these datasets, while the outlier ratio is low (no more than 9%) in the majority of datasets with the exception to *Electricity* in which the outlier is almost balanced to the inlier ratio (on/off electricity signal).

It is noticeable that all the selected real datasets are contaminated by point anomalies, except *NYC Taxicab* which is contaminated by contextual anomalies. To the best of our knowledge, any of the presented detectors that is going to be applied on contextual anomalies will miss at least an event, since these detectors originally designed for point anomalies.

3.2.1.1 NYC Taxicab

NYC Taxicab is a univariate time series of data collected over a 7-month period (from 7/14 - 1/15) capturing the activity of taxi ridership passengers aggregated over a 30-minute time window. An expert according to its knowledge of holidays and events in NYC, defined as real anomalies the data points that fell into any of the eight different events: Independence Day (7/4/14 - 7/6/14), Labor Day (9/1/14), Labor Day Parade (9/6/14), NYC Marathon (11/02/14), Thanksgiving (11/27/14), Christmas (12/25/14), New Year's Day (1/1/15), North American Blizzard (1/26/15 - 1/27/15). The rest data points considered normal. The resulted outlier ratio is shown in Table 3.2.

Dataset	Ratio	Type	Nature
Arrhythmia	5.09	Implanted	Subspace
Diabetes	5.08	Implanted	Subspace
Glass	5.14	Implanted	Subspace
Hypothyroid	5.01	Implanted	Subspace
Ionosphere	5.13	Implanted	Subspace
Pendigits	5.00	Implanted	Subspace
MNIST	9.03	Minority	Fullspace
Electricity	56.67	Real	Fullspace
NYC Taxicab	5.19	Real	Fullspace
Shuttle	7.15	Minority	Fullspace

Table 3.2: The contamination details of real datasets in Table 3.1.

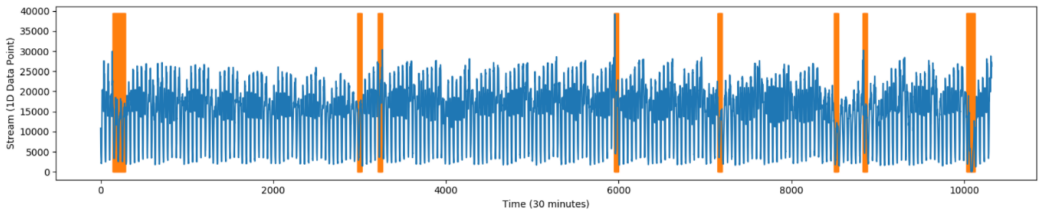

Figure 3.4: The eight contextual anomalies over NYC Taxicab.

In Figure 3.4 we illustrate the time points of the NYC Taxicab, where the X-axis represent the data points and Y-axis their feature value. The orange colored areas indicate each of the 8 anomalous events.

3.2.2 Synthetic

Table 3.3 depicts the seven synthetic datasets we used in our experiments along their respective number of samples and features. We observe a high variability in the number of features (from 1 to 100) of our datasets while the number of samples is relatively the same (∼1000) in all datasets with the exception of *Mulcross* which contains 2 orders of magnitude more samples. Note that, datasets are standardized but not normalized.

The ratio, type and nature of outliers synthetically generated along with outliers, is represented in Table 3.4. We observe that their majority, synthetic outliers lies only to a subset of the feature space of datasets. In the following sections we

Dataset	Samples	Features
$\text{HiCS20}_{(2,3,4,5)SD}$	875	20
$\text{HiCS40}_{(2,3,4,5)SD}$	875	40
$\text{HiCS60}_{(2,3,4,5)SD}$	875	60
$\text{HiCS80}_{(2,3,4,5)SD}$	875	80
$\text{HiCS100}_{(2,3,4,5)SD}$	875	100
Mulcross	262.144	4
Sine wave	1730	1

Table 3.3: The 7 synthetic datasets.

further detail synthetic datasets generation.

Dataset	Ratio	Type	Nature
$\text{HiCS20}_{(2,3,4,5)SD}$	2.0	Density	Subspace
$\text{HiCS40}_{(2,3,4,5)SD}$	2.0	Density	Subspace
$\text{HiCS60}_{(2,3,4,5)SD}$	2.0	Density	Subspace
$\text{HiCS80}_{(2,3,4,5)SD}$	2.0	Density	Subspace
$\text{HiCS100}_{(2,3,4,5)SD}$	2.0	Density	Subspace
Mulcross	10.0	Density	Subspace
Sine wave	1.16	Real	Fullspace

Table 3.4: The contamination details of synthetic datasets in Table 3.3.

It is noticeable that all the selected synthetic datasets are contaminated by point anomalies, except *Sine wave* which is contaminated by collective anomalies. To the best of our knowledge, any of the presented detectors that is going to be applied on collective anomalies can only detect the start and the end of a collective anomaly, since these detectors originally designed for point anomalies.

3.2.2.1 HiCS

HiCS is a multivariate synthetic dataset with highly correlated features, originally introduced for benchmarking binary classifiers [32]. Several HiCS datasets have been released of varying data dimensionality: 10-d, 20-d, 30-d, 40-d, 75-d and 100-d. Each of these HiCS variation, is contaminated in a systematic way throwing

points far away from dense regions over 2-d, 3-d, 4-d and 5-d feature subsets, thus introducing density-based subspace outliers. LOF has been applied exhaustively on each of these subspaces of interest to score the data points. Moreover, subspace dimensionality has different frequency of feature subsets, but there are 5 outliers hidden in each feature subset. Specifically, in 100-d the frequency of the 2-d subspaces is 8 (i.e., 40 outliers hidden in 2-d subspaces). Also, the frequency of 3-d as well as 4-d subspaces is 6 and the frequency of 5-d subspaces is 4. Note that subspace dimensionality is disjoint to each other, since each feature belong to the feature subsets of only one subspace dimensionality (e.g., 2-d).

In our experiments we focus on the 100-d HiCS dataset. We removed outliers that are visible to more than one subspace and then we generated five dataset variations with 20-d, 40-d, 60-d, 80-d and 100-d dimensions that contain exactly the same outliers in 2-d, 3-d, 4-d and 5-d subspaces (see constant outlier ration in 3.4). Our aim was to increase the irrelevant feature ratio as we increase the dimensionality of datasets. In the 20-d dataset all features belong to at least one of the subspaces outliers are visible i.e., they are *relevant* to the subspace outliers of the dataset. As the same outliers are contained in all datasets, in the 100-d dataset we have 20 relevant and 80 irrelevant features.

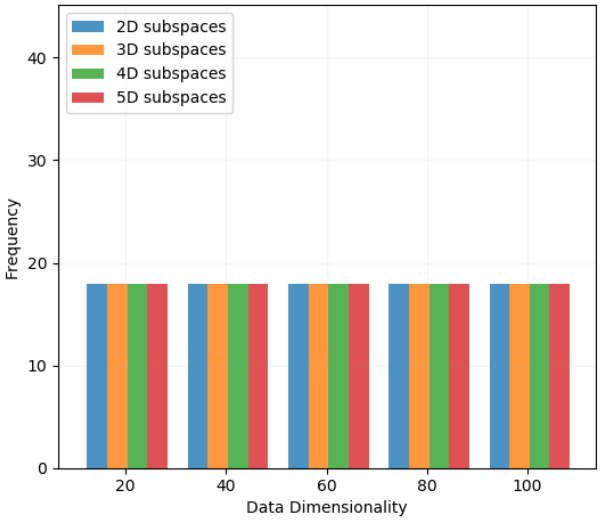

Figure 3.5: Number of outliers per subspace dimensionality in HiCS datasets,

The number of outliers per subspace dimensionality in our variation of HiCS datasets is shown in Figure 3.5. More precisely, per dataset we have included 18 outliers visible in exactly one 2-d, 3-d, 4-d or 5-d subspace. The difficulty of detecting outliers visible in these subspace is varying. Clearly, a combination of 5 features is much harder to be found, than a combination of 2 features. This difficulty increases as the number of irrelevant features increases from 20-d to 100-d HiCS datasets.

3.2.2.2 Mulcross

Mulcross is a Gaussian mixture distribution of inlier and outlier data samples [59], where each sample has 4 features. The dataset has been generated using the same parameters as proposed in [45]: inliers are samples of 1 large Gaussian distribution and outliers are samples of two smaller Gaussian distributions, with the distance between the center of normal and anomaly distributions to be in distance 2. Mulcross is widely known for its cluster instead of uniformly distributed (scatter) anomalies. The resulted outlier ratio is shown in Table 3.4.

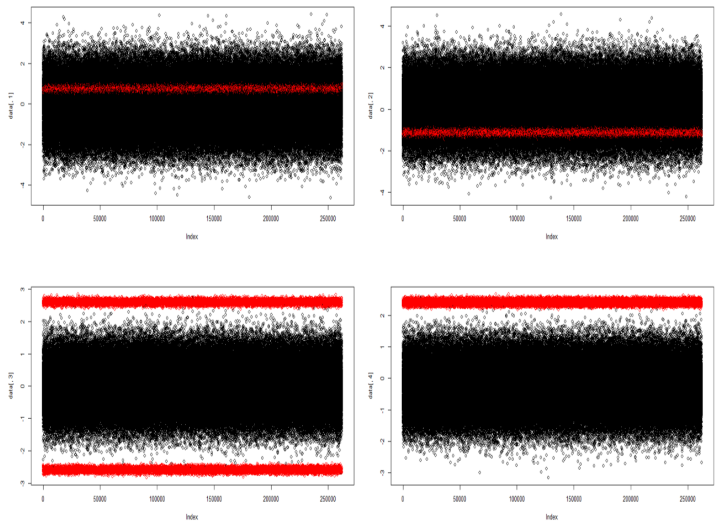

Figure 3.6: Mulcross 1D projections.

In Figure 3.6, we illustrate the data points of Mulcross dataset from each of its 1D subspace. The outliers are colored red and inliers colored black. We observe that subspaces {1} and {2} are less informative than {3} and {4}, since outliers are not separated from inliers.

Furthermore, in Figure 3.7 we illustrate the data points of Mulcross from each of its 2-d subspace, constructing a 4x4 table. Each cell represents a 2-d subspace of different features combination, in which normal and anomalous data points are illustrated. We observe that the 2-d subspaces {1, 2} and {2, 1} are less informative than the rest, since each of its features are already considered less informative.

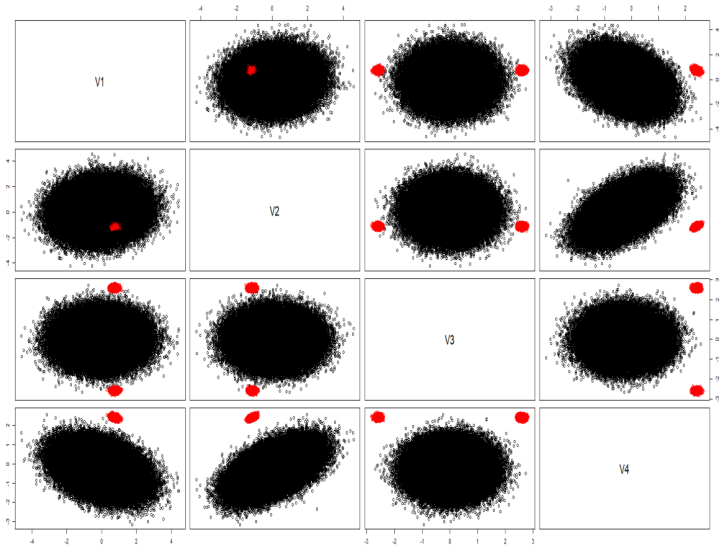

Figure 3.7: Mulcross 2-d projections.

3.2.2.3 Sine Wave

Sine wave is a univariate time series which composes a sine wave, with a dip of 20 time points [26]. The dip is artificially started at the point 1235, considered as a collective anomaly since the individual instances are not outlier themselves. The resulted outlier ratio is shown in Table 3.4.

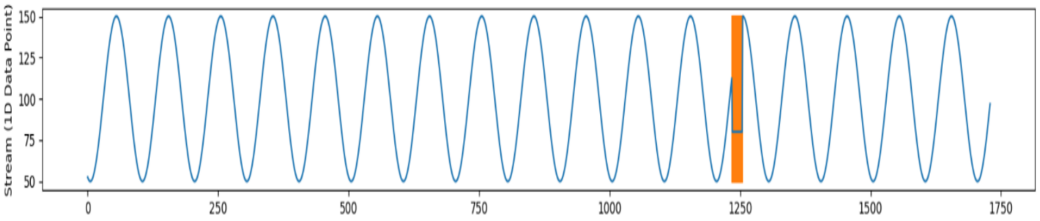

Figure 3.8: A collective anomaly over a sine wave.

In Figure 3.8 we illustrate the time points of the Sine wave, where the X-axis represent the data points and Y-axis their feature value. The orange colored area indicates the anomalous data points.

3.2.3 Dataset Profiling

In this section we present a numerous of dataset characteristics, extracted from our real and synthetic datasets, based on our empirical understanding of the weaknesses of detectors presented in Chapter 2. We know that nearest-neighbor detectors are highly affected by the actual feature values, because the outlierness degree of a data point is extracted according to a distance metric. In addition,

we know that tree-based detectors are highly affected by the number of features and their correlation with true outliers, because the outlierness degree of a data point is extracted according to its tree model structure. So, we dive into a deeper analysis, resulting four new categorical characteristics extracted using if-then rules and capturing monotonic patterns over datasets.

3.2.3.1 AVG Feature Value Range

For each dataset, we first compute the value range of each feature and then result the average over all of its features. An average feature value range is considered low if ≤ 100, otherwise high.

Dataset	**AVG Feature VR**
Arrhythmia	30788354
Diabetes	177
Glass	4
Hypothyroid	64
Ionosphere	2
Pendigits	100
MNIST	182
Electricity	1

Table 3.5: Real Dataset Profiling: AVG Feature Value Range.

From our real datasets, in Table 3.5 we observe that {Glass, Hypothyroid, Ionosphere, Pendigits} characterized with low and the rest {Arrhythmia, Diabetes, MNIST} with high average feature value range.

Dataset	**AVG Feature VR**
$\text{HiCS20}_{(2,3,4,5)SD}$	1.027
$\text{HiCS40}_{(2,3,4,5)SD}$	1.027
$\text{HiCS60}_{(2,3,4,5)SD}$	1.027
$\text{HiCS80}_{(2,3,4,5)SD}$	1.027
$\text{HiCS100}_{(2,3,4,5)SD}$	1.027

Table 3.6: Synthetic Dataset Profiling: AVG Feature Value Range.

From our synthetic datasets, in Table 3.6 we observe that all of our HiCS variations characterized with low average feature value range.

3.2.3.2 Outlier Ratio

The ratio of outliers (OR) is given by the following formula:

$$OR = \#outliers/\#inliers + \#outliers \tag{3.1}$$

Dataset	**OR**
Arrhythmia	5.09
Diabetes	5.08
Glass	5.14
Hypothyroid	5.01
Ionosphere	5.13
Pendigits	5.00
MNIST	9.03
Electricity	56.67

Table 3.7: Real Dataset Profiling: Outlier Ratio.

OR is considered as low if $\leq 30\%$, otherwise high. As we can see in Table 3.7 Arrhythmia, Diabetes, Glass, Hypothyroid, Ionosphere, Pendigits and MNIST have low OR while Electricity has a high OR.

Dataset	**OR**
$\text{HiCS20}_{(2,3,4,5)SD}$	2.0
$\text{HiCS40}_{(2,3,4,5)SD}$	2.0
$\text{HiCS60}_{(2,3,4,5)SD}$	2.0
$\text{HiCS80}_{(2,3,4,5)SD}$	2.0
$\text{HiCS100}_{(2,3,4,5)SD}$	2.0

Table 3.8: Synthetic Dataset Profiling: Outlier Ratio.

The outlier ratio per synthetic dataset is shown in Table 3.8. We observe that all of our HiCS variations have low outlier ratio.

3.2.3.3 Feature VR Outlier Correlation

For each dataset, we compute the percentage of true outliers that have at least, in one of their feature values, a value corresponding to the max value of an HVR feature (we say that these outliers described by those features). A feature that its value range is ≥ 100 characterized high value range *HVR* feature, otherwise low value range *LVR* feature. The correlation between the feature value range and true outliers is low, if the number of outliers described by HVR features is $\leq 30\%$, otherwise high.

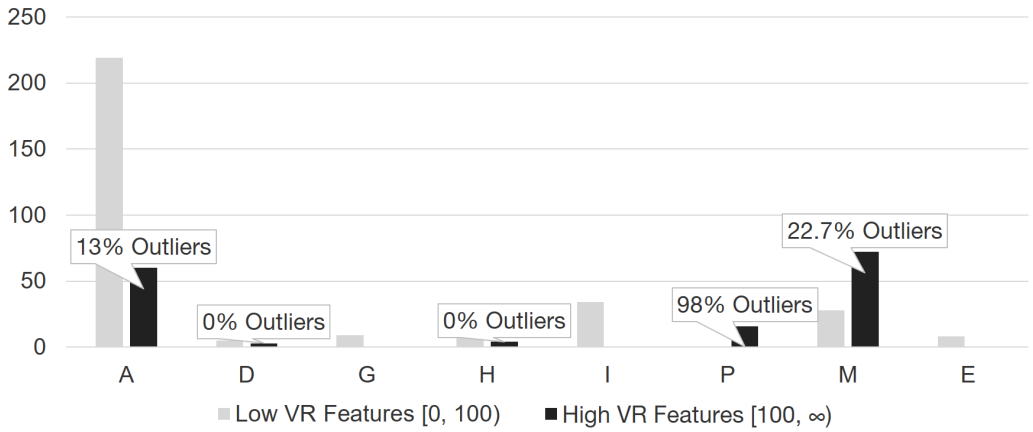

Figure 3.9: Real Dataset Profiling: Feature VR Outlier Correlation.

In Figure 3.9 we illustrate for each real dataset, the percentage of true outliers that are correlated with HVR and LVR features. X-axis represent the datasets and Y-axis the number of features. We observe that {Arrhythmia, Diabetes, Glass, Hypothyroid, Ionosphere, MNIST, Electricity} have low and {Pendigits} has high feature value range outlier correlation.

3.2.3.4 Irrelevant Features to Outliers

For each dataset we first calculate the number of relevant and irrelevant features w.r.t. the outlier subspaces they contain and then we compute the ratio of irrelevant feature according to the formula 3.2. Recall that *relevant* is a feature that belong to at least one subspace, with the rest features to be considered as *irrelevant*. The higher the irrelevant features ratio, the higher the feature irrelevance to outliers.

Dataset	Features	Relevant	Irrelevant	IFR
HiCS20$_{(2,3,4,5)SD}$	20	20	0	0.00
HiCS40$_{(2,3,4,5)SD}$	40	20	20	0.50
HiCS60$_{(2,3,4,5)SD}$	60	20	40	0.67
HiCS80$_{(2,3,4,5)SD}$	80	20	60	0.75
HiCS100$_{(2,3,4,5)SD}$	100	20	80	0.80

Table 3.9: Synthetic Dataset Profiling: Irrelevant Features to Outliers.

$$IFR = \#irrelevant/\#relevant + \#irrelevant \tag{3.2}$$

In Table 3.9 we observe that as we increase the data dimensionality we increase the ratio of irrelevant features (IFR) to the subspace outliers and thus the task of detecting them becomes harder.

3.2.3.5 Discussion

The average value range (AVR), outlier ratio (OR) and irrelevant features ratio (IFR) are the three extracted characteristics that will help us to understand the effectiveness of detectors over synthetic datasets. In Table 3.10 we aggregate these characteristics per synthetic dataset.

Dataset	AVR	OR	IFR
HiCS20$_{(2,3,4,5)SD}$	low	low	-
HiCS40$_{(2,3,4,5)SD}$	low	low	low
HiCS60$_{(2,3,4,5)SD}$	low	low	moderate
HiCS80$_{(2,3,4,5)SD}$	low	low	high
HiCS100$_{(2,3,4,5)SD}$	low	low	very high

Table 3.10: The three extracted characteristics by profiling the synthetic datasets.

Dataset	AVR	OR	VOC
Arrhythmia	high	low	high
Diabetes	high	low	low
Glass	low	low	low
Hypothyroid	low	low	low
Ionosphere	low	low	low
Pendigits	low	low	low
MNIST	high	low	high
Electricity	low	high	low

Table 3.11: The three extracted characteristics by profiling the real datasets.

The average value range (AVR), outlier ratio (OR) and feature value range outlier correlation (VOC), are the three extracted categorical characteristics that are going to be used for the effectiveness explanation of detectors over real datasets. In Table 3.11 we aggregate these characteristics per dataset.

3.3 Algorithms

We have implemented from scratch in Java 8 the two tree-based detectors: HST/F and RRCF (see Chapter 2). We have integrated in our platform the original implementation in java 8 of the distance-based online outlier detection MCOD [35]. For all of the offline detectors we used the python implementation of KNN_W, LOF, iForest from the scikit-learn library version 0.21.2 [54]. In the following subsections we detail the sanity test we used to quantify how well our implementations of HST and RRCF approximate the original ones and introduce candidate values for the hyper-parameters of all detectors.

3.3.1 RRCF Implementation

The original implementation of RRCF is not open-source available [26]. Nevertheless, there exists an open-source RRCF implementation but in python [6] on which we have based our implementation in java 8. The correctness of our implementation is verified through a sanity test, in which we run our implemented detector 30 times and we use the hyper-parameter values as proposed by RRCF authors [26].

Our RRCF implementation is first tested on the *Sine Wave* dataset (see Section 3.3). Our goal goal is to determine if the beginning and the end of the synthetically injected collective anomaly can be spotted. The dataset is treated as a stream, scoring a new point at timestamp 1000+1 with the model built up until time 1000.

The hyper-parameters of this experiment are trees = 100, forget threshold = 512, window size = 4, window slide = 1.

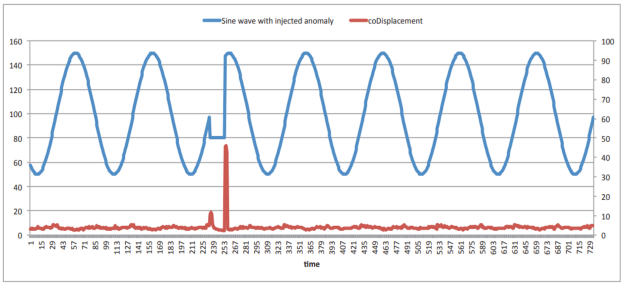

Figure 3.10: Original RRCF implementation on Sine Wave.

In Figure 3.10 we illustrate the anomaly scores of the original RRCF implementation and in Figure 3.11 the anomaly scores of our implementation. In both figures the red curve represent the anomaly score distribution over the sine wave dataset. In our implementation the orange area represents the outliers. We observe that the anomaly score distribution of the original and our implementation, both result their peaks around the point 235 and 255. These points represent the end and the start of the collective anomaly respectively. Therefore, we conclude that both implementations capture the anomaly equally well.

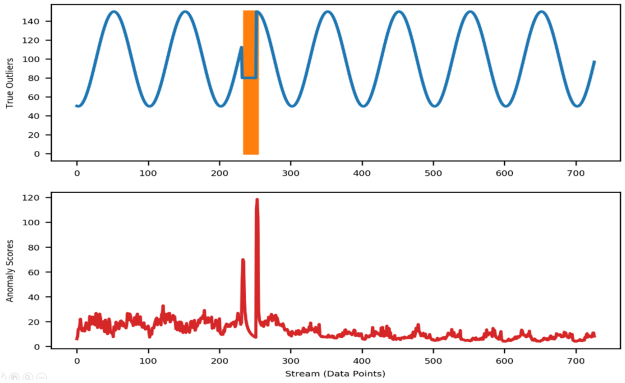

Figure 3.11: Our RRCF implementation on Sine Wave.

We continue the evaluation of our RRCF implementation using the taxi ridership data from the *NYC Taxicab* dataset, presented in Section 3.1. The goal is to determine whether contextual anomalies can be spotted this time. We treat this dataset as a stream, scoring a new data point at time 3740+1 with the model built up until 3740. The hyper-parameters of this experiment are trees = 200, forget threshold = 3072, window size = 48, window slide = 1.

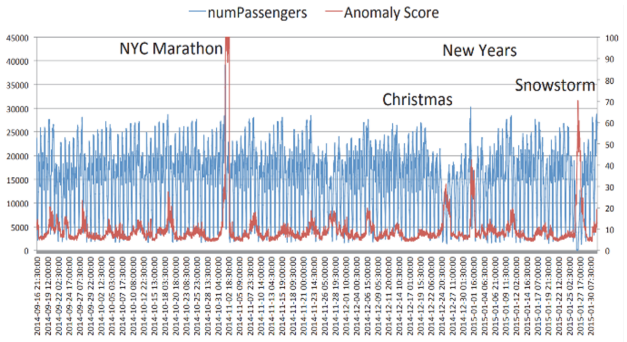

Figure 3.12: Original RRCF implementation on NYC Taxicab.

Figure 3.12 illustrates the anomaly scores of the original RRCF implementation and Figure 3.13 the anomaly scores of our implementation. In both figures the red curve represents the anomaly score distribution over NYC Taxicab. The blue curve represents the data points of the NYC Taxicab dataset. In our implementation, the orange area represents the anomalous events; NYC marathon, thanksgiving, Christmas, new year and snowstorm.

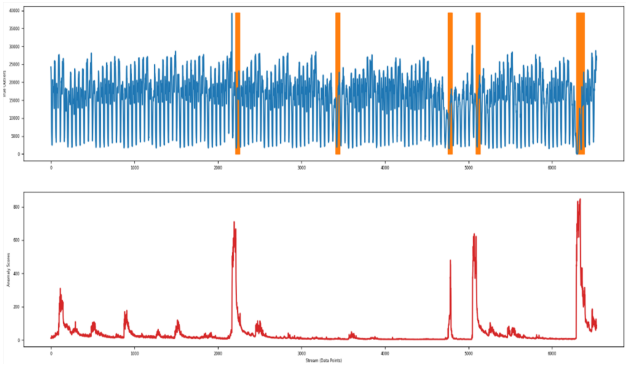

Figure 3.13: Our RRCF implementation on NYC Taxicab.

We observe that our implementation approximates the original, since all the anomalous events captured except *thanksgiving* event, which is not captured by any implementation. The miss of that event, indicates the importance of the proper shingle size selection for its featuraization technique.

We stretched our RRCF implementation in comparison to the original one, on both synthetic and real datasets. We observe that both implementation perform equally good, with small differences to focus on the non-deterministic nature of RRCF.

3.3.2 HST Implementation

The original HST implementation is open-source available in Matlab [67]. An open-source implementation in python is also available [42]. These implementations guided our implementation of the HST in java 8. The correctness of our implementation is verified through a sanity test, in which we run our implemented detector 30 times and we use the hyper-parameter values; trees = 25, max depth = 15 and window size = 250, as proposed by the authors [66].

Our HST implementation is first tested on the Mulcross dataset, presented in Section 3.3. The goal is to determine if the (injected) cluster anomalies can be spotted. The dataset is treated as a sequence of data points, scoring a new data point at the time 183.500+1 with the data structure built up until time 183.500. Therefore, the train size is 183.500 and the sample size is 131.072 data points.

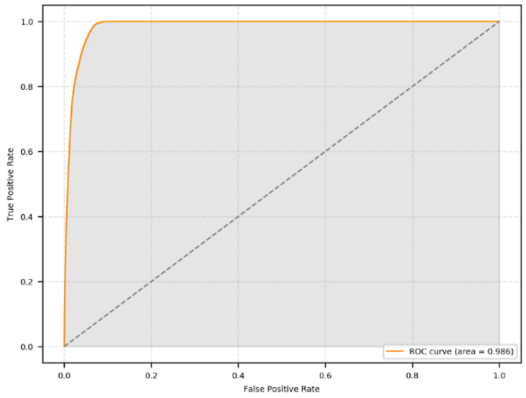

Figure 3.14: The AUC ROC of our HST implementation on Mulcross.

The authors of the original HST implementation reported AUC ROC 0.998 (i.e., 99.8%) on Mulcross dataset. Through our implementation we resulted 0.986 (i.e., 98.6%), as shown in Figure 3.14. Therefore, we observe that both implementation perform equally good.

We continue the evaluation of our HST implementation using the shuttle dataset, presented in Section 3.1. The goal is to determine if our implementation can spot the scatter anomalies. The dataset is treated as a sequence of data points, scoring a new data point at time 34.367+1 with the data structure built up until time 34.367. Therefore, the train size is 34.367 and the sample size is 24.548 data points. The authors of the original HST implementation reported AUC ROC 0.99 (i.e., 99.9%). Through our implementation we resulted 0.988 (i.e., 98.8%), as shown in Figure 3.15. Therefore, we observe that both implementation perform equally good.

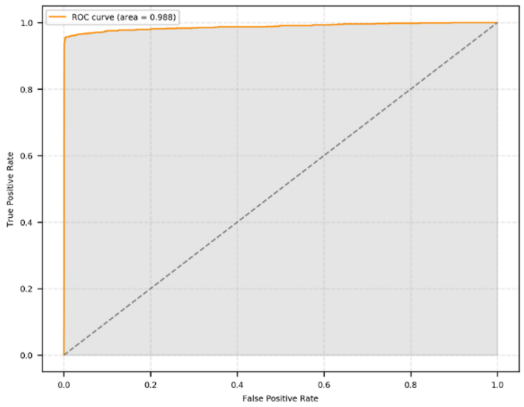

Figure 3.15: The AUC ROC of our HST implementation on Shuttle.

We observed that the efficiency of our implementation is almost equal to the efficiency of original HST implementation, with small differences due to the non-deterministic nature of HST.

3.3.3 Hyper-parameters and Candidate Values

We are in front of a multivariate optimization problem, since detectors have plenty of hyper-parameters especially the online ones that additionally get affected by the window parameters. In Figure 3.16 we illustrate the relation between outlier detectors, dataset and window techniques. We observe that offline detectors have direct access on a dataset, when online detectors access to a dataset through a window technique.

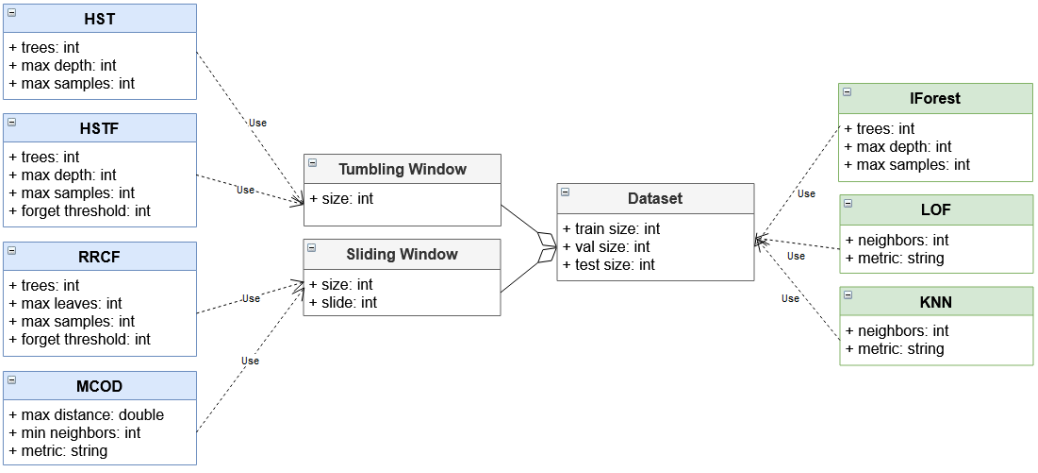

Figure 3.16: The hyper-parameters of our Experimental Environment.

Some of these hyper-parameters have fixed values. The fixed ones are presented

in Table 3.12. We observe fixed values in the form of percentage, refers to the total number of data points in a dataset.

Hyper-parameter	**Fixed Value**
train size	50%
max samples	40%
val size	10%
test size	50%
size	128
slide	64
metric	Euclidean
max depth	ceil(log2(max samples))
max leaves	5000

Table 3.12: The fixed values of the hyper-parameters presented in Figure 3.16.

The rest of the hyper-parameters needs to be optimized. The value range (candidate values) of these hyper-parameters is shown in Table 3.13. We observe that each hyper-parameter corresponds to one or more detectors. The range of the candidate values, selected empirically to reassure the existence of at least an optimal value per detector.

Hyper-parameter	**Value Range**	**Detectors**
trees	25, 50, 100	HST, HSTF, RRCF, iForest
forget threshold	64, 128, 256, 512, max samples	HSTF, RRCF
max distance	$\{0.1 \text{ up to } 4\}_{x300}$	MCOD
min neighbors	$\{1 \text{ up to } 64\}_{x300}$	MCOD
neighbors	5, 10, 15, 20, 30, 50, 100, 150	LOF, KNN_W

Table 3.13: The candidate values of the hyper-parameters presented in Figure 3.16.

3.4 Setup

In this section we present the software and hardware settings of our machine, under which we run all of our experiments.

3.4.1 Software

From the perspective of software, we used Java and Python and windows as an operating system. The versions of these components are shown in the following list.

- Operating System (OS): Windows 10 Pro x64 bit
- Java Version: 1.8
- Python Version: 3.7.4
- Sci-kit Learn Version: 0.21.3

3.4.2 Hardware

From the perspective of hardware, a powerful personal computer was enough for our experiments. The characteristics of that machine are shown in the following list.

- Processor: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
- Installed memory (RAM): 32.0 GB DDR3 1600MHz
- Disk Drive (SSD): 500 GB OCZ-VERTEX3

Chapter 4

Benchmarking Methodology

In this chapter we address the main methodological questions in order to benchmark in a correct and fair way online as well as offline detectors. In Section 4.1, we explain how we can generate sequences of windows whose data points are randomly sampled from the batch datasets of our benchmark (see Section 3.2). In Section 4.2, we investigate the metrics as well as the protocols that we can be used to fairly compare the effectiveness of online detectors on windows with the offline detectors running on batch datasets. In Section 4.3, we present a greedy optimization technique to tune the hyper-parameters of algorithms in order to ensure that detectors may achieve an optimal effectiveness per dataset. In Section 4.4, we introduce a distance metric to compare the effectiveness of detectors across all datasets of our benchmark. In Section 4.5, we illustrate how our methodological choices are taken into account by the architecture of our benchmarking platform (see Figure 3.1).

4.1 MQ1: How streams of inliers and outliers can be simulated from batch datasets?

The key idea to answer this question is to associate the data points contained in a window in the stream with random samples of a batch dataset. To this end, we stratify outliers over inliers using a step related to the outlier ratio of the batch dataset. Hence we are able to guarantee that if $step \leq window\,size$, then each window is going to have similar number of outliers. Besides that, outliers get shuffled before stratification to spread the outliers of different difficulty over windows and therefore to avoid scenarios where all the hard or easy detectable outliers, gathered in a window.

4.1.1 Stream

We consider a data stream in our benchmark, denoted as S, as a sequence of stratified outliers over inliers. In a first step we consider two shuffled sets per

batch dataset: inliers I and outliers O. Next, we create a queue Q which contains all the points of O stratified in the points of I, using a step size = int (number of data points / number of outliers). That queue represents our data stream.

4.1.2 Stream Partitioning

A stream S gets partitioned into two large sequences; train and test. The *train* contains the first *train size* data points of S, which is further partitioned into *sub sample* sequence using the first *max samples* points and *validation* sequence using the rest *val size* points. The *test* contains the rest *test size* data points of S. The hyper-parameters test, train, val size as well as max samples introduced in Table 3.16.

4.1.3 Stream Windowing

Stream partitions gets processed sequentially in window/s by an online detector or at once in a single large window by an offline detector. The stream partition *sub sample* is used by an online as well as offline detector to train a model. Both online and offline detectors use their model to predict scores on stream partitions *validation* and *test*, when online detector additionally updates its model.

4.2 MQ2: What are the evaluation protocols and metrics to best assess the effectiveness of online and offline detectors?

To address this question we rely on the binary or real-valued anomaly scores, returned by an online as well as offline detector. The crux is to choose the suitable protocol to traverse its anomaly scored points and the metrics to evaluate its effectiveness based on these scores.

4.2.1 Evaluation Protocols

We consider two evaluation protocols inspired from supervised learning and regression analysis of time series. The first protocol evaluates in one step the effectiveness of detectors after the last window of data stream by considering the scores of points obtained in all windows. The second protocol evaluates the effectiveness of detectors in each window and average their effectiveness after the processing of the last window.

The protocol 1 estimates the effectiveness of a detector, by applying an evaluation metric to the entire validation or test partition at once. It is influenced by *K-Fold Cross Validation* [71].

The Protocol 2 estimates the average effectiveness of a detector, by applying an evaluation metric to the windows of a validation or test partition per se. It is influenced by *Forward Chaining Cross Validation* [7].

Both evaluation protocols are equally good for evaluating the effectiveness of offline detectors, as for static models not updated in windows they result in the same scoring of inliers and outliers. On the other hand, as the models build by online detectors get updated in each window, only protocol 2 can capture the variance of effectiveness across all the windows required to process a stream partition.

4.2.2 Evaluation Metrics

The effectiveness of a score-based detector can be assessed by two metrics; AUC ROC and AP. The former measures how well the score of outliers is separated from the score of inliers, separability can be achieved even if inliers are higher scored than outliers. The latter measures whether outliers obtain a high score compared to inliers. In addition, we should stress that score-based detectors can be evaluated without the need to normalize its scores (e.g., between [0,1]). That is particularly useful for protocol 2 in which points may obtain different scoring degrees according to the window they have processed, since the more windows the more model updates and therefore the more reliable anomaly scores.

4.2.2.1 AUC ROC

The goodness of a score-based detector on how well outliers are separated from inliers according to scores of points. This property can be measured using the Area Under the Curve ROC (AUC ROC) as proposed in [22]. Receiver Operating Characteristic (ROC) is a two dimensional plot of false positive rate FPR in x-axis and true positive rate TPR in y-axis, for different score thresholds. The higher the AUC ROC the more probable our model to correctly classify the scored data points under outlier and inlier classes. The 0.5 value indicates a random classification and 1.0 value indicates perfect. A model with perfect discrimination, no overlap between outliers and inliers, passes through the upper left corner.

The algorithm, first rank data points in descending score order. Then, a ROC curve is initialized starting from (0,0). For each data point, if its true class is outlier then ROC cursor moves up by 1/outliers, otherwise moves right by 1/inliers. The best threshold to distinguish outliers from inliers, correspond to the score of the point received the highest TPR/FPR value in ROC. The AUC correspond to the area under the constructed ROC.

4.2.2.2 AP

The goodness of a score-based detector on how high outliers are ranked in comparison to inliers according to scores of points. This property can be measured using the Average Precision (AP) as proposed in [74]. The higher the AP the less overlap exist between the lists of scored outliers and inliers. The 0.0 value indicates that all inliers scored higher than true outliers and 1.0 value indicates that all true outliers scored higher than inliers.

The algorithm, first rank data points in descending score order. For each data point, if its true class is outlier then it calculates the current AP, otherwise current AP is set to zero. AP is computed as the average of all AP over the number of true outliers.

4.2.2.3 Appropriate Metric

The AP is more informative than AUC ROC when evaluating score-based classifiers on imbalanced (e.g., anomaly contaminated) datasets. This is a claim supported by many researchers in already published works [61], [15], [46], [44] and [9]. Therefore, AP looks a better option than AUC ROC.

4.3 MQ3: How we measure effectiveness under optimal conditions per detector with respect to its hyper parameters?

Clearly, throughway tuning of their hyper-parameters will enable detectors to exhibit an optimal effectiveness during the validation phase. Random search (RS) is a family of numerical optimization methods that do not require the gradient of the problem to be optimized [57]. Such optimization methods are also known as direct-search, derivative-free, or black-box methods. RS can hence be used on functions that are not continuous or differentiable [25].

Moreover, according to the isolation requirement [28] and [33], the optimization phase should take place on an additional validation data partition, with the data in test partition to remain unseen. Therefore, there are three steps that we need to apply in order to result the final model of a deterministic or not detector. These steps are described as it follows:

1. Generate C configuration settings by random searching (RS) on the candidate hyper-parameter values of a detector presented in Section 3.3.3.

2. Train C models, a model per configuration, and evaluate their effectiveness using AP on the *validation* stream partition.

3. Return the best configuration, corresponding to the model with the maximum evaluated effectiveness over the C models.

These steps are being repeated 100 times for non-deterministic detectors (iForest, HST/F and RRCF). The final model corresponds to the, best configuration, model that is closer to the average effectiveness of 100 runs.

Important Remark. It is also noticeable that commonly in other experimental works, such as [17], [18] and [11], we observe the usage of many default algorithm hyper-parameters using the excuse *"recommended by the authors"*. Driven by that

observation, we highlight that the hyper-parameter optimization is not matter of preference but matter of a fair evaluation between algorithms.

4.4 MQ4: How we rank the effectiveness of detectors across all datasets of our benchmark?

Our answers to the previous methodological questions aim to ensure a fair comparison of different detectors on the same dataset. The final question that arises is how to consistently rank the effectiveness of different detectors across all datasets of our benchmark. To this end, we consider the performance vectors v_a and v_b of two detectors a and b across all datasets. Then we consider the Euclidean norm (L2) [64], to compute the length of the line segment connecting the v_a to v_b and vice versa as follows:

$$L2(v_a, v_b) = \sqrt{\sum_{i=1}^{S}(v_a i - v_b i)^2} \tag{4.1}$$

For each detector, we calculate the L2 distance between its PV and the OPV. Performance Vector (PV) is a vector of S evaluated performances of a detector, where S is the number of datasets. Optimum PV (OPV) is the performance vector of S maximum, according to the evaluation metric, performances. Thereafter, detectors ranked in ascending order according to their calculated distances, in which the lower the L2 the better the detector is. We should stress that L2 is a fair distance metric to rank detectors, since the actually order of performances in a PV, does not affect the final ranking of the detectors. In other words, detectors are not favored by the order of the dataset performances.

We recommend the usage of Euclidean distance, since it gives the shortest or minimum distance between two points (performance vectors). Manhattan distance [63], on the other hand, gives priority to the distance between points only along with the grids, but this requirement does not add any value to the ranking comparison.

4.4.1 A Practical Example of L2

Let us assume three datasets, two detectors and the AP as the performance evaluation metric. Consider the following performance vectors:

- PV_1 = [0.23, 0.18, 0.45].
- PV_2 = [0.33, 0.15, 0.35].
- OPV = [1.00, 1.00, 1.00].

L2(PV_1, OPV) = 1.252 while L2(PV_2, OPV) = 1.262. We observe that none of these results is absolute zero, since L2(OPV, OPV) = 0. But, the L2 of the first

detector is closer to zero than the second detector and therefore the first detector is the best.

4.4.2 Ranking Details

The rank of detectors using statistical metrics like Friedman test [16] does not guarantee a reliable ranking on small PV size ≤ 15 [41]. On the other hand, distance metrics (such as L2) are highly recommended to be used on small PV size [49].

4.5 The Benchmark Evaluation Pipeline

The components of our general benchmark framework, illustrated in Figure 3.1, are grouped based on the four previous methodological questions MQs. The proper sequence of these components formulate the benchmark pipeline of online and offline detectors over contaminated streams.

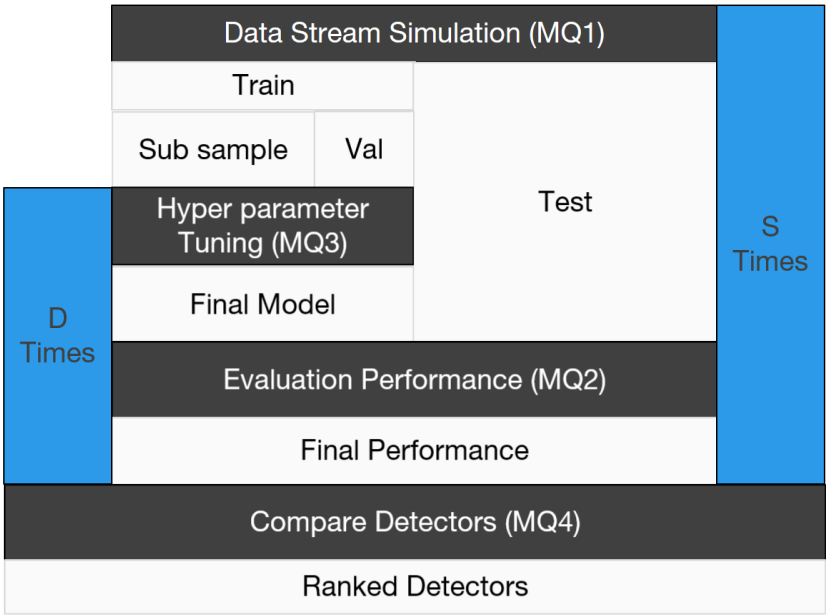

Figure 4.1: The benchmark pipeline according to the Figure 3.1.

The pipeline of our benchmark is illustrated in Figure 4.1. Methodologies are black colored, iterations are blue colored and intermediate results are colored grey. A benchmark process of D detectors over S simulated streams, can be done in the following eight steps:

1. Apply the MQ1 to simulate a contaminated stream.
2. Split stream into train and test partition.

3. Split train into subsample and validation partition.
4. Apply MQ3 to optimize the hyper-parameters of a detector on the validation partition and result its final model.
5. Apply MQ2 to evaluate the effectiveness of its final model on the test partition and result that effectiveness.
6. Repeat D times the steps 4 and 5.
7. Repeat S times all of the above steps.
8. Apply the MQ4 to rank the D detectors according to their S effectiveness.

Chapter 5

Experimental Evaluation

In this chapter we present three major experiments aiming to provide insights regarding the following questions: (1) What is the *effectiveness* of online and offline detectors on real data? (2) What is the *robustness* of online and offline detectors against increasing data and subspace dimensionality? (3) How *sensitive* the online detectors are with respect to their hyper-parameters?

To answer these questions we rely on the benchmarking environment presented in Section 3.1 and evaluation methodology introduced in Section 4. We have designed three comprehensive series of experiments providing evidence regarding (a) which data characteristics favor the effectiveness of online and offline detectors (see Section 5.1); (b) how well they scale as we increase the dimensionality of data and subspace (see Section 5.2); and (c) how sensitive they are in tuning hyper-parameters such as the number and depth of random trees, the size and speed of windows (see Section 5.3). In total, over than 50.000 experimental runs were performed, because of the model optimization especially of the non-deterministic detectors.

5.1 Online versus Offline effectiveness

The objective of this experiment is to quantify how well the effectiveness of online approximate offline detectors on real datasets contaminated by sub/full space outliers. To this end, we benchmark four online detectors {HST, HSTF, RRCF, MCOD} and three offline detectors {LOF, KNN_W, iForest} using eight real datasets. Six of them {Arrhythmia, Diabetes, Glass, Hypothyroid, Ionosphere, Pendigits} are contaminated by subspace outliers and two {MNIST, Electricity} by fullspace outliers.

Table 5.1 depicts the hyper-parameters of the detectors as they have been optimized per real dataset. We can easily observe that the effectiveness of nearest-neighbor detectors and in particular of the online one (MCOD) heavily depends on the selection of hyper-parameters R and K per dataset.

Dataset	$\mathbf{K}_{KNN_W}$	$\mathbf{K}_{LOF}$	Trees	Forget T.	Depth	R	$\mathbf{K}_{MCOD}$
Arrhythmia	5	5	100	180	2	2.279	7
Diabetes	5	5	100	347	8	1.152	48
Glass	5	5	100	96	6	0.374	1
Hypothyroid	5	5	100	1508	10	3.922	30
Ionosphere	5	5	100	140	7	0.689	54
Pendigits	5	5	100	4396	12	2.101	26
MNIST	150	150	100	3041	11	3.475	16
Electricity	150	5	100	4000	11	0.415	28

Table 5.1: Optimized Hyper-parameters of online and offline detectors.

5.1.1 Results

Table 5.2 illustrates the resulted (M)AP of all online and offline detectors on the eight real datasets featuring different outlier types and ratios. The leading detector per dataset is noted in bold. Online outperform offline detectors on 4/8 datasets (Arrhythmia, Diabetes, MNIST and Electricity) while the inverse behavior is observed on 3/8 datasets (Glass, Ionosphere and Pendigits). There is also a tie in the effectiveness of online and offline detectors in one dataset (Hypothyroid).

Dataset	HST	HSTF	RRCF	MCOD	LOF	$\mathbf{KNN}_W$	iForest
Arrhythmia	0.151	**0.157**	0.085	0.087	0.069	0.082	0.092
Diabetes	0.081	0.076	**0.098**	0.083	0.094	0.089	0.057
Glass	0.060	0.060	0.087	0.088	**0.127**	0.102	0.061
Hypothyroid	0.082	0.084	**0.108**	0.081	**0.108**	0.083	0.059
Ionosphere	0.070	0.110	0.112	0.142	**0.154**	0.089	0.077
Pendigits	0.118	0.128	0.293	0.077	**0.691**	0.524	0.130
MNIST	0.668	**0.676**	0.285	0.122	0.374	0.402	0.319
Electricity	0.744	0.720	0.651	**0.780**	0.534	0.505	0.509

Table 5.2: (M)AP of online and offline detectors on real datasets.

According to (M)AP presented in Table 5.2, online and offline detectors are ranked from best to worst (see Section 4.4), as follows: {LOF:2.160, KNN_W:2.231,

HSTF:2.241, HST:2.261, RRCF:2.281, MCOD:2.400, iForest:2.407}. We observe that online tree-based detectors (HST/F and RRCF) actually outperform offline detectors (iForest). This is not true for nearest-neighbor algorithms where offline detectors (LOF, KNN_W) outperform online detectors (MCOD). In average, we observe that in real datasets of our benchmark, offline detectors are slightly better ranked (average range value 2.266) than online detectors (average rank value 2.296). In other words, online approximate very well the offline detectors.

In Table 5.3 we summarize the main characteristics of the real datasets (see Section 3.2.3) along with the online and offline detectors exhibiting the leading effectiveness (MAP) per dataset. In this analysis, we focus on the outlier ratio (OR) and on the average feature value range (AVR), since these two data characteristics provide sufficient insights to understand the behavior of nearest-neighbor in comparison with tree-based detectors.

Dataset	**AVR**	**OR**	**Best Detector**
Arrhythmia	high	low	HSTF
Diabetes	high	low	RRCF
Glass	low	low	LOF
Hypothyroid	low	low	LOF
Ionosphere	low	low	LOF
Pendigits	low	low	LOF
MNIST	high	low	HSTF
Electricity	low	high	MCOD

Table 5.3: Leading effectiveness analysis over detectors in Table 5.2.

In *low* average feature value range datasets (Glass, Hypothyroid, Ionosphere and Pendigits) LOF is on average 86% more effective than online detectors. This is due to the fact that nearest-neighbor detectors are better on capturing outliers close to inliers, since a well-tuned model learns very well the actual distances between data points. Moreover, in a *high* outlier ratio dataset (Electricity) MCOD is 60% more effective than offline detectors. This is due to the fact that online detectors better separate outliers from inliers on high contamination in data using a windowing sampling technique. Specifically, MCOD outperform LOF, because it is favored by its model reconstruction (i.e., PD list and Micro-clusters update). Furthermore, in *high* average feature value range datasets (Arrhythmia, Diabetes and MNIST) HSTF and RRCF outperform offline detectors. On average, HSTF is 36% and RRCF is 13% more effective over subspace (Arrhythmia, Diabetes) outliers while HSTF is 60% more effective and RRCF is 25% less effective over

fullspace (MNIST) outliers. This is due to the fact that tree-based detectors better capture extreme feature value outliers, since their feature structure isolates more easily points with such extreme values.

Continuing our analysis, we investigate the conditions under which tree-based detectors exhibit the best effectiveness. Table 5.4 details the effectiveness of leading tree-based detectors per dataset by considering the outlier ratio (OR), the feature value range outlier correlation (VOC) and nature of outliers (Nature).

Dataset	**VOC**	**OR**	**Nature**	**Best Tree-based Detector**
Arrhythmia	high	low	subspace	HSTF
Diabetes	low	low	subspace	RRCF
Glass	low	low	subspace	RRCF
Hypothyroid	low	low	subspace	RRCF
Ionosphere	low	low	subspace	RRCF
Pendigits	low	low	subspace	RRCF
MNIST	high	low	fullspace	HSTF
Electricity	low	high	fullspace	HST

Table 5.4: Leading effectiveness analysis over tree-based detectors in Table 5.2.

In datasets contaminated by *subspace* outliers (Diabetes, Glass, Hypothyroid, Ionosphere and Pendigits) RRCF is on average 47% more effective than HST/F detectors. This is due to the fact that model reconstruction allows to periodically update the structure of trees with feature subsets that are more relevant (i.e., contained) to the subspaces in which outliers were introduced. In datasets contaminated by *fullspace* outliers (MNIST, Electricity) or with a *high* feature value range outlier correlation dataset (Arrhythmia), HST/F is on average 42% more effective than RRCF. This is due to the fact that models (eventually with a forgetting mechanism) that periodically update only the leaves of trees (i.e., the mass profiles) can effectively isolate outliers defined in the full feature set. Furthermore, in a *high* outlier ratio dataset (Electricity) HST is 3% better than HSTF. This is attributed to the fact that updates on the tree leaves due to the forgetting mechanism may accidentally damage the mass profiles of inlier feature regions and so may cause more false positives.

5.1.2 Insights

According to our previous analysis, we conclude that online not only approximate very well offline detectors, but they may also outperform them under specific

conditions. This is a surprising result given that dynamic models of online detector are built over samples of the datasets used to build the static models of offline detectors.

1. *Outlier Ratio (OR).* Online detectors (HST/F, RRCF and MCOD) are more robust than offline in high outlier ratios.

2. *Average Feature Value Range (AVR).* Tree-based online detectors (HST/F, RRCF) are particularly effective to isolate outliers with extreme feature values.

   (a) *Fullspace Outliers (FO).* Detectors (HSTF) updating only the mass profiles in the leaves of the trees (i.e., with low computational cost) are sufficient to detect fullspace outliers.

   (b) *Subspace Outliers (SO).* Detectors (RRCF) updating the entire tree structure (i.e., with a high computational cost) are needed to detect subspace outliers.

   Online (MCOD) and offline (LOF, KNN) nearest-neighbor detectors can effectively identify outliers whose feature values are close to the inliers.

5.2 Robustness

In this experiment we evaluate the scalability of online and offline detectors against increasing data and subspace dimensionality, with constant outlier ratio. To this end, we benchmark four online detectors {HST, HSTF, RRCF, MCOD} and three offline detectors {LOF, KNN_W, iForest} using 20 synthetic datasets. The 5 data dimensionality variations {HiCS20, HiCS40, HiCS60, HiCS80, HiCS100}, where each of them contaminated by four variations of only 2-d, only 3-d, only 4-d and only 5-d subspace outliers (see Section 3.2.2.1).

Table 5.5 depicts the hyper-parameters of the detectors as they have been optimized per synthetic dataset. We can easily observe again (as in Table 5.1) that an optimal MCOD effectiveness strongly depends on hyper-parameters tuning per dataset.

Dataset	$\mathbf{K}_{KNN_W}$	$\mathbf{K}_{LOF}$	Trees	Forget T.	Depth	R	$\mathbf{K}_{MCOD}$
HiCS20_{2D}	5	10	100	350	9	1.509	40
HiCS20_{3D}	5	10	100	350	9	1.527	35
HiCS20_{4D}	5	10	100	350	9	1.877	33
HiCS20_{5D}	5	10	100	350	9	2.033	49
HiCS40_{2D}	10	10	100	350	9	2.599	33
HiCS40_{3D}	10	10	100	350	9	2.533	17
HiCS40_{4D}	10	10	100	350	9	2.710	45
HiCS40_{5D}	10	10	100	350	9	2.563	43
HiCS60_{2D}	10	20	100	350	9	3.236	20
HiCS60_{3D}	10	20	100	350	9	2.599	33
HiCS60_{4D}	10	20	100	350	9	3.514	48
HiCS60_{5D}	10	20	100	350	9	3.375	53
HiCS80_{2D}	5	5	100	350	9	3.875	47
HiCS80_{3D}	5	5	100	350	9	3.546	55
HiCS80_{4D}	5	5	100	350	9	3.661	34
HiCS80_{5D}	5	5	100	350	9	3.447	28
HiCS100_{2D}	15	15	100	350	9	4.349	17
HiCS100_{3D}	15	15	100	350	9	4.262	47
HiCS100_{4D}	15	15	100	350	9	4.348	37
HiCS100_{5D}	15	15	100	350	9	4.615	51

Table 5.5: Optimized Hyper-parameters of online and offline detectors.

5.2.1 Results

Table 5.6 reports the (M)AP of all online and offline detectors over the five synthetic datasets of increasing data dimensionality (20-d, 40-d, 60-d, 80-d and 100-d). Specifically, each detector is applied four times on every of these datasets, since we have four subspaces (2-d, 3-d, 4-d and 5-d) contamination variations per dataset, and then we report the average of the (M)AP for each subspace dimensionality. The best effectiveness of online and offline detectors per dataset has been noted in bold.

Dataset	HST	HSTF	RRCF	MCOD	LOF	KNN_W	iForest
HiCS20	0.020	0.020	0.038	**0.164**	**0.173**	0.153	0.090
HiCS40	0.031	0.034	0.047	**0.131**	**0.073**	0.055	0.048
HiCS60	0.024	0.030	0.038	**0.120**	**0.043**	0.040	0.038
HiCS80	0.032	0.032	0.044	**0.116**	**0.050**	0.046	0.040
HiCS100	0.036	0.039	0.043	**0.120**	0.055	**0.058**	0.034

Table 5.6: The average (M)AP effectiveness of online and offline detectors over increasing data dimensionality.

As we can see in Table 5.6, proximity-based detectors (LOF, KNN_W and MCOD) outperform tree-based detectors (iForest, RRCF and HST/F) regardless of the data dimensionality. This is due to the fact that in these datasets the latter struggle to isolate outliers in low average feature value range (see Section 5.1.2). MCOD is one order of magnitude more effective than tree-based ones in each HiCS dataset. This behavior also observed in LOF and KNN_W, but only at the 20-d dataset. On average, LOF is 11% better than KNN_W, because the introduced subspace outliers are density-based (see Section 3.2.2.1). On the other hand, RRCF is on average 30% more effective than HST/F, due to the updates in the feature structure of trees (see Section 5.1.2). As a matter of fact, HST exhibits the worst effectiveness, due to updates only on the mass profile in the leaves of the trees and the absence of a forgetting mechanism (unlike HSTF see Section 2.3.3). Specifically, we observe that HSTF is 10% better than HST. In the following section, we detail the (M)AP of online and offline detectors for 2-d, 3-d, 4-d, 5-d subspace outliers across all HiCS datasets.

5.2.1.1 MAP Across Increasing Data Dimensionality

Moreover, in Table 5.6 we can observe that the effectiveness of LOF, KNN_W and iForest gets *decreased* while *increasing* the data dimensionality. This is due to the fact that their ability to separate outliers from inliers depends both on the data dimensionality and on the ratio of irrelevant features. As by increasing data dimensionality all pairs of data points become almost equidistant (*distance concentration*), LOF and KNN_W struggle to separate outliers from inliers. In the experiment reported in [75] this effect has been observed at 100-d, while in our experiment started earlier even at 40-d. This is due to the fact that HiCS datasets are contaminated by subspace instead of fullspace outliers. iForest fails to isolate subspace outliers through a lower ratio of relevant features, since by increasing the number of features which are irrelevant to the outliers, the noise in tree structures constructed by uniformly sampling features gets increased. In the experiment reported in [60] this effect has been observed with the addition of 30

irrelevant features, while in our experiment started earlier even with the addition of 20 irrelevant features due to the contamination of HiCS datasets with subspace outliers.

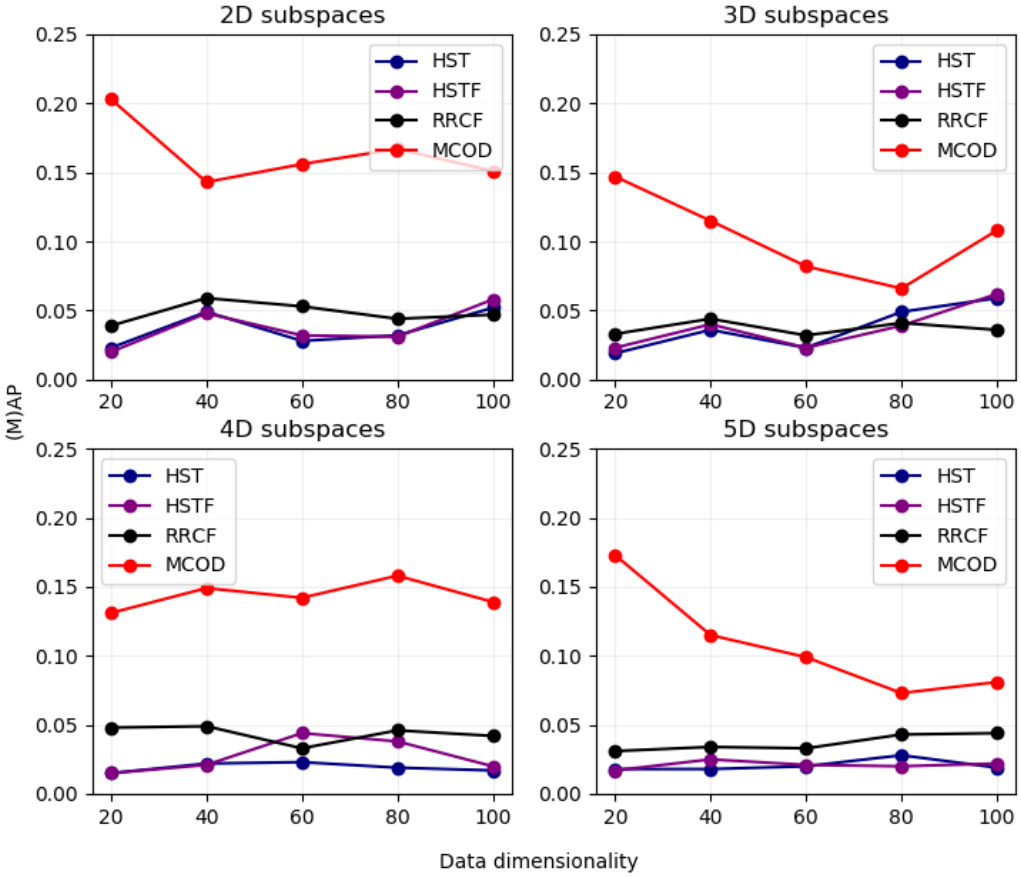

Figure 5.1: (M)AP effectiveness of online detectors for subspace outliers of different dimensionality.

The effectiveness of MCOD, RRCF and HST/F remains *stable* while *increasing* data dimensionality. This is mainly due to the usage of data sub-sampling and incremental model updates. Unlike offline detectors working in one large window, online detectors process data in several small windows. These windows are fed in our benchmark with stratified outliers over shuffled inliers (see Section 4.1.3). It is well known that processing data in sub-samples reduces both the outlier swamping and masking effects [45] and [76], that may incur as we increase the dataset dimensionality. In addition, using windows of size not multiple times larger to the data dimensionality, as in the case of one large window, a better separability (almost linear in 100-d HiCS) between points is being achieved [14]. For these reasons, we observe in Figure 5.1 that online detectors can identify subspace outliers of a

specific dimensionality (e.g., 2-d) similarly well across all HiCS datasets (i.e., 20-d, 40-d, 60-d, 80-d and 100-d). Note that, the small discrepancies in detectors effectiveness is attributed to the non-deterministic nature (for HST/F and RRCF) and hyper-parameter optimization (for MCOD, HST/F and RRCF) of the detection algorithms.

In addition, due to the incremental model updates they perform, online detectors better adapt to the data characteristics of the windows they process. As we can see in Figure 5.1, 2-d and 3-d (low dimensional) subspace outliers are well captured by all online detectors: HST/F, RRCF and MCOD. On the other hand, 4-d and 5-d (high dimensional) subspace outliers are best captured only by MCOD and RRCF. Clearly, as HD subspace outliers are harder to be detected than LD subspace outliers (see Section 3.2.2.1), the model update and forgetting mechanisms implemented by MCOD and RRCF are crucial for their good effectiveness. In the following subsection, we will detail the contribution of the model updates of each detector, on capturing outliers of different subspace dimensionality.

5.2.1.2 MAP Across Subspace Dimensionality

The (M)AP of HST, HSTF, RRCF and MCOD for subspace outliers of different dimensionality (2-d, 3-d, 4-d and 5-d) is presented in Tables 5.7, 5.8, 5.9 and 5.10 respectively. The *AVG* column in these tables, indicates per subspace dimensionality the average (M)AP across all HiCS datasets.

HST	**20D**	**40D**	**60D**	**80D**	**100D**	**AVG**
2D	0.023	0.049	0.028	0.032	0.052	0.037
3D	0.019	0.036	0.023	0.049	0.059	0.037
4D	0.015	0.022	0.023	0.019	0.017	0.019
5D	0.018	0.018	0.020	0.028	0.019	0.021

Table 5.7: HST (M)AP for subspace outliers of different dimensionality.

HSTF	**20D**	**40D**	**60D**	**80D**	**100D**	**AVG**
2D	0.020	0.048	0.032	0.031	0.058	0.038
3D	0.023	0.040	0.023	0.039	0.062	0.038
4D	0.015	0.021	0.044	0.038	0.020	0.028
5D	0.017	0.025	0.021	0.020	0.022	0.021

Table 5.8: HSTF (M)AP for subspace outliers of different dimensionality.

RRCF	20D	40D	60D	80D	100D	AVG
2D	0.039	0.059	0.053	0.044	0.047	0.049
3D	0.033	0.044	0.032	0.041	0.036	0.037
4D	0.048	0.049	0.033	0.046	0.042	0.044
5D	0.031	0.034	0.033	0.043	0.044	0.037

Table 5.9: RRCF (M)AP for subspace outliers of different dimensionality

MCOD	20D	40D	60D	80D	100D	AVG
2D	0.203	0.143	0.156	0.167	0.151	0.164
3D	0.147	0.115	0.082	0.066	0.108	0.104
4D	0.131	0.149	0.142	0.158	0.139	0.144
5D	0.173	0.115	0.099	0.073	0.081	0.109

Table 5.10: MCOD (M)AP for subspace outliers of different dimensionality.

MCOD outperforms by one order of magnitude the other online detectors on finding true outliers in any subspace dimensionality. This is due to the fact that MCOD benefit from re-computing the actual (L2) distances between points in subsamples but also from updating both the content (i.e., with inliers) and the number of micro-clusters (i.e., deleting old and inserting new). Then, tree-based online detectors exhibit a similar effectiveness across subspaces of different dimensionality. RRCF is slightly better than HST/F as it updates its tree model with feature subsets that are more relevant to the subspaces of outliers contained in a window. This is not the case of HST/F that updates only the mass profiles of its constant feature regions. Clearly, the activation of a forgetting mechanism allows HSTF to capture not only 2-d and 3-d subspace outliers as HST, but also 4-d subspace outliers. Both however, face difficulties in finding 5-d subspace outliers.

5.2.2 Insights

According to our previous analysis, we conclude that online detectors exhibit a robust effectiveness when increasing data and subspace dimensionality.

1. *Irrelevant Features Ratio (IFR).* Online detectors (HST/F, RRCF and MCOD) are more robust than offline (LOF, KNN and iForest) detectors against an increasing ratio of irrelevant features. In particular, MCOD well-tuned to the characteristics of the data stream archives an optimal effectiveness for every data and subspace dimensionality.

2. *Subspace Outliers Dimensionality (SOD).* Online detectors updating the entire structure of their model (MCOD and RRCF) are more robust than detectors updating only the mass profiles of tree leaves (HST/F) against subspace outliers of increasing dimensionality. In particular, RRCF exhibits the best effectiveness among all tree-based detectors.

5.3 Sensitivity

In this category of experiments we evaluate the sensitivity of online detectors regarding the tuning of hyper-parameters such as the Window *Size* and *Slide* (Section 5.3.1) or the *Forgetting Threshold* and *Maximum Depth* (Section 5.3.2).

We benchmark four tree-based online detectors {HST, HSTF, RRCF, MCOD} against the real dataset Hypothyroid contaminated by subspace outliers. The remaining hyper-parameters are optimized as depicted in Table 5.1.

5.3.1 Varying Window Hyper-parameters

The objective of this experiment is to trace the effectiveness of tree-based online detectors by increasing their window size and slide hyper-parameters. Table 5.11 illustrates the (M)AP of the tumbling window based detectors (HST, HSTF) by varying the window size. The best detector per window size is noted in bold. We observe that the effectiveness of both HST and HSTF decrease while increasing window size. This is due to the fact that the outlier over inlier ratio gets also decreased and therefore the effectiveness of detectors becomes more prone to false positives.

Size	**HST**	**HSTF**
128	**0.082**	**0.084**
256	0.071	0.071
512	0.060	0.062

Table 5.11: (M)AP for increasing window size.

Table 5.12 presents the (M)AP of the sliding window based detectors (RRCF and MCOD) by varying the window slide. The best detector per window slide is noted in bold. We observe that the effectiveness of both RRCF and MCOD decrease while increasing the window slide. This is due to the fact that less windows are needed to process the stream and as a result fewer model updates are performed. Therefore, the average effectiveness of detectors is affected by both false positives and negatives rates. We finally observe that RRCF is less sensitive to the window slide than MCOD in tumbling mode (slide = size = 512). This is due to the fact that MCOD is based on the window slide to forget past points.

Slide	RRCF	MCOD	Size
64	**0.108**	**0.081**	512
128	0.085	0.066	512
256	0.082	0.066	512
512	0.092	0.062	512

Table 5.12: (M)AP for increasing window slide.

5.3.2 Varying Detector Hyper-parameters

The objective of this experiment is to trace the effectiveness of tree-based online detectors when increasing their forgetting threshold and maximum depth hyper-parameters. Table 5.13 reports the resulted MAP of HST and HSTF over increasing maximum depth values. Also, Table 5.13 depicts the MAP of RRCF and HSTF using 5 and 11 as maximum depth over increasing forgetting threshold values.

Max Depth	HST	HSTF
5	0.085	0.089
10	0.084	0.081
11	0.082	0.084
15	0.086	0.078

Table 5.13: (M)AP for increasing maximum depth.

Forget Threshold	HSTF11	HSTF5	RRCF
60%	0.085	0.082	0.130
70%	0.085	0.085	0.125
80%	0.083	0.077	0.120

Table 5.14: (M)AP for increasing forgetting threshold.

We observe that HSTF reaches an effectiveness peak when the maximum depth equals to 5 (see Table 5.13). We also observe that its effectiveness over increasing the forgetting thresholds is relatively stable when the maximum depth equals to 11. (see Table 5.14). This is due to the fact that 5 is a relatively low depth w.r.t. the number of data points in the selected dataset (Hypothyroid) and therefore a tree model cannot split the data space into the required mass regions. Furthermore, we

observe that HSTF_{11} and RRCF exhibit a decreasing effectiveness when increasing the forgetting threshold. This is due to the fact until starting forgetting, HSTF continuously increases the mass profiles in the tree leafs and RRCF continuously inserts the new data points in the trees while reconstructs their feature structure whenever is needed (i.e., for outliers).

Chapter 6

Conclusions and Future Work

In this thesis we introduced a novel framework for benchmarking online in contrast to offline anomaly detectors. We have payed particular attention in understanding which detector is more suited for a dataset with specific characteristics (e.g., high ratio of irrelevant features, low average feature value range, sub/full space contamination, etc.). To this end, we have addressed several methodological questions that allow us to fairly measure the effectiveness of different online and offline detectors over the same set of benchmark datasets. We introduced the stream simulation and its partitioning, showing that the key point is on stratification of outliers over inliers. We showed the importance of tuning the hyper-parameters of online and offline algorithms using few initial windows and adopted an evaluation protocol influenced by Forward Chaining Cross Validation that average the precision of detectors across several windows. Last but not least, we exploited the Euclidean Distance (L2-norm) as a ranking metric of detectors executed over several datasets.

More precisely, we measured the (mean) average precision, robustness and sensitivity of 7 top-notches online and offline detectors on real and synthetic datasets contaminated with subspace and fullspace outliers. The gist of our findings is that online detectors are quite effective in detecting increasing rates of subspace outliers in data streams. In particular, proximity-based detectors like MCOD and tree-based detectors like RRCF prove to be more efficient than their offline counterparts due to the heavy reconstruction of their models as new data points arising in a stream. For fullspace outliers less computationally costly algorithms that update only the mass profiles of tree leafs seems to work. In addition, RRCF and HSTF are best for detecting outliers with extreme feature values (due to their feature structure) while MCOD when adequately tuned can overcome outliers swamping and masking problems even in high dimensional datasets, since according to [76] the accuracy of the anomaly detection depends on the estimation resolution of the data density and thus the k-NN distance.

More precisely, the effectiveness of online detectors is more robust than offline detectors when increasing the ratio of irrelevant features in datasets contaminated

with subspace outliers. In addition, MCOD and RRCF shown to be more robust than HST/F against increasing subspace dimensionality of outliers. Despite its great effectiveness, MCOD is shown to be quite sensitive to its hyper-parameter tuning like the size and slide of windows.

Our benchmarking platform is available for download as an open source code project [23]. It relies on a generic architecture that can easily incorporate additional datasets and detectors to conduct new experiments. We could for instance include more online detectors such as RST [73] (another variation of HST) and LODA [55] (a lightweight online detector) as well as additional datasets of high data dimensionality (exceeding 200-dimensions) and volume (dozens of Gigabytes) [8] that due to time constraints haven't be considered in our work.

The main limitations of our work are related to the completeness of our meta-analysis and the trade-off between the effectiveness and efficiency of the detectors included in our benchmark. More precisely, the meta-characteristics of datasets are chosen empirically based on the experiments regarding the effectiveness and robustness of detectors, using if-then rules fitting to the size, dimensionality and contamination ratio and value range of our datasets. Although our chosen meta-characteristics (e.g., AVR) are applicable to any new dataset, we may need to re-calibrate their if-then rules (e.g., low and high) when the new dataset has out-range size, dimensionality or contamination values (e.g., 1000-d dataset).

Additionally, more thorough experiments are needed to study the trade-off between the MAP and execution time of detectors. Although, we introduced the complexity of detectors for scoring, training and updating their model, we have not conducted stress tests to measure their actual execution time. We plan to benchmark the time required by detectors to update their model and score points against increasing data dimensionality and window size, slide according to their complexity parameters.

Based on our findings we hope to encourage more researchers to address the challenges of online detectors. It is clear that a lot of space remains for improving existing online detectors. In the sequel, we provide some preliminary ideas under three research directions:

- It is remarkable that the feature selection of all tree-based detectors is random and uniformly, except RRCF that uses a proportionally to the (dataset) min/max feature value range technique. But when the min/max values are the same for all features (e.g., normalized data), then that technique fails (returned into uniform selection). Depth-based methods are very powerful, but there is a need for a new feature selection method to be plugged in. A useful work to that direction proposed [24], in which they estimate one-class Gini index to select the split feature and value, assuming uniform as the theoretical distribution of the outliers.

- The necessity for model reconstruction and forgetting mechanism is indisputable for a dynamic model (see RRCF and MCOD). MCOD uses the slide

of the sliding windows as its forgetting mechanism (i.e., delete data from PD list and Micro-clusters), and therefore there are not much to suggest on that. On the other hand, RRCF forgets by the deletion of internal and external nodes. But that is an error prone task from the perspective of false positives and false negatives. Our suggestion focus on the deletion of entire trees as a forgetting mechanism, instead of nodes.

- Another way to update a dynamic model of an online detector, shown in HST. They do not reconstruct the tree models, but they simply apply learn (i.e., increase the mass profiles of nodes) updates. We improved their idea by adding also forget (i.e., decrease mass profiles of nodes) updates, proving the need of forgetting mechanism even in that case of model updates. The idea of mass updates is very simple. We suggest researchers to be experimented with different criteria (such as angle or density) to estimate the goodness of a feature region.

- Closing, online distance-based detectors such as MCOD are much more dependent on their hyper-parameters than tree-based ones. We also showed that from tree-based detectors RRCF has the less hyper-parameters. Essentially, the max height on HST/F indicates the depth (same) for all feature regions and the forget threshold on RRCF indicates the max number of leaf nodes.

Acknowledgements

This project was partially supported by SAP company (France, Paris 92309 Levallois Perret, +33 1 46 17 70 00).

Bibliography

[1] Charu C. Aggarwal and Saket Sathe. *Outlier Ensembles - An Introduction*. Springer, 2017. ISBN: 978-3-319-54764-0. DOI: 10.1007/978-3-319-54765-7. URL: https://doi.org/10.1007/978-3-319-54765-7.

[2] Charu C. Aggarwal and Saket Sathe. "Theoretical Foundations and Algorithms for Outlier Ensembles." In: *SIGKDD Explorations* 17.1 (2015), pp. 24–47. DOI: 10.1145/2830544.2830549. URL: https://doi.org/10.1145/2830544.2830549.

[3] Subutai Ahmad et al. "Unsupervised real-time anomaly detection for streaming data." In: *Neurocomputing* 262 (2017), pp. 134–147. DOI: 10.1016/j.neucom.2017.04.070. URL: https://doi.org/10.1016/j.neucom.2017.04.070.

[4] Tyler Akidau et al. "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing." In: *PVLDB* 8.12 (2015), pp. 1792–1803. DOI: 10.14778/2824032.2824076. URL: http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf.

[5] Peter Bailis, Deepak Narayanan, and Samuel Madden. "MacroBase: Analytic Monitoring for the Internet of Things." In: *CoRR* abs/1603.00567 (2016). arXiv: 1603.00567. URL: http://arxiv.org/abs/1603.00567.

[6] Matthew Bartos, Abhiram Mullapudi, and Sara C. Troutman. "rrcf: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams." In: *J. Open Source Software* 4.35 (2019), p. 1336. DOI: 10.21105/joss.01336. URL: https://doi.org/10.21105/joss.01336.

[7] Christoph Bergmeir and José Manuel Benıtez. "On the use of cross-validation for time series predictor evaluation." In: *Inf. Sci.* 191 (2012), pp. 192–213. DOI: 10.1016/j.ins.2011.12.028. URL: https://doi.org/10.1016/j.ins.2011.12.028.

[8] Sanjay Sharma Bhagyashri Karkhanis. "Outlier Detection in High Dimensional Data Streams to Detect Lower Subspace Outliers Effectively." In: *IJERT* 08 (2019). DOI: 2278-0181. URL: https://www.ijert.org/outlier-detection-in-high-dimensional-data-streams-to-detect-lower-subspace-outliers-effectively.

[9] Paula Branco, Luués Torgo, and Rita P. Ribeiro. "A Survey of Predictive Modelling under Imbalanced Distributions." In: *CoRR* abs/1505.01658 (2015). arXiv: 1505.01658. URL: http://arxiv.org/abs/1505.01658.

[10] Markus M. Breunig et al. "LOF: Identifying Density-Based Local Outliers." In: *SIGMOD Rec.* 29.2 (May 2000), pp. 93–104. ISSN: 0163-5808. DOI: 10.1145/335191.335388. URL: https://doi.org/10.1145/335191.335388.

[11] Guilherme O. Campos et al. "On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study." In: *Data Min. Knowl. Discov.* 30.4 (July 2016), pp. 891–927. ISSN: 1384-5810. DOI: 10.1007/s10618-015-0444-8. URL: https://doi.org/10.1007/s10618-015-0444-8.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." In: *ACM Comput. Surv.* 41.3 (2009), 15:1–15:58. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[13] Paolo Collela. *5G and IoT: Ushering in a new era.* https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/5g-and-iot-ushering-in-a-new-era. 2020.

[14] Thomas M. Cover. "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition." In: *IEEE Trans. Electronic Computers* 14.3 (1965), pp. 326–334. DOI: 10.1109/PGEC.1965.264137. URL: https://doi.org/10.1109/PGEC.1965.264137.

[15] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006.* Vol. 148. ACM International Conference Proceeding Series. ACM, 2006, pp. 233–240. DOI: 10.1145/1143844.1143874. URL: https://doi.org/10.1145/1143844.1143874.

[16] Janez Demsar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *J. Mach. Learn. Res.* 7 (2006), pp. 1–30. URL: http://jmlr.org/papers/v7/demsar06a.html.

[17] Remi Domingues et al. "A comparative evaluation of outlier detection algorithms: Experiments and analyses." In: *Pattern Recognit.* 74 (2018), pp. 406–421. DOI: 10.1016/j.patcog.2017.09.037. URL: https://doi.org/10.1016/j.patcog.2017.09.037.

[18] Rmi Domingues et al. "A Comparative Evaluation of Outlier Detection Algorithms." In: *Pattern Recogn.* 74.C (Feb. 2018), pp. 406–421. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2017.09.037. URL: https://doi.org/10.1016/j.patcog.2017.09.037.

[19] Dheeru Dua and Casey Graff. "UCI Machine Learning Repository." In: (2017). URL: http://archive.ics.uci.edu/ml.

[20] Sahibsingh A. Dudani. "The Distance-Weighted k-Nearest-Neighbor Rule." In: *IEEE Trans. Systems, Man, and Cybernetics* 6.4 (1976), pp. 325–327. DOI: 10.1109/TSMC.1976.5408784. URL: https://doi.org/10.1109/TSMC.1976.5408784.

[21] Andrew Emmott et al. *A Meta-Analysis of the Anomaly Detection Problem.* 2015. arXiv: 1503.01158 [cs.AI].

[22] Tom Fawcett. "An introduction to ROC analysis." In: *Pattern Recognit. Lett.* 27.8 (2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010. URL: https://doi.org/10.1016/j.patrec.2005.10.010.

[23] Michail Giannoulis. *Macrobase - Benchmarking Online and Offline Anomaly Detectors in Streaming Manner.* https://github.com/mgiannoulis. 2020.

[24] Nicolas Goix et al. "One Class Splitting Criteria for Random Forests." In: *CoRR* abs/1611.01971 (2016). arXiv: 1611.01971. URL: http://arxiv.org/abs/1611.01971.

[25] Oleg N. Granichin, Zeev Volkovich, and Dvora Toledano-Kitai. *Randomized Algorithms in Automatic Control and Data Mining.* Vol. 67. Intelligent Systems Reference Library. Springer, 2015. ISBN: 978-3-642-54785-0. DOI: 10.1007/978-3-642-54786-7. URL: https://doi.org/10.1007/978-3-642-54786-7.

[26] Sudipto Guha et al. "Robust Random Cut Forest Based Anomaly Detection on Streams." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016.* Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 2712–2721. URL: http://proceedings.mlr.press/v48/guha16.html.

[27] Manish Gupta et al. "Outlier Detection for Temporal Data: A Survey." In: *IEEE Trans. Knowl. Data Eng.* 26.9 (2014), pp. 2250–2267. DOI: 10.1109/TKDE.2013.184. URL: https://doi.org/10.1109/TKDE.2013.184.

[28] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* Springer Series in Statistics. Springer, 2009. ISBN: 9780387848570. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7.

[29] D. M. Hawkins. *Identification of Outliers.* Monographs on Applied Probability and Statistics. Springer, 1980. ISBN: 978-94-015-3996-8. DOI: 10.1007/978-94-015-3994-4. URL: https://doi.org/10.1007/978-94-015-3994-4.

[30] Victoria J. Hodge and Jim Austin. "A Survey of Outlier Detection Methodologies." In: *Artif. Intell. Rev.* 22.2 (2004), pp. 85–126. DOI: 10.1023/B:AIRE.0000045502.10941.a9. URL: https://doi.org/10.1023/B:AIRE.0000045502.10941.a9.

[31] Nathalie Japkowicz, Catherine Myers, and Mark A. Gluck. "A Novelty Detection Approach to Classification." In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1995, pp. 518–523. URL: http://ijcai.org/Proceedings/95-1/Papers/068.pdf.

[32] Fabian Keller, Emmanuel Müller, and Klemens Böhm. "HiCS: High Contrast Subspaces for Density-Based Outlier Ranking." In: *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*. IEEE Computer Society, 2012, pp. 1037–1048. DOI: 10.1109/ICDE.2012.88. URL: https://doi.org/10.1109/ICDE.2012.88.

[33] Ron Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection." In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1995, pp. 1137–1145. URL: http://ijcai.org/Proceedings/95-2/Papers/016.pdf.

[34] Maria Kontaki et al. "Continuous monitoring of distance-based outliers over data streams." In: *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*. IEEE Computer Society, 2011, pp. 135–146. DOI: 10.1109/ICDE.2011.5767923. URL: https://doi.org/10.1109/ICDE.2011.5767923.

[35] Maria Kontaki et al. *Distance-based outlier detection in data streams - Repository*. http://infolab.usc.edu/Luan/Outlier. 2011.

[36] Hans-Peter Kriegel et al. "Interpreting and Unifying Outlier Scores." In: *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*. SIAM / Omnipress, 2011, pp. 13–24. DOI: 10.1137/1.9781611972818.2. URL: https://doi.org/10.1137/1.9781611972818.2.

[37] Zimek. Kriegel Kroger. *Outlier Detection Techniques*. https://www.dbs.ifi.lmu.de/~zimek/publications/KDD2010/kdd10-outlier-tutorial.pdf. 2010.

[38] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, 2013. ISBN: 978-1-4614-6849-3. DOI: 10.1007/978-1-4614-6849-3. URL: https://doi.org/10.1007/978-1-4614-6849-3.

[39] Alexander Lavin and Subutai Ahmad. "Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark." In: *CoRR* abs/1510.03336 (2015). arXiv: 1510.03336. URL: http://arxiv.org/abs/1510.03336.

[40] Alexander Lavin and Subutai Ahmad. *The Numenta Anomaly Benchmark (NAB)*. https://github.com/numenta/NAB. 2015.

[41] Erich Leo Lehmann and Joseph P. Romano. *Testing Statistical Hypotheses, Third Edition*. Springer texts in statistics. Springer, 2008. ISBN: 978-0-387-98864-1.

[42] Anthony Li. *Fast Anomaly Detection for Streaming Data - Repository*. https://github.com/yli96/HSTree. 2018.

[43] Jin Li et al. "Semantics and Evaluation Techniques for Window Aggregates in Data Streams." In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. ACM, 2005, pp. 311–322. DOI: 10.1145/1066157.1066193. URL: https://doi.org/10.1145/1066157.1066193.

[44] Ariel Linden. "Measuring diagnostic and predictive accuracy in disease management: an introduction to receiver operating characteristic (ROC) analysis." In: *Journal of Evaluation in Clinical Practice* 12.2 (Apr. 2006), pp. 132–139. DOI: 10.1111/j.1365-2753.2005.00598.x. URL: https://doi.org/10.1111%2Fj.1365-2753.2005.00598.x.

[45] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest." In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*. IEEE Computer Society, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17. URL: https://doi.org/10.1109/ICDM.2008.17.

[46] Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. "AUC: a misleading measure of the performance of predictive distribution models." In: *Global Ecology and Biogeography* 17.2 (Mar. 2008), pp. 145–151. DOI: 10.1111/j.1466-8238.2007.00358.x. URL: https://doi.org/10.1111%2Fj.1466-8238.2007.00358.x.

[47] A. Bifet M. Harries J. Gama. *Electricity Dataset*. https://datahub.io/machine-learning/electricity. 2014.

[48] Matt. *Vertatique How Many Billion IoT Devices by 2020?* http://www.vertatique.com/50-billion-connected-devices-2020. 2020.

[49] Hasan Önder. "A Comparative Study of Permutation Tests with Euclidean and Bray-Curtis Distances for Common Agricultural Distributions in Regression." In: *Journal of Applied Animal Research* 34.2 (2008), pp. 133–136. DOI: 10.1080/09712119.2008.9706957. eprint: https://doi.org/10.1080/09712119.2008.9706957. URL: https://doi.org/10.1080/09712119.2008.9706957.

[50] Gustavo Henrique Orair et al. "Distance-Based Outlier Detection: Consolidation and Renewed Bearing." In: *PVLDB* 3.2 (2010), pp. 1469–1480. DOI: 10.14778/1920841.1921021. URL: http://www.vldb.org/pvldb/vldb2010/pvldb%5C_vol3/I09.pdf.

[51] Aleksas Pantechovskis. "Determining Criteria for Choosing Anomaly Detection Algorithm." Master's Thesis. LT-44404 Kaunas: Vytautas Magnus University Faculty of Informatics Department of Applied Informatics, 2019.

[52] Aleksas Pantechovskis. *Macrobase*. https://github.com/anomaly-detection-macrobase-benchmark/macrobase. 2019.

[53] Animesh Patcha and Jung-Min Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends." In: *Comput. Networks* 51.12 (2007), pp. 3448–3470. DOI: 10.1016/j.comnet.2007.02.001. URL: https://doi.org/10.1016/j.comnet.2007.02.001.

[54] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830. URL: http://dl.acm.org/citation.cfm?id=2078195.

[55] Tomás Pevný. "Loda: Lightweight on-line detector of anomalies." In: *Mach. Learn.* 102.2 (2016), pp. 275–304. DOI: 10.1007/s10994-015-5521-0. URL: https://doi.org/10.1007/s10994-015-5521-0.

[56] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient Algorithms for Mining Outliers from Large Data Sets." In: *SIGMOD Rec.* 29.2 (May 2000), pp. 427–438. ISSN: 0163-5808. DOI: 10.1145/335191.335437. URL: https://doi.org/10.1145/335191.335437.

[57] LA Rastrigin. "The convergence of the random search method in the extremal control of a many parameter system." In: *Automaton & Remote Control* 24 (1963), pp. 1337–1342.

[58] Shebuti Rayana. *ODDS Library*. http://odds.cs.stonybrook.edu. 2016.

[59] David M. Rocke and David L. Woodruff. "Identification of Outliers in Multivariate Data." In: *Journal of the American Statistical Association* 91.435 (1996), pp. 1047–1061. DOI: 10.1080/01621459.1996.10476975. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1996.10476975. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.199.10476975.

[60] Jeremy Rogers and Steve R. Gunn. "Identifying Feature Relevance Using a Random Forest." In: *Subspace, Latent Structure and Feature Selection, Statistical and Optimization, Perspectives Workshop, SLSFS 2005, Bohinj, Slovenia, February 23-25, 2005, Revised Selected Papers*. Vol. 3940. Lecture Notes in Computer Science. Springer, 2005, pp. 173–184. DOI: 10.1007/11752790_12. URL: https://doi.org/10.1007/11752790%5C_12.

[61] Takaya Saito and Marc Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets." In: *PLOS ONE* 10.3 (Mar. 2015), e0118432. DOI: 10.1371/journal.pone.0118432. URL: https://doi.org/10.1371%2Fjournal.pone.0118432.

[62] *SAP Computer Software Company*. SAP France. 1972. URL: https://www.sap.com/corporate/en/company/office-locations/france.html.

[63] "Manhattan Distance." In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar and Hui Xiong. Springer, 2008, p. 631. DOI: 10.1007/978-0-387-35973-1_733. URL: https://doi.org/10.1007/978-0-387-35973-1%5C_733.

[64] "Euclidean Distance." In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar, Hui Xiong, and Xun Zhou. Springer, 2017, p. 556. DOI: 10.1007/978-3-319-17885-1_100372. URL: https://doi.org/10.1007/978-3-319-17885-1%5C_100372.

[65] Bas van Stein, Matthijs van Leeuwen, and Thomas Bäck. "Local Subspace-Based Outlier Detection using Global Neighbourhoods." In: *CoRR* abs/1611.00183 (2016). arXiv: 1611.00183. URL: http://arxiv.org/abs/1611.00183.

[66] Swee Chuan Tan, Kai Ming Ting, and Fei Tony Liu. "Fast Anomaly Detection for Streaming Data." In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. IJCAI/AAAI, 2011, pp. 1511–1516. DOI: 10.5591/978-1-57735-516-8/IJCAI11-254. URL: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-254.

[67] Swee Chuan Tan, Kai Ming Ting, and Fei Tony Liu. *Mass Estimation*. http://mass-estimation.sourceforge.net/. 2014.

[68] Kai Ming Ting et al. "Defying the gravity of learning curve: a characteristic of nearest neighbour anomaly detectors." In: *Mach. Learn.* 106.1 (2017), pp. 55–91. DOI: 10.1007/s10994-016-5586-4. URL: https://doi.org/10.1007/s10994-016-5586-4.

[69] Luan Tran, Liyue Fan, and Cyrus Shahabi. "Distance-based Outlier Detection in Data Streams." In: *PVLDB* 9.12 (2016), pp. 1089–1100. DOI: 10.14778/2994509.2994526. URL: http://www.vldb.org/pvldb/vol9/p1089-tran.pdf.

[70] Joaquin Vanschoren et al. "OpenML: networked science in machine learning." In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. DOI: 10.1145/2641190.2641198. URL: https://doi.org/10.1145/2641190.2641198.

[71] Sudhir Varma and Richard Simon. "Bias in error estimation when using cross-validation for model selection." In: *BMC Bioinform.* 7 (2006), p. 91. DOI: 10.1186/1471-2105-7-91. URL: https://doi.org/10.1186/1471-2105-7-91.

[72] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. "Progress in Outlier Detection Techniques: A Survey." In: *IEEE Access* 7 (2019), pp. 107964–108000. DOI: 10.1109/ACCESS.2019.2932769. URL: https://doi.org/10.1109/ACCESS.2019.2932769.

[73] Ke Wu et al. "RS-Forest: A Rapid Density Estimator for Streaming Anomaly Detection." In: *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*. IEEE Computer Society, 2014, pp. 600–609. DOI: 10.1109/ICDM.2014.45. URL: https://doi.org/10.1109/ICDM.2014.45.

[74] Ethan Zhang and Yi Zhang. "Average Precision." In: *Encyclopedia of Database Systems*. Springer US, 2009, pp. 192–193. DOI: 10.1007/978-0-387-39940-9_482. URL: https://doi.org/10.1007/978-0-387-39940-9%5C_482.

[75] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. "A survey on unsupervised outlier detection in high-dimensional numerical data." In: *Statistical Analysis and Data Mining* 5.5 (2012), pp. 363–387. DOI: 10.1002/sam.11161. URL: https://doi.org/10.1002/sam.11161.

[76] Arthur Zimek et al. "Subsampling for efficient and effective unsupervised outlier detection ensembles." In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*. ACM, 2013, pp. 428–436. DOI: 10.1145/2487575.2487676. URL: https://doi.org/10.1145/2487575.2487676.